Level Set and PDE Methods for Computer Graphics

Notes for SIGGRAPH 2004 Course #27
Los Angeles, CA
August 10, 2004

Organizer

David Breen — Drexel University

Speakers

David Breen	Drexel University
Ron Fedkiw	Stanford University
Ken Museth	Linköping University
Stanley Osher	University of California, Los Angeles
Guillermo Sapiro	University of Minnesota
Ross Whitaker	University of Utah

Course Abstract

Level set methods, an important class of partial differential equation (PDE) methods, define dynamic surfaces implicitly as the level set (iso-surface) of a sampled, evolving nD function. The course begins with preparatory material that introduces the concept of using partial differential equations to solve problems in computer graphics, geometric modeling and computer vision. This will include the structure and behavior of several different types of differential equations, e.g. the level set equation and the heat equation, as well as a general approach to developing PDE-based applications. The second stage of the course will describe the numerical methods and algorithms needed to actually implement the mathematics and methods presented in the first stage. The course closes with detailed presentations on several level set/PDE applications, including image/video inpainting, pattern formation, image/volume processing, 3D shape reconstruction, image/volume segmentation, image/shape morphing, geometric modeling, anisotropic diffusion, and natural phenomena simulation.

Prerequisites

Knowledge of calculus, linear algebra, computer graphics, geometric modeling, image processing and computer vision. Some familiarity with differential geometry, differential equations, numerical computing and image processing is strongly recommended, but not required.

Speaker Biographies

David Breen is an Assistant Professor in the Computer Science Department at Drexel University. He has held research positions at the Center for Advanced Computing Research and the Computer Graphics Lab at the California Institute of Technology, the European Computer-Industry Research Centre, the Fraunhofer Institute for Computer Graphics, and the Rensselaer Design Research Center. His research interests include level set models for computer graphics, volume segmentation, large data

visualization, and geometric modeling. He has published over 50 research papers in these and other areas, as well as the book *Cloth Modeling and Animation*. Breen received a B.A. in Physics (Colgate University, 1982), and a Ph.D. in Computer and Systems Engineering (Rensselaer Polytechnic Institute, 1993). E-mail: david@cs.drexel.edu

Ronald Fedkiw received his Ph.D. in Mathematics from UCLA in 1996 and did postdoctoral studies both at UCLA in Mathematics and at the California Institute of Technology in Aeronautics before joining the Stanford Computer Science Department. He was awarded a Packard Foundation Fellowship, a Presidential Early Career Award for Scientists and Engineers (PECASE), an Office of Naval Research Young Investigator Program Award (ONR YIP), a Robert N. Noyce Family Faculty Scholarship, and two distinguished teaching awards. He is on the editorial board of the Journal of Scientific Computing, IEEE Transactions on Visualization and Computer Graphics, and Communications in Mathematical Sciences, and participates in the reviewing process for a number of journals and funding agencies. He has published over 45 research papers in computational physics, computer graphics and vision, as well as a new book on level set methods. Additionally, he has been a consultant for Industrial Light + Magic for the last three years.
E-mail: fedkiw@cs.stanford.edu

Ken Museth recently joined the faculty at Linköping University as a Professor of Computer Graphics. He received his Ph.D. in computational quantum dynamics from the University of Copenhagen in 1997 and came to the California Institute of Technology as a visiting faculty in 1998. While at Caltech he shifted into the computer science field and joined the Computer Graphics Lab as a research scientist in 2000, followed by the Center for Advanced Computing Research in 2002. He has also been a scientific consultant to Digital Domain and NASA's Jet Propulsion Laboratory. His research activities focus on the areas of deforming geometry and level set methods. E-mail: kenmu@itn.liu.se

Stanley Osher has been a Professor of Mathematics at UCLA since 1977, where he is the Director of the Applied Mathematics Program and the Director of Special Projects at the Institute for Pure and Applied Mathematics. He is co-inventor of the Level Set Method, TV based image restoration and high resolution nonoscillatory methods (ENO, WENO). He works in the area of scientific computing with applications to many areas,

including image processing, computer vision and graphics. From 1988-95 he was cofounder and co-CEO of Cognitech, Inc. Since 1998 he has been President of Level Set Systems, Inc. He has been both a Fulbright and an Alfred P. Sloan Fellow. He received a NASA Public Service Award, the 2002 Japan Society of Mechanical Engineers Computational Mechanics Award, the 2003 Society of Industrial and Applied Mathematics Pioneer Prize, and is an original ISI Highly Cited Researcher. Osher has co-authored and co-edited two books on Level Set Methods.
E-mail: sjo@math.ucla.edu

Guillermo Sapiro is a Professor with the Department of Electrical and Computer Engineering at the University of Minnesota. He works on differential geometry and geometric partial differential equations, both in theory and applications in computer vision, computer graphics, and medical imaging. Sapiro has recently written a book in the area and co-edited special issues in the IEEE Transactions on Image Processing and the Journal of Visual Communication and Image Representation. He has received the Gutwirth Scholarship for Special Excellence in Graduate Studies, the Ollendorff Fellowship for Excellence in Vision and Image Understanding, the Rothschild Fellowship for Post-Doctoral Studies, the ONR Young Investigator Award, the PECASE Award, and the NSF Career Award. E-mail: guille@ece.umn.edu

Ross Whitaker is an Associate Professor in the School of Computing at the University of Utah and is affiliated with the Institute for Scientific Computing and Imaging. He received a B.S. in Electrical Engineering and Computer Science (Princeton University, 1986) and a Ph.D. in Computer Science (University of North Carolina, Chapel Hill, 1993). He was a research staff member at the European Computer-Industry Research Centre, and an Assistant Professor at the University of Tennessee. He teaches and conducts research in computer vision, image processing, medical imaging, and computer graphics/visualization. He has published numerous papers and book chapters on PDE/level set methods for image processing and computer graphics. E- mail: whitaker@cs.utah.edu

Course Schedule

Session 1 - PDE and Level Set Fundamentals
- 8:30 Welcome - Breen
- 8:40 Introduction to PDEs and Their Application to Imaging - Sapiro
- 9:40 Introduction to Level Set Methods - Osher

10:15 Break

Session 2 – Numerical Methods and Applications
- 10:30 Dynamic Visibility in an Implicit Framework - Osher
- 11:00 Level Set Numerical Methods - Whitaker
- 11:25 Level Set Surface Reconstruction and Processing - Whitaker
- 11:55 Level Set Methods on a Streaming Architecture - Whitaker

12:15 Lunch Break

Session 3 – PDE/Level Set Applications
- 1:45 Image Inpainting - Sapiro
- 2:15 Computing Generalized Geodesics for Computer Graphics - Sapiro
- 2:45 Algorithms for Level Set Modeling - Museth
- 3:10 Level Set Surface Editing Operators - Museth

3:30 Break

Session 4 - Level Set Segmentation and Simulation
- 3:45 3D Volume Segmentation (framework, multiple non-uniform datasets, diffusion tensor MRI, sinograms) - Breen
- 4:30 Simulation of Water, Fire and Smoke - Fedkiw

5:30 Course Ends

Web Sites

Osher Home Page
http://www.math.ucla.edu/~sjo

UCLA CAM Technical Reports
http://www.math.ucla.edu/applied/cam

Level Set Systems, Inc.
http://www.levelset.com

Sapiro Home Page
http://www.ece.umn.edu/users/guille

Whitaker Home Page
http://www.cs.utah.edu/~whitaker

VISPack Web Site
http://www.cs.utah.edu/~whitaker/vispack

Fedkiw Home Page
http://www.graphics.stanford.edu/~fedkiw

Museth Home Page
http://gg.itn.liu.se

Breen - Geometric Modeling and Deformable Models
http://www.cs.drexel.edu/~david/geom_mod.html
http://www.cs.drexel.edu/~david/deform_mod.html

Sethian Home Page
http://www.math.berkeley.edu/~sethian

Citations For Included Papers

- D. Breen, R. Whitaker, K. Museth and L. Zhukov, "Level Set Segmentation of Biological Volume Datasets," to appear in the *Handbook of Medical Image Analysis*, 2004.
- D. Enright, S. Marschner and R. Fedkiw, "Animation and Rendering of Complex Water Surfaces," *Proc. SIGGRAPH 2002 Conference*, pp. 736-744, July 2002.
- R. Fedkiw, G. Sapiro, and C.-W. Shu, "Shock Capturing, Level Sets, and PDE Based Methods in Computer Vision and Image Processing: A Review of Osher's Contributions," *Journal of Computational Physics*, Vol. 185, No. 2, pp. 309 – 341, March 2003. Also published as UCLA CAM Report # 01-33.
- N. Foster and R. Fedkiw, "Practical Animation of Liquids," *Proc. SIGGRAPH 2001 Conference*, pp. 23-30, July 2001.
- A.E. Lefohn, J.M. Kniss, C.D. Hansen and R.T. Whitaker, "A Streaming Narrow-Band Algorithm: Interactive Computation and Visualization of Level Sets," to appear in *IEEE Transactions on Visualization and Computer Graphics*, 2004.
- F. Losasso, F. Gibou and R. Fedkiw, "Simulating Water and Smoke with an Octree Data Structure," *Proc. SIGGRAPH 2004 Conference*, August 2004.
- K. Museth, D.E. Breen, R.T. Whitaker and A.H. Barr, "Level Set Surface Editing Operators," *Proc. SIGGRAPH 2002 Conference*, pp. 330-338, July 2002.
- D.Q. Nguyen, R. Fedkiw and H.W. Jensen, "Physically Based Modeling and Animation of Fire," *Proc. SIGGRAPH 2002 Conference*, pp. 721-728, July 2002.
- S. Osher and R. Fedkiw, "Level Set Methods: An Overview and Some Recent Results," *Journal of Computational Physics*, Vol. 169, pp. 475-502, 2001. Also published as UCLA CAM Report # 00-08.
- T. Tasdizen, R. Whitaker, P. Burchard and S. Osher, "Geometric Surface Processing via Normal Maps," *ACM Transactions on Graphics*, Vol. 22, No. 4, pp. 1012-1033, October 2003.
- R. Tsai, P. Burchard, L.-T. Cheng, S. Osher and G. Sapiro, "Dynamic Visibility in an Implicit Framework," UCLA CAM Report # 02-06.
- H.-K. Zhao, S. Osher and R. Fedkiw, "Fast Surface Reconstruction Using the Level Set Method," *Proc. 1st IEEE Workshop on Variational and Level Set Methods,* pp. 194–202, 2001. Also published as UCLA CAM Report # 01-01.

Related Books

S. Osher and R. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*, Springer, New York, 2002.

S. Osher and N. Paragios (eds.), *Geometric Level Set Methods in Imaging, Vision and Graphics*, Springer, New York, 2003.

G. Sapiro, *Geometric Partial Differential Equations and Image Analysis*, Cambridge University Press, 2001.

J. Sethian, *Level Set Methods and Fast Marching Methods,* Cambridge University Press, 1999

Table of Contents

Session 3 – PDE/Level Set Applications

Image Inpainting: An Overview (Slides)
G. Sapiro
The Art of Distance and Geodesic Computations (Slides)
G. Sapiro
Distance Functions and Geodesics on Point Clouds (Slides)
G. Sapiro
Level Set Surface Editing Operators
K. Museth, D. Breen, R. Whitaker and A. Barr
Algorithms for Interactive Editing of Level Set Models
K. Museth, D. Breen, R. Whitaker, S. Mauch and D. Johnson

Session 4 – Level Set Segmentation and Simulation

Level Set Segmentation of Biological Volume Datasets
D. Breen, R. Whitaker, K. Museth and L. Zhukov
Physically Based Modeling and Animation of Fire
D. Nguyen, R. Fedkiw and H. Jensen
Practical Animation of Liquids
N. Foster and R. Fedkiw
Animation and Rendering of Complex Water Surfaces
D. Enright, S. Marschner and R. Fedkiw
Simulating Water and Smoke with an Octree Data Structure
F. Losasso, F. Gibou and R. Fedkiw

Session 1

PDE and Level Set Fundamentals

SIGGRAPH 2002

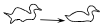
PDE's in Image Processing, Computer Vision, and Computer Graphics

Guillermo Sapiro
Electrical and Computer Engineering
University of Minnesota
guille@ece.umn.edu

Supported by NSF, ONR, NIH

Moving Curves

Basic curve evolution

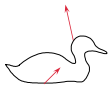


- Planar curve:

$$C(p):[0,1] \rightarrow R^2$$

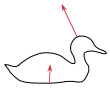
- General flow:

$$\frac{\partial C}{\partial t} = \alpha \vec{T} + \beta \vec{N}$$



- General geometric flow:

$$\frac{\partial C}{\partial t} = \beta \vec{N}$$



Mathematical morphology

- Classical theory, based on Minkowsky addition.
- The old and (probably wrong) way of doing geometric image analysis.
- Has very important lessons to learn!!!!
- Basic definitions:
  - *A*: Image in Euclidean space (*R* or *Z*)
  - *B*: Structuring element (symmetric)
  - Nothing else than Minkowsky addition

Mathematical morphology: Definitions

$$A \oplus B := \{a+b, a \in A, b \in B\} = \bigcup_{b \in B} A_b$$

$$A \Theta B := (A^c \oplus B)^c = \bigcap_{b \in B} A_b$$

$$A \circ B := (A \Theta B) \oplus B = \bigcup_{\{y : B_y \subseteq A\}} A_y$$

$$A \bullet B := (A \oplus B) \Theta B$$

Mathematical morphology: Is it good or bad?

- Advantages:
  - Nice mathematical properties (set theory)
  - Extension to Lattices
- Disadvantages:
  - Discrete Minkowsky addition does not look good, has to be replaced by better ways of computing "discrete distances."
- Major important concept: Level-sets

$$f: R^N \rightarrow R, \qquad g: R^N \rightarrow \{0, -\infty\}$$

$$f \oplus g = \max_{\text{support of } g} \{f\}$$

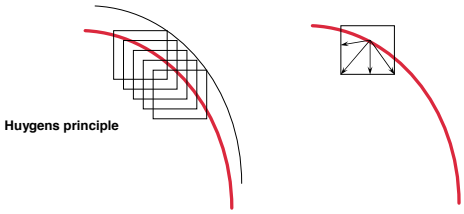
  - Commutes with thresholding (level-sets): Do binary on each level sets or do gray-level on all the image => same result
- It is in certain sense a particular case of curve evolution (before the lattices part)

Mathematical morphology via curve evolution

Convex structuring elements B:

$$A \oplus (B \oplus C) = (A \oplus B) \oplus C$$

$$A \oplus (rB) = A \oplus r_1 B \oplus r_2 B \oplus \ldots$$



Mathematical morphology via curve evolution (cont.)

- General velocity:

$$\beta = \sup_{\theta}\{r(\theta)\cdot N\}$$

- Examples:

$\beta = N$	$B = disk$
$\beta = \max\{N_x, N_y\}$	$B = diamond$
$\beta = \lvert N_x\rvert + \lvert N_y\rvert$	$B = square$

- Nothing else than changing the metric (distance).
- Can be explained also based on dynamic programming and time of arrival
- See Sapiro et al., Brocket-Maragos, Alvarez et al., Evans, Falcone

Planar differential geometry

- Euclidean invariant parametrization

$$\left\| \frac{\partial C(s)}{\partial s} \right\| = 1$$

$$< C_s, C_{ss} > = 0$$

$$C_s \perp C_{ss}$$

$$\kappa := \|C_{ss}\|$$

- Affine invariant parametrization

$$\left\| \frac{\partial C(s)}{\partial s}, \frac{\partial^2 C(s)}{\partial s^2} \right\| = 1$$

$$\|C_s, C_{sss}\| = 0$$

$$\kappa C_s + C_{sss} = 0$$

$$\kappa = \|C_{ss}, C_{sss}\|$$

Planar differential geometry (cont.)

- Curvature constant for circles or straight lines (=0)
- Curvature defines curve up to Euclidean motion
- At least 4 points with *dk/ds=0*
- Defined for all curves

- Curvature constant for ellipses (>0), hyperbolas (<0), and parabolas (=0)
- Curvature defines curve up to affine motion
- At least 6 points with *dk/ds=0*
- Defined only for convex curves: segment at inflection points

Planar differential geometry (cont.)

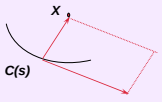
$$d(X, C(s)) := < X - C(s), X - C(s) >$$

$$\frac{\partial d}{\partial s} = < X - C, C_s >$$

$$\Downarrow (=0)$$

$$X - C(s) \perp C_s(s)$$
$$X - C(s) \parallel C_{ss}(s)$$

$$d(X, C(s)) := \|X - C(s), C_s(s)\|$$



$$\frac{\partial d}{\partial s} = \|X - C(s), C_{ss}(s)\|$$

$$\Downarrow (=0)$$

$$X - C(s) \parallel C_{ss}(s)$$

Distance has a local extrema iff X is on the norma

3D Differential geometry

- Remember mean and Gaussian curvatures?
- Each regular surface has two principal curvatures. The average is the mean curvature, the product the Gaussian. These are also related to the tangential map, etc, etc. See DoCarmo for details.

Riemannian geometry, Lie theory

- What about other non-Euclidean metrics?
- What about invariants to other (Lie) groups, e.g., projective?
- What about differential invariants? Semi-differential invariants? Are there any general theories?

Smoothing by classical heat flow

$$\frac{\partial C}{\partial t} = \Delta C \quad \Leftrightarrow \quad \begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{bmatrix} x_{pp} \\ y_{pp} \end{bmatrix}$$

- Linear
- Equivalent to Gaussian filtering
- Unique linear scale-space

- Non geometric
- Shrinks the shape
- Implementation problems

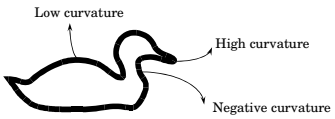
Invariant shape deformations

- Formulate shape deformations
  - Geometric
  - Invariant to camera transform
  - The "best" possible
  - Change only the desired features
- Motivation:
  - Mathematics:
    - From static differential geometry to dynamic
    - Beautiful
  - Computer vision and image processing:
    - Invariant shape segmentation and analysis
    - Image processing via image deformations
  - Robotics:
    - Motion planning
    - Accurate geometric object detection and tracking
    - Robot manipulation and grasping

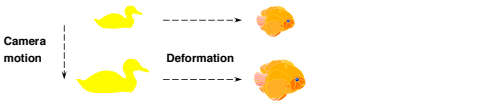
Basic planar differential geometry

- For every Lie group we will consider, exists and invariant parametrization s, the group *arc-length*

- For every such a group exists and *invariant signature*, the *group curvature*, *k*



What and why invariant



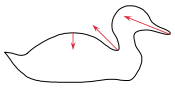
- Camera/object movement in the space

- Transformations description (for "flat" objects):
  - Euclidean
    - Motion parallel to the camera and planar projection
  - Affine
    - Planar projection
  - Projective

Euclidean geometric heat flow

- Use the Euclidean arc-length: $\|C_s\| = 1$
- The deformation:

$$\frac{\partial C}{\partial t} = \frac{\partial^2 C}{\partial s^2} = \kappa \vec{N}$$



- Smoothly deforms to a circle (Gage-Hamilton, Grayson)
- Geometric smoothing
- Reduces length as fast as possible

Affine geometric heat flow (Sapiro-Tannenbaum)

- Use the affine arc-length: $\left\| C_s \times C_{ss} \right\| = 1$
- The flow:

$$C_{ss} = \kappa^{1/3} \vec{N} + f(\kappa, \dot{\kappa}) \vec{T}$$

$$\frac{\partial C}{\partial t} = \begin{cases} C_{ss} & \text{non - inflection} \\ 0 & \text{inflection} \end{cases}$$

$$C_t = \kappa^{1/3} \vec{N}$$

Affine geometric heat flow (cont.)

- Geometric smoothing (preserving area if desired)
  - Total curvature decreases
  - Maxima of curvature decreases
  - Number of inflections decreases
- Smoothly deforms a shape into an ellipse
- Decreases area as fast as possible (in an affine form)
- Existence also for non-smooth curves
  - Viscosity framework (Alvarez-Guichard-Morel-Lions)
  - Polygons (Angenent-Sapiro-Tannenbaum)

- Applications:
  - Curvature computation for shape recognition: reduce noise (Morel et al.)
  - Simplify curvature computation (Faugeras '95)
  - Object recognition for robot manipulation (Cipolla '95)

General invariant flows

- Theorem: For every sub-group of the projective group the most general invariant curve deformation has the form

$$\frac{\partial C}{\partial t} = \frac{\partial^2 C}{\partial s^2} f(\kappa, \kappa_s, \kappa_{ss}, \ldots)$$

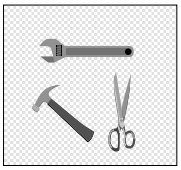
- Theorem: In general dimensions, the most general invariant flow is given by

$$u_t = \frac{g}{E(g)} f(curvatures)$$

  - *u*: graph locally representing the surface
  - *g*: invariant metric
  - *E(g)*: variational derivative of *g*

- See Olver *et al.*, Alvarez *et al.*, Caselles-Sbert

General Geometric Framework For Object Segmentation



Introduction

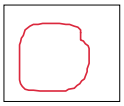
- Goal: Object detection

- Approach: Curve/surface deformation
  - Geometry dependent regularization
  - Image dependent velocity

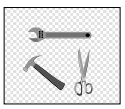
- Characteristics:
  - Unifies previously considered independent approaches
  - Relates segmentation with anisotropic diffusion
  - General:
    - Any topology
    - Any type of image data
    - Any dimension
  - Holds formal results

Notation

- Deforming curve: $C(p) : [0,1] \rightarrow R^2$



- Image: $I: [0,1]\times[0,1] \rightarrow R^2 \quad (R^N)$



Basic active contours approach

- Terzopoulos *et al.*, Cohen *et al.*

$$E(C) = \lambda \int |C'(p)|^2 \, dp + \gamma \int |C''(p)|^2 \, dp - \int |\nabla I(C)| \, dp$$

- Drawbacks:
  - Too many parameters
  - Non-geometric
  - Handling topology changes

Geodesic active contours (Caselles-Kimmel-Sapiro)

$$E(C) = \lambda \int |C'(p)|^2 \, dp + \gamma \int |C''(p)|^2 \, dp - \int |\nabla I(C)| \, dp$$

- Generalize image dependent energy
- Eliminate high order smoothness term
- Equal internal and external energies

$$E(C) = \int |C'(p)|^2 \, dp + \int g[|\nabla I(C(p))|] \, dp$$

- Maupertuis and Fermat principles of dynamical systems

$$E(C) = \int g[|\nabla I(C(s))|] \, ds$$

Geodesic computation

- Gradient-descent

$$E(C) = \int ds \quad \Rightarrow \quad \frac{\partial C}{\partial t} = \kappa \vec{N}$$

$$E(C) = \int g[|\nabla I(C(s))|] \, ds \quad \Rightarrow \quad \frac{\partial C}{\partial t} = g \kappa \vec{N} - \nabla g \cdot \vec{N}$$

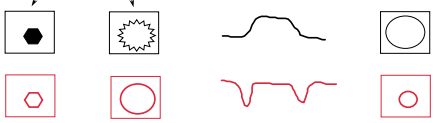
- Level-sets (Osher-Sethian)

$$\frac{\partial C}{\partial t} = \beta \vec{N} \quad \Rightarrow \quad \frac{\partial \Phi}{\partial t} = \beta |\nabla \Phi|$$

Further geometric interpretation

- The geodesic flow

$$\frac{\partial C}{\partial t} = g \kappa \vec{N} - \nabla g \cdot \vec{N} \qquad (+ g \vec{N})$$



Model correctness

- Theorem: The deformation is independent of the level-sets embedding function
- Theorem: There is a unique solution to the flow in the viscosity framework
- Theorem: The curve converges to ideal objects when present in the image
- Related work:
  - Kimia-Tannenbaum-Zucker
  - Caselles *et al.*
  - Malladi-Sethian-Vemuri
  - Kichenassamy *at al.*
  - Tek-Kimia, Whitacker
- New work:
  - Chan-Vese
  - Paragios-Deriche
  - Yezzi et al.
  - Faugeras et al.

Extensions

$$E(C) = \int g[|\nabla I(C(s))|] \, ds$$

- Gray-level values
  - *ds* - length element (geodesics)
  - Ordinary edge detector (gradient)
- Surfaces
  - *ds* - area element (minimal surfaces)
  - 3D edge detector
- Vector-valued images (color, texture, medical, etc)
  - *ds* - length element
  - Vector-valued edge detector (vector geodesics)
    - Eigenvalues of the first fundamental form in Riemannian space
- Invariant detection (affine area geodesics)
  - *ds* - affine length element (area related)
  - Affine invariant edge detector
  - Affine norm for "gradient descent"

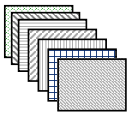
Why color edges?


Notation

- Image

$$X : [0,N]_x[0,N] \rightarrow \mathbf{R}^N$$

$L^*a^*b^*$ 

- Texture: Gabor decomposition 

Color edge computation

- Given a metric (Euclidean) $\Rightarrow \Delta X = \sqrt{\sum_i \Delta X_i^2}$
- Compute first fundamental form

$$[g_{ij}] = \frac{\partial X}{\partial i} \cdot \frac{\partial X}{\partial j}$$

- Compute eigenvectors and eigenvalues
- Edge: maximal eigenvalue and its eigenvector

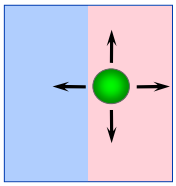
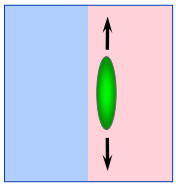
$$(\lambda_+, \lambda_-, \theta_+, \theta_-)$$

- Basic properties:
  - Eigenvectors are orthonormal
  - Minimal eigenvalue is not zero

Moving Images

Anisotropic diffusion

Isotropic vs. Anisotropic Smoothing

Isotropic smoothing

Anisotropic smoothing

Isotropic diffusion (Koenderink, Witkin)

- All "equivalent:"
  - Gaussian filtering of the image Φ
  - Heat flow

$$\frac{\partial \Phi}{\partial t} = \Delta \Phi$$

  - Minimize the L2 norm

$$\int \|\nabla \Phi\|^2$$

Isotropic diffusion: Good things

- Gaussian filtering if and only if
  - Linear
  - Shift-invariant
  - No creation of zero crossings
- Gaussian filtering if and only if
  - Linear
  - Shift-invariant
  - Semi-group property
  - Scale-invariant (dimensionless)
- Unique linear filter that defines a scale-space: Do not creates information at coarser scales
- Where everything started (Koenderink, Witkin)

Isotropic diffusion: Bad things and possible solutions

- Non-geometric
- Problems with implementations

- Who said linear? Replace heat flow by "parabolic" PDE's (Hummel's original idea)

- Why parabolic? Because of the maximum principle.

Perona-Malik anisotropic diffusion

- Replace the L2 by a different norm (e.g., L1, Rudin-Osher-Fatemi; Lorentzian, Black et. al.; etc)

$$\iint h(\|\nabla\Phi\|) \Rightarrow \frac{\partial \Phi}{\partial t} = \operatorname{div}\left(h'(I)\frac{\nabla I}{\|\nabla I\|}\right)$$

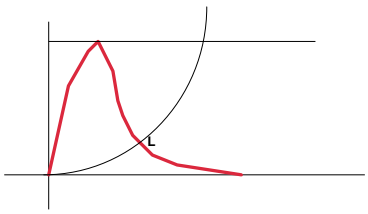
Selection of "stopping term" h

- How do we select h?
- h=x*x => L2 => linear => Isotropic diffusion
- h=x => L1 (Rudin-Osher-Fatemi)

$$\iint(\|\nabla\Phi\|) \Rightarrow \frac{\partial \Phi}{\partial t} = \kappa$$

Robust anisotropic diffusion

- General theory for selection "h", based on the theory of influence functions in robust statistics
- Edges should be considered outliers: At certain point, h', the influence, should be zero.



Directional diffusion

- Diffuse in the direction perpendicular to the edges (Avarez et al.)

$$\frac{\partial \Phi}{\partial t} = \kappa\|\nabla\Phi\| = \Phi_{\zeta\zeta}$$

$$\zeta \perp \nabla\Phi$$

From active contours to anisotropic diffusion

- Replace embedding function in level-sets formulation by image itself

$$\frac{\partial \Phi}{\partial t} = g(I)\,\kappa\,\|\nabla\Phi\| + \nabla g(\mathrm{I})\cdot\nabla\Phi$$

$$\frac{\partial I}{\partial t} = g(I)\,\kappa\,\|\nabla I\| + \nabla g(I)\cdot\nabla I$$

Anisotropic diffusion (Alvarez et al.)

Shock-filters (Osher-Rudin)

Relation with Perona-Malik anisotropic diffusion

$$\frac{\partial \Phi}{\partial t} = g(I)\,\kappa\,\|\nabla\Phi\| + \nabla g(\mathrm{I})\cdot\nabla\Phi$$

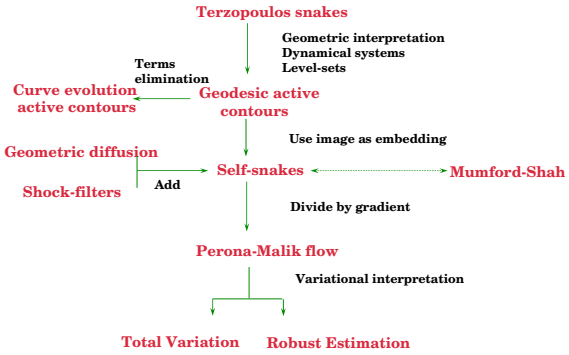
$$\Updownarrow$$

$$\frac{\partial \Phi}{\partial t} = \|\nabla\Phi\|\,\mathrm{div}\left(g(I)\,\frac{\nabla \mathrm{I}}{\|\nabla I\|}\right)$$

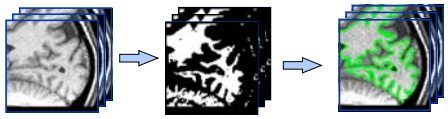
$$\iint h(\|\nabla\Phi\|) \Rightarrow \frac{\partial \Phi}{\partial t} = \mathrm{div}\left(h'(I)\,\frac{\nabla \mathrm{I}}{\|\nabla I\|}\right)$$

Total variation, Robust estimation — Anisotropic diffusion

Concluding remarks



Anisotropic Diffusion of the Posterior



ADP in MRI

Review: MAP Estimation

- 3 classes: sulcus, gray matter, white matter
- Prior probability: Pr(class=C)
- Posterior probability: Pr(class=C | data)
- *MAP*: Choose class C that maximizes posterior:

$$C^* = \arg\max_C \Pr(\text{class}=C \mid \text{data})$$

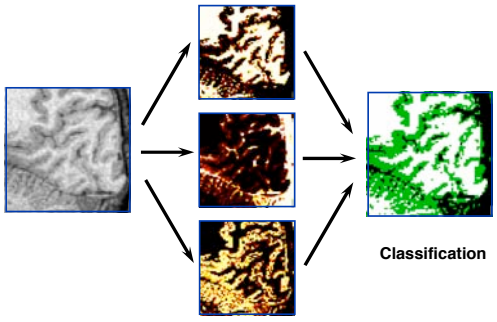
- *Bayes' Rule*:

$$\Pr(\text{class}=C \mid \text{data}) = \frac{\Pr(\text{data} \mid \text{class}=C).\Pr(\text{class}=C)}{\Pr(\text{data})}$$

- *What is our prior,* Pr(class=C)*?*

ADP: Common Techniques

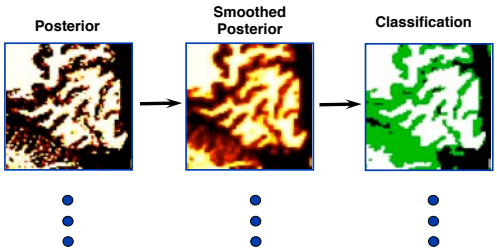
MAP Estimation: Uniform Prior



Classification

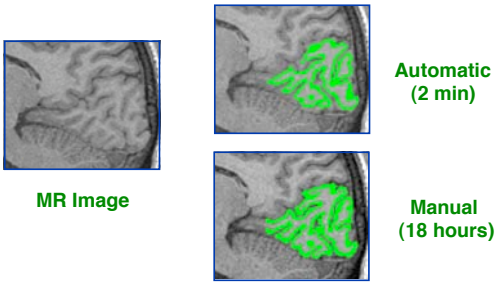
ADP: Results

Anisotropic smoothing of posterior (Teo-Sapiro-Wandell)

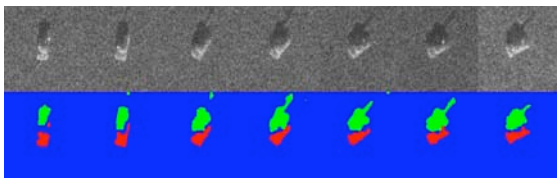


ADP: Comparisons

Comparison with manual segmentation

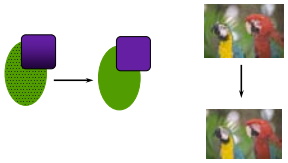


SAR segmentation via vector probability processing



With A. Pardo (see also Haker-Sapiro-Tannenbaum)

Anisotropic Diffusion in Vector Space



Goal and approach (Ringach-Sapiro)

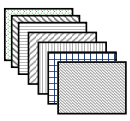
- Goal:
  - Enhancement of vector valued data
  - Extend classical theories of scalar PDE's in image processing
- Approach:
  - Work in vector space
  - Compute vector edges
  - Anisotropic diffusion
- Important: Works for any vector data
- See also: Cumani, Di Zenzo, Chambolle

Notation

- Image

$$X : [0,N]_x[0,N] \rightarrow \mathbf{R}^N$$

$L^*a^*b^*$ 

- Texture: Gabor decomposition 

Color edge computation

- Given a metric (Euclidean) $\Rightarrow \Delta X = \sqrt{\sum_i \Delta X_i^2}$
- Compute first fundamental form

$$[g_{ij}] = \frac{\partial X}{\partial i} \cdot \frac{\partial X}{\partial j}$$

- Compute eigenvectors and eigenvalues
- Edge: maximal eigenvalue and its eigenvector

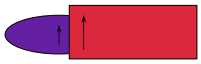
$$(\lambda_+, \lambda_-, \theta_+, \theta_-)$$

- Basic properties:
  - Eigenvectors are orthonormal
  - Minimal eigenvalue is not zero

Color anisotropic diffusion

- Direction: *Minimal change* (θ_-)
- Strength: $g(\lambda_+, \lambda_-)$

$$\frac{\partial X}{\partial t} = g(\lambda_+, \lambda_-) \frac{\partial^2 X}{\partial \theta_-^2}$$



Level lines for vectorial images (Chung-Sapiro)

Vector and scalar representation sharing level-lines

Contrast Enhancement (Sapiro-Caselles, and Caselles-Lisani-Morel-Sapiro)

- Contrast enhancement via image deformations
  - Approach: Histogram modification

$$\frac{\partial I(x,y)}{\partial t} = I(x,y) - (\#\, pixels\ of\ value \geq I(x,y))$$

$$U(I) = \frac{1}{2} \int [I(\vec{x}) - 1/2]^2 \, d\vec{x} - \frac{1}{4} \iint [I(\vec{x}) - I(\vec{z})] \, d\vec{x} \, d\vec{z}$$

  - Characteristics:
    - Simultaneous contrast enhancement and denoising
    - First explanation of histogram modification in image domain
    - Extended to local
    - First semi-global partial differential equation in image processing
    - Formal existence results

Beyond the flat manifolds

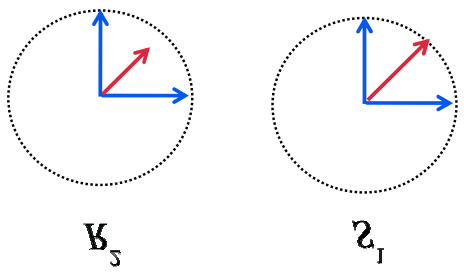
The main problem and our goal (Tang-Sapiro-Caselles)

- Goal: Enhancement and analysis of directional data (and data on non-flat manifolds)
- Problem: Directions are unit vectors:
  - Regular images vs Directions

$$I: R^2 \to R^N \quad \text{vs} \quad I: R^2 \to S^{N-1}$$

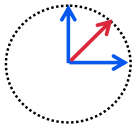
- Applications:
  - Optical flow, Gradients
  - Vector data (normalized)
  - Color image enhancement
  - Surface normals and principal directions
  - Flows in general manifolds

Average



Most popular previous approaches

- Work with angles: Operations on the sphere
  - Average, median, etc
  - *Statistics of directional data*, Mardia
  - "Orientation Diffusion," Perona (1998)



- Tensor diffusion
  - Weickert, Granlund-Knuttson
- See also Chan-Shen

Anisotropic Diffusion

$$\frac{\partial I(x,y,t)}{\partial t} = \Delta I \qquad \frac{\partial I(x,y,t)}{\partial t} = div(g(|\nabla I|)\nabla I)$$

What have we learned from images?

Robust Estimation: $\min_I \int_\Omega \rho(|\nabla I|) d\Omega$

Robust function

$\Downarrow$

Gradient Descent: $\frac{\partial I(x,y,t)}{\partial t} = div\left(\rho' \frac{\nabla I}{|\nabla I|}\right)$

$\Downarrow$

$g := \frac{\rho'}{|\nabla I|}$

Influence function
(defines outliers)

Anisotropic Diffusion: $\frac{\partial I(x,y,t)}{\partial t} = div(g(|\nabla I|)\nabla I)$

Anisotropic Diffusion

$$\frac{\partial I(x,y,t)}{\partial t} = \Delta I \qquad \frac{\partial I(x,y,t)}{\partial t} = div(g(|\nabla I|)\nabla I)$$

Back to Directions: Basic Idea

- Use the theory of harmonic maps
  - Find a map *I* from two manifolds *(M,g)* and *(N,h)* such that

$$\min_{I:M\to N} \int_\Omega \left\| \nabla_M I \right\|^p dvol_M$$

  - In particular, liquid crystals:

$$\min_{I:R^2\to S^{n-1}} \int_\Omega \left\| \nabla I \right\|^p dx\, dy$$

The Gradient-Descent Equations

General (p=2)

$$\frac{\partial I}{\partial t} = \Delta_M I + A_N(I) < \nabla_M I, \nabla_M I >$$

Liquid crystals

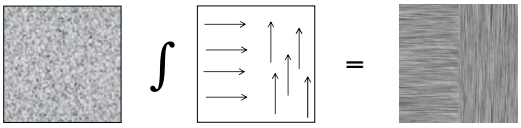
$$\frac{\partial I}{\partial t} = div(\|\nabla I\|^{p-2} \nabla I) + I\|\nabla I\|^p$$

A Few Theoretical Results (over hundreds relevant)

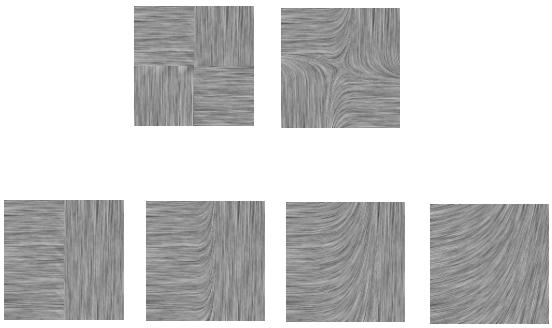
- For 2D unit vectors (n=1), and p=2, a unique solution exists and singularities are isolated points (if they exists at all). For smooth data, singularities do not occur.
- Singularities occur for 3D unit vectors (p=2).
- Singularities well characterized for 1<p<=2.
- Energy well characterized for 1<p<=2.
- No singularities for manifolds with non-positive curvature.

Intermezzo: Visualizing Directions

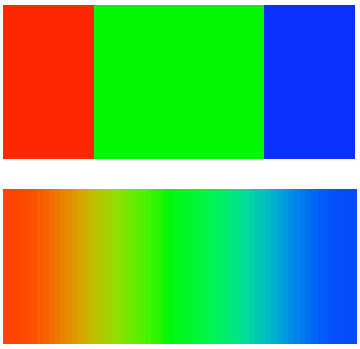
- Arrows
- Color Map
- Line Integral Convolution



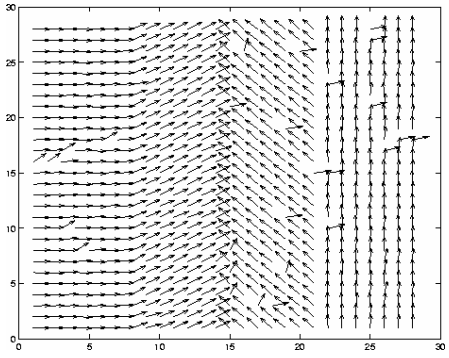
Examples (Isotropic)

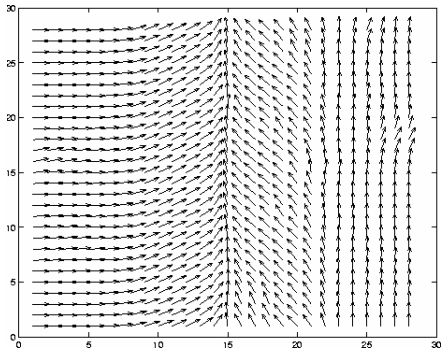


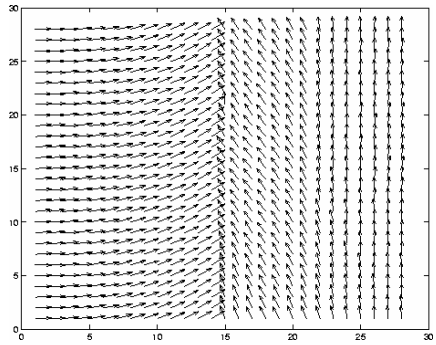
3D vector (Isotropic)



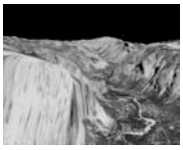
Denoising

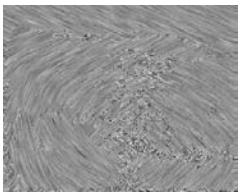


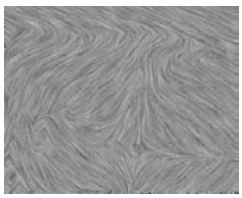




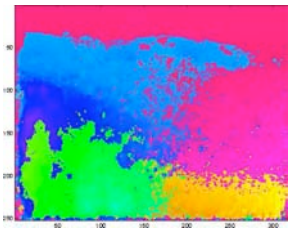
Optical flow

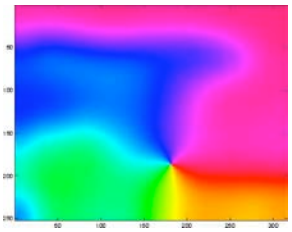




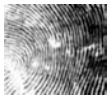


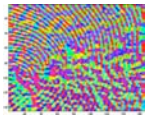
Optical flow (cont.)

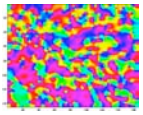


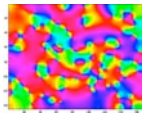


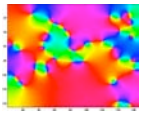
Gradient



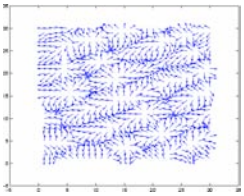


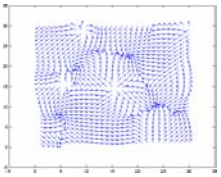


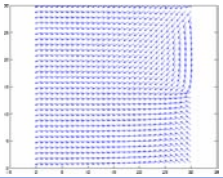




Gradient (cont.)







Color Image Enhancement

Color image enhancement (cont.)

Color image enhancement (cont.)

Vector probability diffusion (with Alvaro Pardo)

- Perform diffusion on the hyperplane representing probabilities

$$\min_{I:R^2 \to H^n} \int_\Omega \|\nabla I\|^p \, dx\, dy$$

$$\frac{\partial I}{\partial t} = \Delta_M I + A_N(I) < \nabla_M I, \nabla_M I >$$

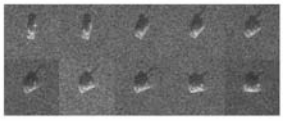
$$\frac{\partial I}{\partial t} = div(\|\nabla I\|^{p-2} \nabla I)$$

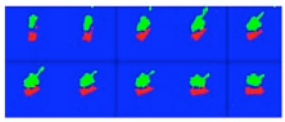
Vector probability diffusion (cont.)

- The numerical implementation also stays on the hyperplane
- The numerical implementation also holds a maximum and minimum principle

Vector probability diffusion (cont.)

- Diffuse posterior probabilities (following Teo-Sapiro-Wandell and Haker-Sapiro-Tannenbaum)





WHY THE WORLD LOVES THE LEVEL SET METHOD (AND RELATED) METHODS

Stan Osher

Mathematics Department
UCLA
sjo@math.ucla.edu

and

Level Set Systems, Inc.
1058 Embury St.
Pacific Palisades, CA 90272-2501
sjo@levelset.com

Google: "Level Set Methods"

$\approx$ 8200 responses

(September 17, 2003)

Osher-Sethian original paper (1988)

Now cited $\approx$ 720 times (web-of-science)

ICCV (2003) Nice

New Book

S. Osher & R.P. Fedkiw

Level Set Methods and Dynamic Implicit Surfaces.
Springer-Verlag

It's out!!

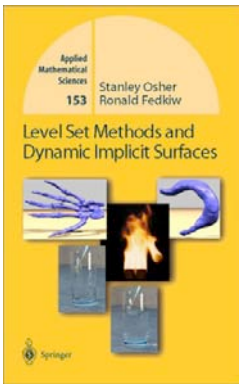
Also

S. Osher & N. Paragios (Eds)

Geometric Level Set Methods in Imaging, Vision & Graphics

Springer-Verlag (2003)

Also
Course at SIGGRAPH (2002)
San Antonio, July 2002



Given an interface in R^n, call it Γ, of codimension one,

Move it normal to itself under velocity $\vec{v}$

$\vec{v} = \vec{v}$ (x, geometry, external physics)

O & Sethian (1987)

Also: Unreferenced papers by

Dervieux, Thomassett, (1979, 1980).

Some of the key ideas in obscure proceedings.

Trivial fact

Zero level contour

$$\phi(x,t) = 0$$

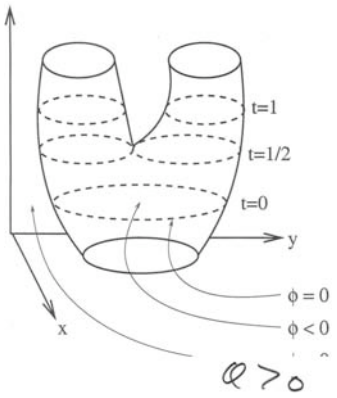
$$\frac{d}{dt}\phi(x(t),t) = 0$$

$$\varphi_t + \vec{v}\cdot\vec{\nabla}\varphi = 0$$

Normal to $\Gamma: \vec{n} = \frac{\vec{\nabla}\varphi}{|\nabla\varphi|}$

$$\varphi_t + v_n|\vec{\nabla}\varphi| = 0$$

$$v_n = \vec{v}\cdot\frac{\vec{\nabla}\varphi}{|\vec{\nabla}\varphi|}$$



Tracking:

VOF

Merging is difficult

3D is difficult

Reparametrization needed

Advect $\chi(x) \equiv 1$ if $x \in \Omega$

$\equiv 0$ outside

\+ Merging ok

-- Spurious discontinuity

-- Hard to compute curvature.

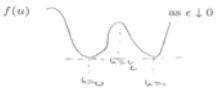
Phase field

e.g.

Mean Curvature

$$u_t = \Delta u - \frac{1}{\varepsilon} f_u$$

get curvature



interface $O(\varepsilon)$

width

But $\Delta x < \varepsilon$, otherwise

(Thm: MBO, phase field gives the wrong answer)

Need adaptive grid,

NO ε in our approach.

(1) Reinitialize

$\varphi \rightarrow$ signed distance to Γ (SSO).

(2) $v_n \rightarrow$ extends smoothly off of Γ (CMOS).

(3) Local level set (near interface) $|\varphi| < \varepsilon$.

Easy to implement

Near boundary singularities, 2 or 3D.

Also

$$v_n = v_n(\vec{n}), \qquad v_n(\vec{n}) = \gamma\left(\frac{\nabla\varphi}{|\nabla\varphi|}\right)$$

$$\varphi_t + |\nabla\varphi|\,\gamma\left(\frac{\nabla\varphi}{|\nabla\varphi|}\right) = 0$$

High order accurate ENO schemes for HJ equations

(Kinks develop)

[OSe] [OSh]

Theoretical Justification

Viscosity solutions for scalar 2nd order (or 1st order)

Evolution eqns.

Motion by mean curvature e.g.

$$\varphi_t = |\nabla\varphi| \nabla\cdot\frac{\nabla\varphi}{|\nabla\varphi|}$$

ESS showed same as classical limit

$$u_t = \Delta u - \frac{1}{\varepsilon} f_u$$

$f =$ 

as $\varepsilon \downarrow 0$

Got e.g. motion of square by mean curvature.

Level Set Dictionary

1. $\Gamma(t)\{x \mid \varphi(x,t) = 0\}$
   $\Omega(t)bdd$ by $\Gamma(t)$
   $\Omega(t) = \{x \mid \varphi(x,t) < 0\}$

2. Unit normal
   $$\vec{n} = +\frac{\vec{\nabla}\varphi}{|\vec{\nabla}\varphi|}$$

3. Mean curvature
   $$\kappa = \nabla\cdot\left(\frac{\vec{\nabla}\varphi}{|\vec{\nabla}\varphi|}\right)$$

4. Delta function on an interface
   $$\delta(\varphi)\,|\nabla\varphi|$$

5. Characteristic function χ of $\Omega(t)$
   $$\chi = H(-\varphi)$$
   $H(x) \equiv 1$ if $x > 0$
   $H(x) \equiv 0$ if $x < 0$

6. Surface integral of $p(x,t)$ over Γ
   $$\int_{R^n} p(x,t)\delta(\varphi)\,|\nabla\varphi|\,dx$$

7. Volume integral of $p(x,t)$ over Ω
   $$\int p(x,t)H(-\varphi)dx$$

8. Distance reinitialization $d(x,t)$ = signed distance to nearest point on Γ
   $|\nabla d| = 1$ $d < 0$ in Ω, $d > 0$ in Ω^c
   $d = 0$ on Γ
   $$\frac{\partial\psi}{\partial\tau} + \text{sign}\varphi(|\nabla\psi| - 1) = 0$$

   as $\tau\uparrow$ $\psi \to$ d *very* fast near $d = 0$.

9. Smooth extension of a quantity e.g. v_n on Γ, off of Γ.

Let $v_n = p(x,t)$

$$\frac{\partial q}{\partial \tau} + (\text{sign}\varphi)\left(\frac{\nabla\varphi}{|\nabla\varphi|}\cdot\nabla q\right) = 0$$
$$q(x,0) = p(x,t)$$

very fast near $d = 0$.

10. Local level set method.

Solve PDE within $6\Delta x$ or so of $d = 0$.

11. Fast marching method: Tsitsiklis (1993)

Rediscovered by (1995): Helmsen P.C.D.,…, & Sethian

$$\varphi_t + v_n(\vec{x})\,|\nabla\varphi| = 0$$
$$v_n > 0$$

Use heap-sort, Godunov's Hamiltonian (upwind, viscosity soln)

Solve in $O(N \log N)$

(First order accurate), jazzed up hyperbolic space Marching.

For this problem, probably fastest.
Although local level set more general & accurate.

For more complicated Hamiltonians

$$H(\vec{x},\nabla\varphi) = c(\vec{x}) > 0$$

H convex in grad phi

Can do a simple *local* update $\begin{smallmatrix} & 0 & \\ 0 & x & 0 \\ & 0 & \end{smallmatrix}$ using a new Formula of Tsai, et. al. (2001)

Sweep in pre-ordained directions. Converges rapidly. No heap sort. No large search and initialization regions.
Zhao: "convergence theorem" in special cases.

Now, with Kao, Jiang & O, can do a very simple sweeping method in very general cases.

Level Set Methods: An Overview and Some Recent Results *

Stanley Osher †
Ronald P. Fedkiw ‡

September 5, 2000

Abstract

The level set method was devised by Osher and Sethian in [64] as a simple and versatile method for computing and analyzing the motion of an interface Γ in two or three dimensions. Γ bounds a (possibly multiply connected) region Ω. The goal is to compute and analyze the subsequent motion of Γ under a velocity field $\vec{v}$. This velocity can depend on position, time, the geometry of the interface and the external physics. The interface is captured for later time as the zero level set of a smooth (at least Lipschitz continuous) function $\varphi(\vec{x}, t)$, i.e., $\Gamma(t) = \{\vec{x} | \varphi(\vec{x}, t) = 0\}$. φ is positive inside Ω, negative outside Ω and is zero on $\Gamma(t)$. Topological merging and breaking are well defined and easily performed.

In this review article we discuss recent variants and extensions, including the motion of curves in three dimensions, the Dynamic Surface Extension method, fast methods for steady state problems, diffusion generated motion and the variational level set approach. We also give a user's guide to the level set dictionary and technology, couple the method to a wide variety of problems involving external physics, such as compressible and incompressible (possibly reacting) flow, Stefan problems, kinetic crystal growth, epitaxial growth of thin films,

*Research supported in part by ONR N00014-97-1-0027, DARPA/NSF VIP grant NSF DMS9615854, AFOSR FQ8671-9801346, NSF DMS 9706827 and ARO DAAG 55-98-1-0323.

†Department of Mathematics, University of California Los Angeles, Los Angeles, California 90095

‡Computer Science Department, Stanford University, Stanford, California 94305.

vortex dominated flows and extensions to multiphase motion. We conclude with a discussion of applications to computer vision and image processing.

1 Introduction

The original idea behind the level set method was a simple one. Given an interface Γ in R^n of codimension one, bounding a (perhaps multiply connected) open region Ω, we wish to analyze and compute its subsequent motion under a velocity field $\vec{v}$. This velocity can depend on position, time, the geometry of the interface (e.g. its normal or its mean curvature) and the external physics. The idea, as devised in 1987 by S. Osher and J.A. Sethian [64] is merely to define a smooth (at least Lipschitz continuous) function $\varphi(x,t)$, that represents the interface as the set where $\varphi(x,t) = 0$. Here $x = x(x_1, \ldots, x_n)\ \varepsilon\ R^n$.

The level set function φ has the following properties

$$\begin{aligned} \varphi(x,t) &> 0 \text{ for } x \in \Omega \\ \varphi(x,t) &< 0 \text{ for } x \notin \bar{\Omega} \\ \varphi(x,t) &= 0 \text{ for } x \in \partial\Omega = \Gamma(t) \end{aligned}$$

Thus, the interface is to be captured for all later time, by merely locating the set $\Gamma(t)$ for which φ vanishes. This deceptively trivial statement is of great significance for numerical computation, primarily because topological changes such as breaking and merging are well defined and performed "without emotional involvement".

The motion is analyzed by convecting the φ values (levels) with the velocity field $\vec{v}$. This elementary equation is

$$\frac{\partial \varphi}{\partial t} + \vec{v} \cdot \nabla \varphi = 0. \tag{1}$$

Here $\vec{v}$ is the desired velocity on the interface, and is arbitrary elsewhere.

Actually, only the normal component of v is needed: $v_N = \vec{v} \cdot \frac{\nabla \varphi}{|\nabla \varphi|}$, so (1) becomes

$$\frac{\partial \varphi}{\partial t} + v_N |\nabla \varphi| = 0. \tag{2}$$

In section 3 we give simple and computationally fast prescriptions for reinitializing the function φ to be signed distance to Γ, at least near the boundary [84], smoothly extending the velocity field v_N off of the front Γ [24] and solving equation (2) only locally near the interface Γ, thus lowering the complexity of this calculation by an order of magnitude [66]. This makes the cost of level set methods competitive with boundary integral methods, in cases when the latter are applicable, e.g. see [42].

We emphasize that all this is easy to implement in the presence of boundary singularities, topological changes, and in 2 or 3 dimensions. Moreover, in the case which v_N is a function of the direction of the unit normal (as in kinetic crystal growth [62], and Uniform Density Island Dynamics [15], [36]) then equation (2) becomes the first order Hamilton-Jacobi equation

$$\frac{\partial \varphi}{\partial t} + |\nabla \varphi| \gamma \left(\vec{N} \right) = 0 \tag{3}$$

where $\gamma = \gamma(\vec{N})$ a given function of the normal, $\vec{N} = \frac{\nabla \varphi}{|\nabla \varphi|}$.

High order accurate, essentially non-oscillatory discretizations to general Hamilton-Jacobi equations including (3) were obtained in [64], see also [65] and [43].

Theoretical justification of this method for geometric based motion came through the theory of viscosity solutions for scalar time dependent partial differential equations [23], [30]. The notion of viscosity solution (see e.g. [8, 27]) – which applies to a very wide class of these equations, including those derived from geometric based motions – enables users to have confidence that their computer simulations give accurate, unique solutions. A particularly interesting result is in [29] where motion by mean curvature, as defined by Osher and Sethian in [64], is shown to be essentially the same motion as is obtained from the asymptotics in the phase field reaction diffusion equation. The motion in the level set method involves no superfluous stiffness as is required in phase field models. As was proven in [53], this stiffness due to a singular perturbation involving a small parameter ϵ will lead to incorrect answers as in [48], without the use of adaptive grids [59]. This is not an issue in the level set approach.

The outline of this paper is as follows: In section 2 we present recent variants, extensions and a rather interesting selection of related fast numerical methods. This section might be skipped at first, especially by newcomers to this subject. Section 3 contains the key definitions and basic level set technology, as well as a few words about the numerical implementation. Section 4 describes applications in which the moving interfaces are coupled to external physics. Section 5 concerns the variational level set approach with applications to multiphase (as opposed to two phase) problems. Section 6 gives a very brief introduction to the ever-increasing use of level set method and related methods in image analysis.

2 Recent Variants, Extensions and Related Fast Methods

2.1 Motion of Curves in Three Spatial Dimensions

In this section we discuss several new and related techniques and fast numerical methods for a class of Hamilton-Jacobi equations. These are all relatively recent developments and less experienced readers might skip this section at first.

As mentioned above, the level set method was originally developed for curves in R^2 and surfaces in R^3. Attempts have been made to modify it to handle objects of high codimension. Ambrosio and Soner [5] were interested in moving a curve in R^3 by curvature. They used the squared distance to the curve as the level set function, thus fixing the curve as the zero level set, and evolved the curve by solving a PDE for the level set function. The main problem with this approach is that one of the most significant advantages of level set method, the ability to easily handle merging and pinching, does not carry over. A phenomenon called "thickening" emerges, where the curve develops an interior.

Attempts have also been made in other directions, front tracking, e.g. see [41]. This is where the curve is parameterized and then numerically represented by discrete points. The problem with this approach lies in finding when merging and pinching will occur and in reparameterizing the curve when it does. The representation we derived in [13] makes use of two level set functions to model a curve in R^3, an approach Ambrosio and Soner also suggested but did not pursue because the theoretical aspects become very difficult. In this formulation, a curve is represented by the intersection between the zero level sets of two level set functions ϕ and ψ, i.e., where $\phi = \psi = 0$. From this, many properties of the curve can be derived such as the tangent vectors, $\vec{T} = \frac{\nabla\psi \times \nabla\phi}{|\nabla\psi \times \nabla\phi|}$, the curvature vectors, $\kappa\vec{N} = \nabla\vec{T} \cdot \vec{T}$, and even the torsion, $\tau\vec{N} = -\nabla\vec{B} \cdot \vec{T}$, where $\vec{N}$ and $\vec{B}$ are the normal and binormal respectively.

Motions of the curve can then be studied under the appropriate system of PDE's involving the two level set functions. The velocity can depend on external physics, as well as on the geometry of the curve (as in the standard level set approach). The resulting system of PDE's for ψ and ϕ is

$$\phi_t = -\vec{v} \cdot \nabla\phi$$
$$\psi_t = -\vec{v} \cdot \nabla\psi$$

A simple example involves moving the curve according to its curvature vectors, for which $\vec{v} = \kappa\vec{N}$. We have shown that this system can also be obtained by applying a gradient descent algorithm minimizing the length of the curve,

$$L(\phi, \psi) = \int_{R^3} |\nabla\psi \times \nabla\phi|\delta(\psi)\delta(\phi)d\vec{x}.$$

This follows the general procedure derived in [88] for the variational level set method for codimension one motion, also described in [90]. Numerical simulations performed in [13] on this system of PDE's, and shown in figures 1 and 2, show that merging and pinching off are handled automatically and follow curve shortening principles.

We repeat the observation made above that makes this sort of motion easily accessible to this vector valued level set method. Namely all geometric properties of a curve Γ which is expressed as the zero level set of the vector equation

$$\begin{aligned}\phi(x, y, z, t) &= 0\\ \psi(x, y, z, t) &= 0\end{aligned}$$

can easily be obtained numerically by computing discrete gradients and higher derivatives of the functions ϕ and ψ restricted to their common zero level set.

This method will be used to simulate the dynamics of defect lines as they arise in heteroepitaxy of non-lattice notched materials, see [79] and [80] for Lagrangian calculations.

An interesting variant of the level set method for geometry based motion was introduced in [53] as diffusion generated motion, and has now been generalized to forms known as convolution generated motion or threshold dynamics. This method splits the reaction diffusion approach into two highly simplified steps. Remarkably, a vector valued generalization of this approach, as in the vector valued level set method described above gives an alternative approach [74] to easily compute the motion (and merging) of curves moving normal to themselves in three dimensions with velocity equal to their curvature.

2.2 Dynamic Surface Extension (DSE)

Another fixed grid method for capturing the motion of self-intersecting interfaces was obtained in [73]. This is a fixed grid, interface capturing formulation based on the Dynamic Surface Extension (DSE) method of Steinhoff

et. al. [82]. The latter method was devised as an alternative to the level set method of Osher and Sethian [64] which is needed to evolve wavefronts according to geometric optics. The problem is that the wavefronts in this case are supposed to pass through each other – not merge as in the viscosity solution case. Ray-tracing can be used but the markers tend to diverge which leads to loss of resolution and aliasing.

The original (ingenious) DSE method was not well suited to certain fundamental self intersection problems such as formation of swallowtails. In [73] we extended the basic DSE scheme to handle this fundamental problem, as well as all other complex intersections.

The method is designed to track moving sets Γ of points of arbitrary (perhaps changing) codimension, moreover there is no concept of "inside" or "outside". The method is, in some sense, dual to the level set method. In the latter, the distance representation is constant tangential to a surface. In the DSE method, the closest point to a surface is constant in directions orthogonal to the surface.

The version of DSE presented in [73] can be described as follows:

For each point in R^n, set the tracked pointed $TP(\vec{x})$ equal to $CP(\vec{x})$ the closest point (to $\vec{x}$) on the initial surface Γ_0. Set $\vec{N}$ equal to the surface normal at the tracked point $TP(\vec{x})$. Let $\vec{v}(TP(\vec{x}))$ be the velocity of the tracked point.

Repeat for all steps:

(1) Evolve the tracked point $TP(\vec{x})$ according to the local dynamics $TP(\vec{x})_t = \vec{v}(TP(\vec{x}))$.

(2) Extend the surface representation by resetting each tracked point $TP(\vec{x})$ equal to the true closest point $CP(\vec{x})$ on the updated surface Γ, where Γ is defined to be the locus of all tracked points, i.e. $\Gamma = \{TP(\vec{x}) | \vec{x}\, \varepsilon\, R^n\}$.

Replace each $\vec{N}(\vec{x})$ by the normal at the updated $TP(\vec{x})$.

This method treats self intersection by letting moving sets pass through each other. This is one of its main virtues in the ray tracing case. However, it has other virtues – namely the generality of the moving set – curves can end or change dimension.

An important extension is motivated by considering first arrival times. This enables us to easily compute swallowtails, for example, and other singular points. We actually use a combination of distance and direction of

motion. One interesting choice arises when nodal values of $TP(\vec{x})$ are set equal to the "Minimizing Point"

$$MP(\vec{x}) = \min_{\vec{y}\ \varepsilon\ \text{Interface}} \beta|(\vec{x}-\vec{y})\cdot\vec{N}^{\perp}(\vec{y})| + \|\vec{x}-\vec{y}\|^2$$

for $\beta > 0$ (rather than $CP(\vec{x})$), since a good agreement with the minimal arrival time representation is found near the surface. Recall that the minimal arrival time at a point $\vec{x}$ is the shortest time it takes a ray emanating from the surface to reach $\vec{x}$. Using this idea gives a very uniform approximation and naturally treats the prototype swallowtail problem.

For the special case of curvature dependent motion we may use an elegant observation of DeGiorgi [28]. Namely the vector mean curvature for a surface of arbitrary codimension is given by $\kappa\vec{N} = -\Delta\nabla\left(\frac{d^2}{2}\right)$ where κ is the local mean curvature and d is the distance to the surface. Using the elementary, but basic fact that

$$d\nabla d = \vec{x} - CP(\vec{x})$$

where $CP(\vec{x})$ is the closest point to $\vec{x}$ on the surface, we obtain a very simple expression for vector mean curvature

$$\kappa\vec{N} = -\Delta(\vec{x} - CP(\vec{x})) = \Delta CP(\vec{x}).$$

Thus motion by a function F, of mean curvature for surfaces of arbitrary codimension can be achieved by using $\vec{v}(TP(\vec{x})) = \Delta CP(\vec{x})$. Then curvature dependent velocities are possible by using

$$\vec{v} = F(\Delta CP(\vec{x})|_{TP(\vec{x})}\cdot\vec{N})\vec{N}.$$

where numerical experiments in [73] have validated these algorithms to some degree.

A variety of interesting topics for future research is still open. In particular, adjustments need to be made if merging is desired. Moreover we can move objects with more complex topology and geometry, such as surfaces with boundaries (or curves with endpoints), objects of composite topology (such as a filament attached to a sheet) and surfaces on curves with triple point junctions (see [88], [53] and section 5 of this paper for successful level set based and diffusion generated based approaches for the codimension one case respectively).

Further work in the area of curvature dependent motions is also possible. Computationally the construction of fast extension methods and localization

as in [66] for the level set method would be of great practical importance. It would be particularly interesting to determine if surfaces fatten (or develop interiors) when mergers occur. See [9] for a detailed discussion of this phenomenon.

Additionally in [73] we successfully calculated a geometric optics expansion by retaining the wave front curvature. Thus this method has the possibility of being quite useful in electromagnetic calculations. We hope to investigate its three dimensional performance and include the effects of diffraction.

2.3 A Class of Fast Hamilton-Jacobi Solvers

Another important set of numerical algorithms involves the fast solution of steady (time independent) Hamilton-Jacobi equations. We also seek methods which are faster than the globally defined schemes originally used to solve equation 2. The level set method of Osher and Sethian [64] for time dependent problems can be localized. This means that the problem

$$\varphi_t + \vec{v} \cdot \nabla \varphi = 0$$

with $\Gamma(t) = \{\vec{x} | \varphi(\vec{x}, t) = 0\}$ as the evolving front, can be solved locally near $\Gamma(t)$. Several algorithms exist for doing this, see [66] and [2]. These both report an $O(N)$ algorithm where N is the total number of grid points on or near the front. However, the algorithm in [66] has $O(N \log(N))$ complexity because a partial differential equation based reinitialization step requires $\log(\frac{1}{\triangle x}) \approx \log(N)$ steps to converge (we are grateful to Bjorn Engquist for pointing this out). The algorithm in [2] claims $O(N)$ complexity, but this is not borne out by the numerical evidence presented there.

However for some special Hamilton-Jacobi equations there is a fast method whose formal complexity is $O(N \log(N))$, but which, in our experience, is around one order of magnitude faster than these general local methods.

The idea is as follows:

For an equation of the form

$$\tilde{H}(\vec{x}, \nabla \psi) = 0,$$

give $\psi = 0$ on a non characteristic set S:

$$\nabla \psi \cdot \tilde{H}_{\nabla \psi} \neq 0$$

then we proved in [63] that the t level set

$$\{\vec{x}|\psi(\vec{x}) = t\} = \Gamma'(t)$$

is the same as the zero level set $\Gamma(t)$ of $\varphi(\vec{x}, t)$, for $t > 0$ where φ satisfies

$$\tilde{H}\left(\vec{x}, -\frac{\nabla\varphi}{\varphi_t}\right) = 0.$$

This means that the viscosity solutions of either problem have level sets which correspond to each other. (This was also suggested in the original level set paper of Osher and Sethian [64]). Thus, one would like to find $\Gamma(t)$, the zero level set of $\varphi(x, t)$, as $\Gamma'(t)$, the t level set of $\psi(x)$.

A canonical example is the eikonal equation

$$\varphi_t + c(\vec{x})|\nabla\varphi| = 0, \quad c(\vec{x}) < 0$$

which can be replaced by:

$$|\nabla\psi| = -\frac{1}{c(\vec{x})} = a(\vec{x}) > 0.$$

So we find first arrival times instead of zero level sets.

In [86] J.N. Tsitsiklis devised a fast algorithm for the eikonal equation. He obtained the viscosity solution using ideas involving Dijkstra's algorithm, adapted to the eikonal equation, heap sort and control theory. From a numerical PDE point of view, however, Tsitsiklis had an apparently nonstandard approximation to $|\nabla\psi|$ on a uniform Cartesian grid.

In (1995) Sethian [76] and Helmsen et. al. [40] independently published what appeared to be a simpler algorithm making use of the Rouy-Tourin algorithm to approximate $|\nabla\varphi|$. This has become known as the "fast marching method". However, together with Helmsen [61] we have proven that Tsitsiklis' approximation is the usual Rouy-Tourin [69] version of Godunov's monotone upwind scheme. That is, the algorithm in [76] and [40] is simply Tsitsiklis' algorithm with a different (simpler) exposition.

Our goal here is to extend the applicability of this idea from the eikonal equation to any geometrically based Hamiltonian. By this we mean a Hamiltonian satisfying the properties:

$$H(\vec{x}, \nabla\psi) > 0, \quad \text{if } \nabla\psi \neq \vec{0} \tag{4}$$

and

$$H(\vec{x}, \nabla\psi) \text{ is homogeneous of degree one in } \nabla\psi \tag{5}$$

We wish to obtain a fast algorithm to approximate the viscosity solution of

$$\tilde{H}(\vec{x}, \nabla\psi) = H(\vec{x}, \nabla\psi) - a(\vec{x}) = 0. \tag{6}$$

The first step is to set up a monotone upwind scheme to approximate this problem. Such a scheme is based on the idea of Godunov used in the approximation of conservation laws. In Bardi and Osher [7], see also [65], the following was obtained (for simplicity we exemplify using two space dimensions and ignore the explicit $\vec{x}$ dependence in the Hamiltonian)

$$\begin{aligned} H(\psi_x, \psi_y) &\approx H^G(D^x_+\psi, D^x_-\psi_j; D^y_+\psi, D^y_-\psi) \\ &= \mathrm{ext}_{u\ \varepsilon\ I_{(u^-,u^+)}} \mathrm{ext}_{v\ \varepsilon\ I_{(v^-,v^+)}} H(u,v) \end{aligned}$$

where

$$I(a,b) = [\min(a,b), \max(a,b)]$$

$$\mathrm{ext}_u I(a,b) = \begin{cases} \min_{a\le u\le b} & \text{if } a \le b \\ \max_{b\le u\le a} & \text{if } a > b \end{cases}$$

$$u_\pm = D^x_\pm\psi_{ij} = \pm\frac{(\psi_{i\pm1,j} - \psi_{ij})}{\Delta x}, v_\pm = D^y_\pm\psi_{ij} = \pm\frac{(\psi_{i,j\pm1} - \psi_{ij})}{\Delta y}.$$

(Note, the order may be reversed in the ext operations above – we always obtain a monotone upwind scheme which is often, but not always, order invariant [65]).

This is a monotone upwind scheme which is obtained through the Godunov procedure involving Riemann problems, extended to general Hamilton-Jacobi equations [7], [65].

If we approximate

$$H(\nabla\varphi) = a(x,y)$$

by

$$H^G(D^x_+\varphi, D^x_-\varphi; D^x_+\psi; D^y_+\psi, D^y_-\psi) \tag{7}$$

for Hamiltonians satisfying (4), (5) above, then there exists a unique solution for $\psi_{i,j}$ in terms of $\psi_{i\pm1,j}, \psi_{i,j\pm1}$ and $\psi_{i,j}$. Furthermore $\psi_{i,j}$ is a nondecreasing function of all these variables.

However, the fast algorithm needs to have property $\mathcal{F}$: The solution to (7) depends on the neighboring $\psi_{\mu,\nu}$ only for $\psi_{\mu,\nu} < \psi_{i,j}$. This gives us a hint as to how to proceed.

For special Hamiltonians of the form: $H(u,v) = F(u^2, v^2)$, with F non-decreasing in these variables, then we have the following result [61]

$$H^G(u_+, u_-; v_+, v_-) = F(\max((u_+^-)^2, (u_-^+)^2); \max((v_+^-)^2, (v_-^+)^2)) \quad (8)$$

where $x^+ = \max(x, 0), x^- = \min(x, 0)$. It is easy to see that this numerical Hamiltonian has property $\mathcal{F}$ described above. This formula, as well as the one obtained in equation 10 below enables us to extend the fast marching method algorithm to a much wider class than was done before. For example, using this observation we were able to solve an etching problem, also considered in [3] where the authors did not use a fast marching method algorithm, but instead used a local narrow band approach and schemes devised in [64]. The Hamiltonian was

$$H(\varphi_x, \varphi_y, \varphi_z) = \sqrt{\varphi_z^2}\left(1 + \frac{4(\varphi_x^2 + \varphi_y^2)}{\varphi_x^2 + \varphi_y^2 + \varphi_z^2}\right).$$

We are able to use the same heap sort technology as for the eikonal equation, for problems of this type. See figures 3 and 4. These figures represent the level contours of an etching process whose normal velocity is a function of the direction of the normal. The process moves down in figure 3 and up in figure 4.

More generally, for $H(u,v)$ having the property

$$uH_1 \geq 0, \quad vH_2 \geq 0 \quad (9)$$

then we also proved [61]

$$H^G(u_+, u_-; v_+, v_-) = \max[H(u_+^-, v_+^-), H(u_-^+, v_+^-), H(u_+^-, v_-^+), H(u_-^+, v_-^+)] \quad (10)$$

and property $\mathcal{F}$ is again satisfied.

Again in [61], we were able to solve a somewhat interesting and very anisotropic etching problem with this new fast algorithm. Here we took

$$H(\varphi_x, \varphi_y) = |\varphi_y|(1 - a(\varphi_y)\varphi_y/(\varphi_x^2 + \varphi_y^2))$$

where

$$\begin{aligned} a &= 0 \quad \text{if} \quad \varphi_y < 0 \\ a &= .8 \quad \text{if} \quad \varphi_y > 0 \end{aligned}$$

and observed merging of two fronts. See figures 5 and 6. These figures show a two dimensional etching process resulting in a merger.

The fast method originating in [86] is a variant of Dijkstra's algorithm and, as such involves the tree like heap sort algorithm in order to compute the smallest of a set of numbers. Recently Boué and Dupuis [11] have proposed an extremely simple fast algorithm for a class of convex Hamiltonians including those which satisfy (4) and (5) above. Basically, their statement is that the standard Gauss-Seidel algorithm, with a simple ordering, converges in a *finite* number of iterations for equation (7). This would give an $O(N)$, not $O(N \log N)$ operations, with an extremely simple to program algorithm – no heap sort is needed. Moreover, for the eikonal equation with $a(x, y) = 1$, the algorithm would seem to converge in $2^d N$ iterations in R^d, $d = 1, 2, 3$, which is quite fast. This gives a very simple and fast redistancing algorithm. For more complicated problems we have found more iterations to be necessary, but still obtained promising results, together with some theoretical justification. See [85] for details, which also include results for a number of nonconvex Hamiltonians. We call this technique the "fast sweeping method" in [85]. We refer to it in section 3 when we discuss the basic distance reinitialization algorithm.

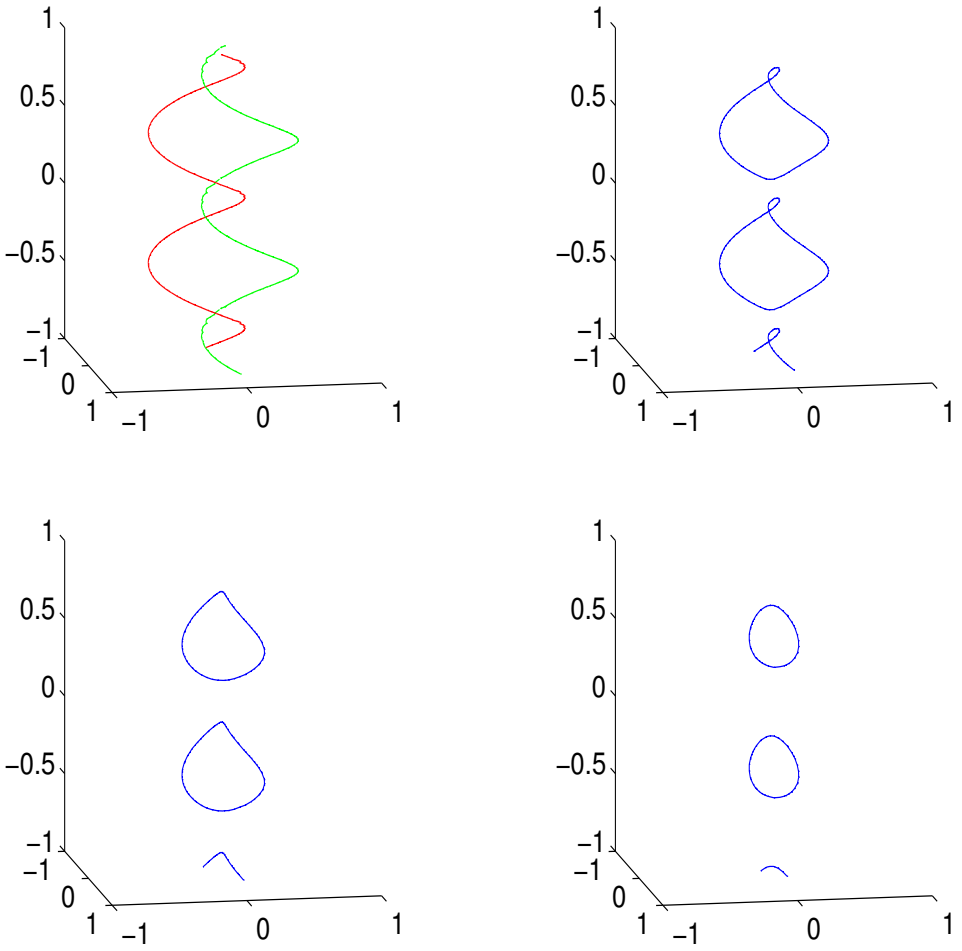


Figure 1: Merging and pinching of curves in R^3 moving by mean curvature. Reprinted from [13].

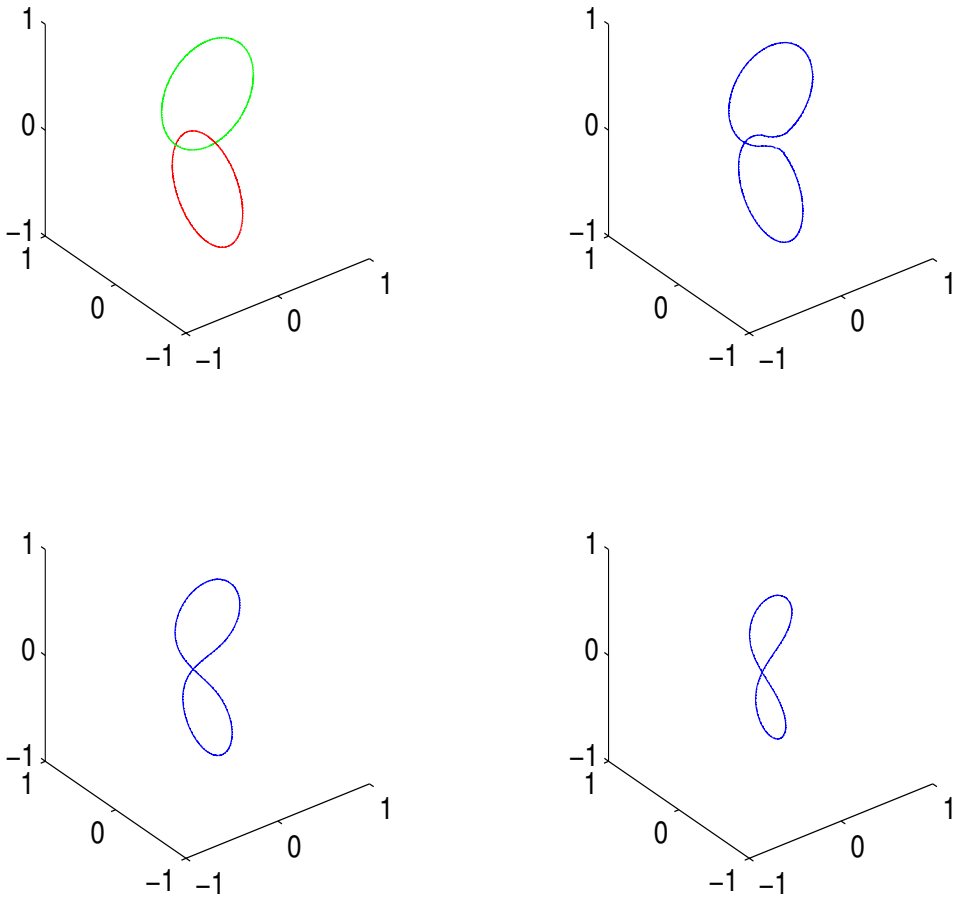


Figure 2: Merging and pinching of curves in R^3 moving by mean curvature. Reprinted from [13].

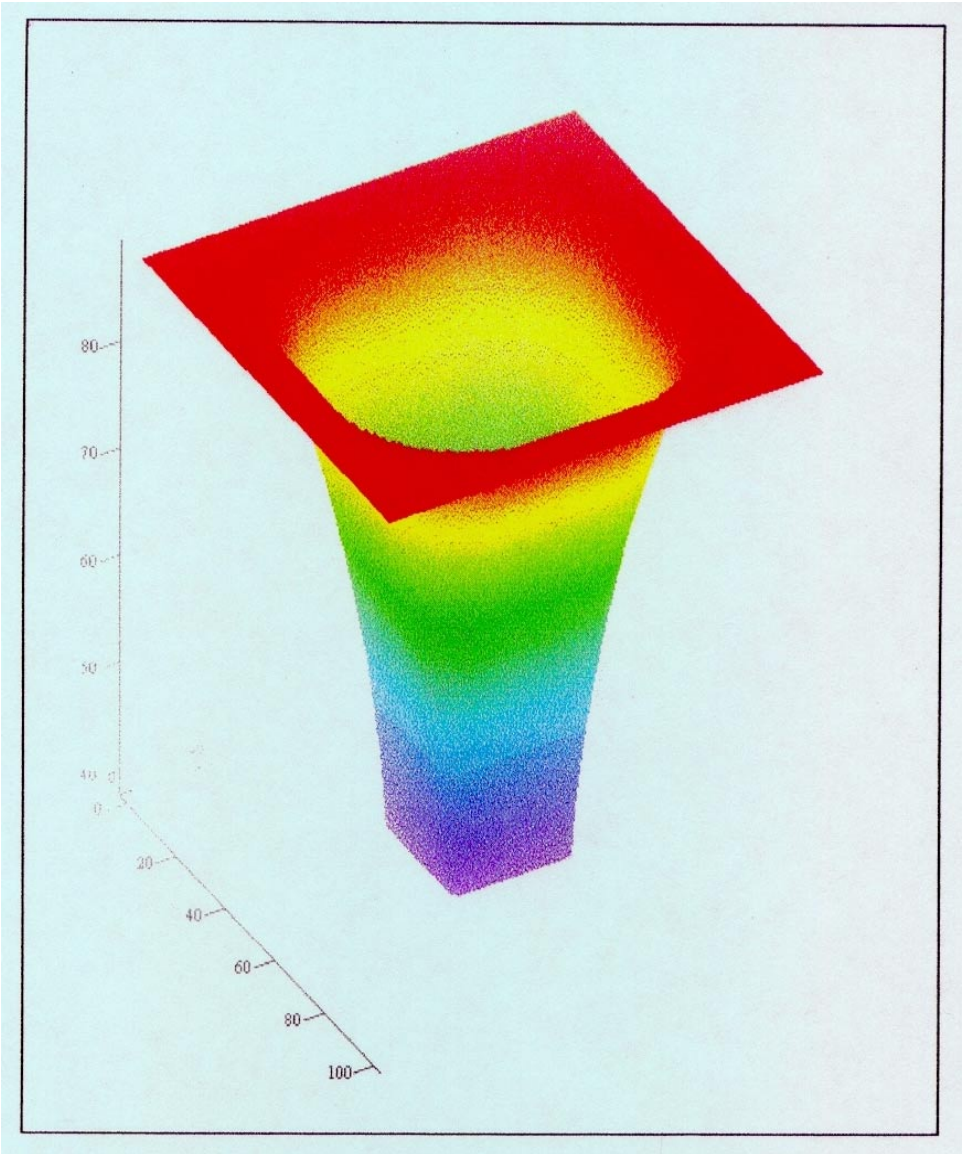


Figure 3: Three dimensional etching using a fast algorithm. Reprinted from [61].

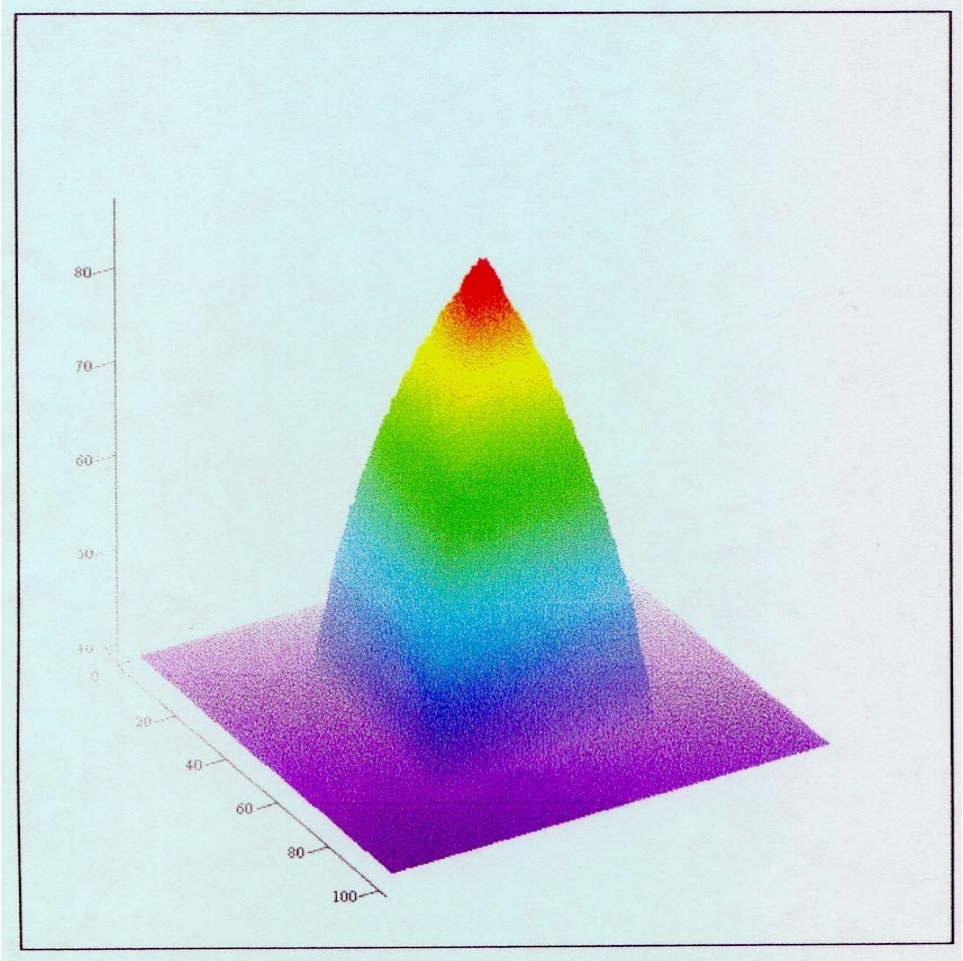


Figure 4: Three dimensional etching using a fast algorithm. Reprinted from [61].

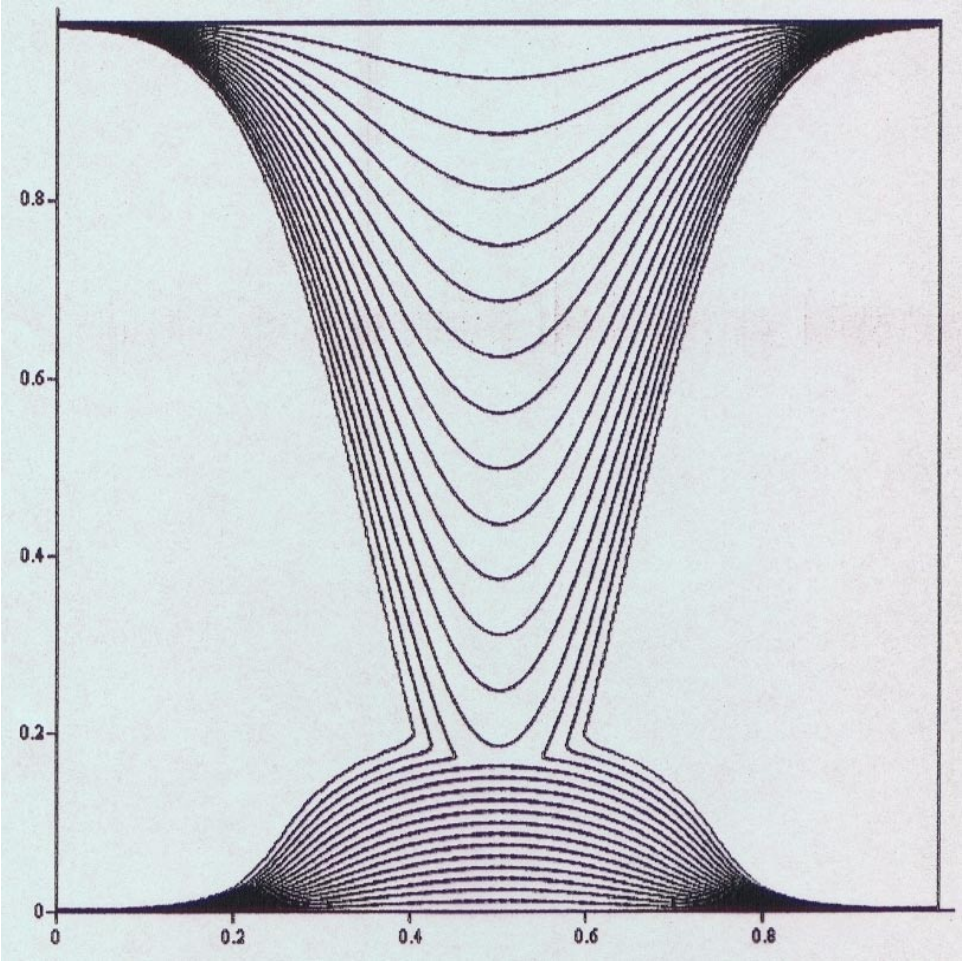


Figure 5: Two dimensional etching with merging using a fast algorithm. Reprinted from [61].

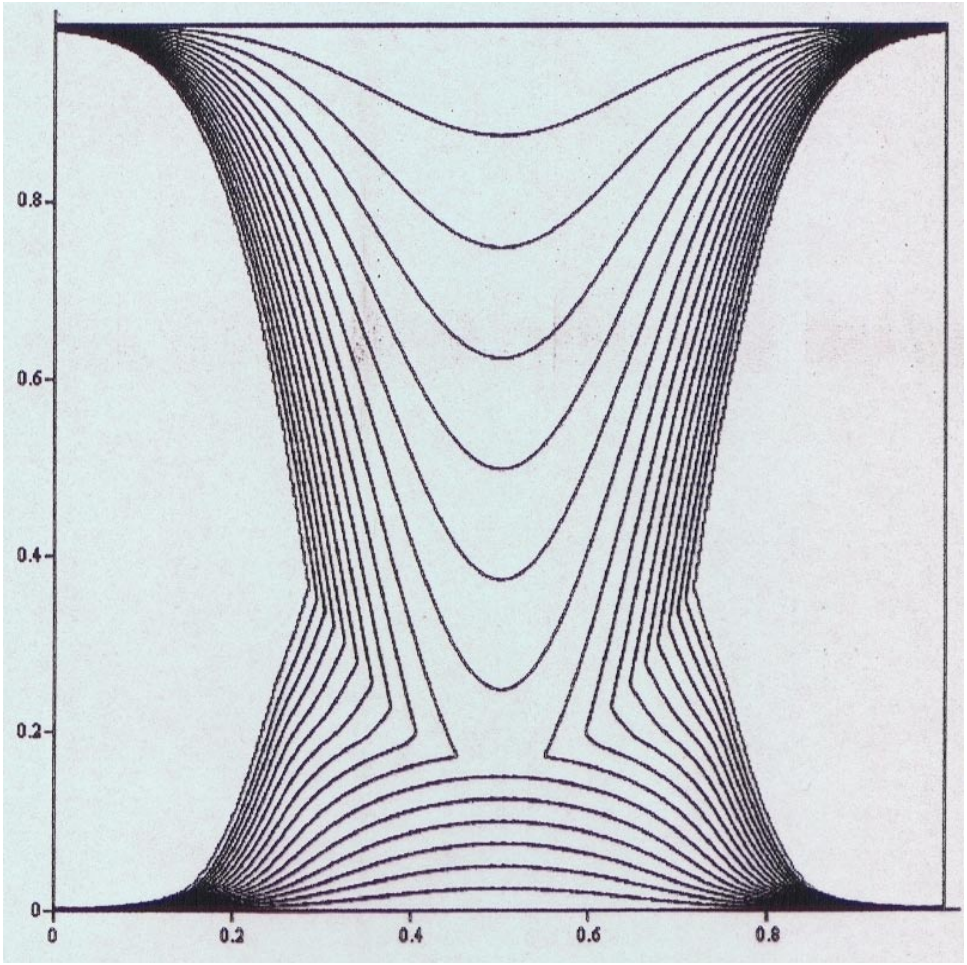


Figure 6: Two dimensional etching with merging using a fast algorithm. Reprinted from [61].

3 Level Set Dictionary, Technology and Numerical Implementation

We list key terms and define them by their level set representation.

1. The interface boundary $\Gamma(t)$ is defined by: $\{\vec{x}|\varphi(\vec{x},t)=0\}$. The region $\Omega(t)$ is bounded by $\Gamma(t)$: $\{\vec{x}|\varphi(\vec{x},t)>0\}$ and its exterior is defined by: $\{\vec{x}|\varphi(\vec{x},t)<0\}$

2. The unit normal $\vec{N}$ to $\Gamma(t)$ is given by

$$\vec{N}=-\frac{\nabla\varphi}{|\nabla\varphi|}.$$

3. The mean curvature κ of $\Gamma(t)$ is defined by

$$\kappa=-\nabla\cdot\left(\frac{\nabla\varphi}{|\nabla\varphi|}\right).$$

4. The Dirac delta function concentrated on an interface is:

$$\delta(\varphi)|\nabla\varphi|,$$

where $\delta(x)$ is a one dimensional delta function.

5. The characteristic function χ of a region $\Omega(t)$:

$$\chi=H(\varphi)$$

where

$$\begin{aligned} H(x) &\equiv 1 \text{ if } x>0 \\ H(x) &\equiv 0 \text{ if } x<0. \end{aligned}$$

is a one dimensional Heaviside function.

6. The surface (or line) integral of a quantity $p(\vec{x},t)$ over Γ:

$$\int_{R^n} p(\vec{x},t)\delta(\varphi)|\nabla\varphi|d\vec{x}.$$

7. The volume (or area) integral of $p(\vec{x},t)$ over Ω

$$\int_{R^n} p(\vec{x},t)H(\varphi)d\vec{x}.$$

Next we describe three key technological advances which are important in many, if not most, level set calculations.

8. The distance reinitialization procedure replaces a general level set function $\varphi(\vec{x},t)$ by $d(\vec{x},t)$ which is the value of the distance from $\vec{x}$ to $\Gamma(t)$, positive outside and negative inside. This assures us that φ does not become too flat or too steep near $\Gamma(t)$. Let $d(\vec{x},t)$, be signed distance of $\vec{x}$ to the closest point on Γ. The quantity $d(\vec{x},t)$ satisfies $|\nabla d| = 1,\ d > 0$ in $\Omega, d < 0$ in $(\bar{\Omega})^c$ and is the steady state solution (as $\tau \to \infty$) to

$$\begin{aligned} \frac{\partial \psi}{\partial \tau} + \mathrm{sgn}(\varphi)(|\nabla \psi| - 1) &= 0 \\ \psi(\vec{x},0) &= \varphi(\vec{x},t). \end{aligned} \tag{11}$$

where $\mathrm{sgn}(x) = 2H(x)-1$ is the one dimensional signum function. This was designed in [84]. The key observation is that in order to define d in a band of width ϵ around Γ, we need solve (11) only for $\tau = O(\epsilon)$. It can easily be shown that this can be used globally to construct distance (with arbitrary accuracy) in $O(N \log N)$ iterations [66]. Alternatively, we may use Tsitsiklis' fast algorithm [86], which is also $O(N \log N)$, with a much smaller constant, but which is only first order accurate. A locally second order accurate (in the high resolution sense) fast marching method was proposed in [77]. While this method has a much lower local truncation error than a purely first order accurate method, it is still globally first order accurate except for special cases. Finally, we might also use the fast sweeping method from [11] and [85] as described in the last section, which appears to have $O(N)$ complexity and which is also only first order accurate, although this complexity estimate has not been rigorously justified.

9. Smooth extension of a quantity, e.g. v_n on Γ to a neighborhood of Γ. Let the quantity be $p(\vec{x},t)$. Solve to steady state ($\tau \to \infty$)

$$\begin{aligned} \frac{\partial q}{\partial \tau} &+ \mathrm{sgn}(\varphi)\left(\frac{\nabla \varphi}{|\nabla \varphi|} \cdot \nabla q\right) = 0 \\ q(\vec{x},0) &= p(\vec{x},t). \end{aligned}$$

Again, we need only solve this for $\tau = O(\epsilon)$ in order to extend p to be constant in the direction normal to the interface in a tube of width ϵ.

This was first suggested and implemented in [24], analyzed carefully in [88], and further discussed and implemented in both [32] and [66]. A computationally efficient algorithm based on heap sort technology and fast marching methods was devised in [1]. There are many reasons to extend a quantity off of Γ, one of which is to obtain a well conditioned normal velocity for level contours of φ close to $\varphi = 0$ [24]. Others involve implementation of the Ghost Fluid Method of [32] discussed in the next section.

10. The basic level set method concerns a function $\varphi(\vec{x}, t)$ which is defined throughout space. Clearly this is wasteful if one only cares about information near the zero level set. The local level set method defines φ only near the zero level set. We may solve (2) in a neighborhood of Γ of width $m\Delta x$, where m is typically 5 or 6. Points outside of this neighborhood need not be updated by this motion. This algorithm works in "φ" space – so not too much intricate computer science is used. For details see [66]. Thus this local method works easily in the presence of topological changes and for multiphase flow. An earlier local level set approach called "narrow banding" was devised in [2].

Finally, we repeat that, in the important special case where v_N in equation 2 is a function only of $\vec{x}$, t and $\nabla\varphi$ (e.g. $v_N = 1$), then equation 2 becomes a Hamilton-Jacobi equation whose solutions generally develop kinks (jumps in derivatives). We seek the unique viscosity solution. Many good references exist for this important subject, see e.g. [8, 27]. The appearance of these singularities in the solution means that special, but not terribly complicated, numerical methods have to be used, usually on uniform Cartesian grids. This was first discussed in [64] and numerical schemes developed there were generalized in [65] and [43]. The key ideas involve monotonicity, upwind differencing, essentially nonoscillatory (ENO) schemes and weighted essentially nonoscillatory (WENO) schemes. See [64], [65] amd [43] for more details.

4 Coupling of the Level Set Method with External Physics

Interface problems involving external physics arise in various areas of science. The computation of such problems has a very long history. Methods of choice include front tracking, see e.g. [87] and [41], phase-field methods, see e.g. [48] and [59], and the volume of fluid (VOF) approach, see e.g. [60] and [12]. The level set method has had major successes in this area. Much of the level set technology discussed in the previous two sections was developed with such applications in mind.

Here, we shall describe level set approaches to problems in compressible flow, incompressible flow, flows having singular vorticity, Stefan problems, kinetic crystal growth and a relatively new island dynamics model for epitaxial growth of thin films. We shall also discuss a recently developed technique, the ghost fluid method (GFM), which can be used (1) to remove numerical smearing and nonphysical oscillations in flow variables near the interface and (2) to simplify the numerical linear algebra arising in some of the problems in this section and elsewhere.

4.1 Compressible Flow

Chronologically, the first attempt to use the level set method in this area came in two phase inviscid compressible flow, [55]. There, to the equations of conservation of mass, momentum and energy, we appended equation (1), which we rewrote in conservation form as

$$(\rho\varphi)_t + \nabla \cdot (\rho\varphi\vec{v}) = 0 \tag{12}$$

using the density of the fluid ρ.

The sign of φ is used to identify which gas occupied which region, so it determines the local equation of state. This (naive) method suffered from spurious pressure oscillations at the interface, as shown in [46] and [45]. These papers proposed a new method which reduced these errors by using a nonconservative formulation near the interface. However, [46] and [45] still smear out the density across the interface, leading to terminal oscillations for many equations of state.

A major breakthrough in this area came in the development of the ghost fluid method (GFM) in [32]. This enables us to couple the level set representation of discontinuities to finite difference calculations of compressible

flows. The approach was based on using the jump relations for discontinuities which are tracked using equation (1) (for two phase compressible flow). What the method amounts to (in any number of space dimensions) is to populate cells next to the interface with "ghost values", which, for two phase compressible flow retain their usual values of pressure and normal velocity (quantities which are continuous across the interface), with extrapolated values of entropy and tangential velocity (which jump across the interface). These quantities are used in the numerical flux when "crossing" an interface.

An important aspect of the method is its simplicity. There is no need to solve a Riemann problem normal to the interface, consider the Rankine-Hugoniot jump conditions, or solve an initial-boundary value problem. Another important aspect is its generality. The philosophy appears to be: at a phase boundary, use a finite difference scheme which takes only values which are continuous across the interface, using the natural values whenever possible. Of course, this implies that the tangential velocity is treated in the same fashion as the normal velocity and the pressure when viscosity is present. The same holds true for the temperature in the presence of thermal conductivity.

Figure 7 shows results obtained for two phase compressible flow using the GFM together with the level set method. Air with density around $1\frac{kg}{m^3}$ is to the left of the interface and water with density around $1000\frac{kg}{m^3}$ is to the right of the interface. Note that there is no numerical smearing of the density at the interface itself which is fortunate as water cavitates at a density above $999\frac{kg}{m^3}$ leading to host of nonphysical problems near the interface. Note too, that the pressure and velocity are continuous across the interface, although there are kinks in both of these quantities. A more complicated multidimensional calculation is shown in figure 8 where a shock wave in air impinges upon a helium droplet. See [32] for more details.

While the GFM was originally designed for multiphase compressible flow, it can be generalized to treat a large number of flow discontinuities. In [33], we generalized this method to treat shocks, detonations and deflagrations in a fashion that removes the numerical smearing of the discontinuity. Figure 9 shows the computed solution for a detonation wave. Note that there is no numerical smearing of the leading wave front which is extremely important when trying to eliminate spurious wave speeds for stiff source terms on coarse grids as first pointed out by [26]. While shocks and detonations have associated Riemann problems, the Riemann problem for a compressible flow deflagration discontinuity is not well posed unless the speed of the deflagration is given. Luckily, there is a large amount of literature on the

G-equation for flame discontinuities which was originally proposed in [50]. The G-equation represents the flame front as a discontinuity in the same fashion as the level set method so that one can easily consult the abundant literature on the G-equation to obtain deflagration speeds for the Ghost Fluid Method. Figure 10 shows two initially circular deflagration fronts that have just recently merged together. Note that the light colored region surrounding the deflagration fronts is a precursor shock wave that causes the initially circular deflagration waves to deform as they attempt to merge.

The GFM was extended in [34] in order to treat the two phase compressible viscous Navier Stokes equations in a manner that allows for a large jump in viscosity across the interface. This paper spawned the technology needed to extend the GFM to multiphase incompressible flow including the effects of viscosity, surface tension and gravity as discussed in the next subsection.

4.2 Incompressible Flow

The earliest real success in the coupling of the level set method to problems involving external physics came in computing two-phase Navier-Stokes incompressible flow [84], [22]. The equations can be written as:

$$\begin{aligned} \vec{u}_t + \vec{u} \cdot \nabla \vec{u} + \frac{\nabla p}{\rho} &= \vec{g} + \frac{\nabla \cdot (2\mu D)}{\rho} + \frac{\delta(\varphi)\sigma\kappa\vec{N}}{\rho} \\ \nabla \cdot \vec{u} &= 0 \end{aligned}$$

where $\vec{u} = (u, v, w)$ is the fluid velocity, p is the pressure, $\rho = \rho(\varphi)$ and $\mu = \mu(\varphi)$ are the piecewise constant fluid densities and viscosities, g is the gravitational force, D is the viscous stress tensor, σ is the surface tension coefficient, κ is the curvature of the interface, $\vec{N}$ is the unit normal and $\delta(\varphi)$ is a delta function. See [87] and [12] for earlier front tracking and VOF methods (respectively) using a similar formulation. This equation is coupled to the front motion through the level set evolution equation (1) with $\vec{v} = \vec{u}$. This involves defining the interface numerically as having a finite width of approximately 3 to 5 grid cells. Within this smeared out band, the density, viscosity and pressure are modeled as continuous functions. Then the $\frac{\sigma\kappa\vec{N}}{\rho}$ term is used to approximate the surface tension forces which are lost when using a continuous pressure [84]. Successful computations using this model were performed in [84] and elsewhere [22]. Problems involving area loss were observed and significant improvements were made in [83].

As mentioned above, the technology from [34] motivated the extension of the Ghost Fluid Method to this two phase incompressible flow problem.

First, a new boundary condition capturing approach was devised and applied to the variable coefficient Poisson equation to solve problems of the form

$$\nabla \left(\frac{1}{\rho} \nabla p \right) = f$$

where the jump conditions $[p] = g$ and $[\frac{1}{\rho} \nabla p \cdot \vec{N}] = h$ are given and ρ is discontinuous across the interface. This was accomplished in [49]. A sample calculation from [49] is shown in figure 11 where one can see that both the solution, p, and its first derivatives are sharp across the interface without numerical smearing. Next, this new technique was applied to multiphase incompressible flow in [44]. Here, since one can model the jumps in pressure directly, there is no need to add the $\frac{\sigma \kappa \vec{N}}{\rho}$ source term to the right hand side of the momentum equation in order to capture the surface tension forces. Instead surface tension is modeled directly by imposing a pressure jump across the interface. In addition, [44] allows for exact jumps in both ρ and μ so that the nonphysical finite width smeared out interface in [84] can be replaced by a sharp interface. A three dimensional calculation of an (invisible) solid sphere impacting water causing a splash is shown in figure 12. Here the air has density near $1 \frac{kg}{m^3}$ while the water has density near $1000 \frac{kg}{m^3}$.

Recently, in [57], this boundary condition capturing technology was extended to treat two phase incompressible flames where the normal velocity is discontinuous across the interface as well. Figure 13 shows an example calculation where two flames have just merged. Note that the velocity vectors in figure 13 clearly indicate that the velocity is kept discontinuous across the flame front. [39] considered two phase incompressible flames as well, proposing a method that keeps the interface sharp and removes numerical smearing. Unfortunately, the method proposed in [39] cannot treat topological changes in the flame front. Our method improves upon [39] allowing flame front discontinuities to merge, as in figure 13, or pinch off. Figure 14 shows two flame fronts shortly after merging in three spatial dimensions.

4.3 Topological Regularization

In [37] and [38], it is shown that the level set formulation provides a new and novel way to regularize certain ill-posed equations of interface motion, by blocking interface self-intersection. We computed two and three dimensional unstable vortex motion without regularization other than that in the

discrete approximation to $\delta(\varphi)$ – this is done over a few grid points. The key observation is that viewing a curve or surface as the level set of a function, and then evolving it with a perhaps unstable velocity field, prevents certain types of blow up from occuring. This is denoted "topological regularization". For example a tracked curve can develop a figure eight pattern, but a level set captured curve will pinch off and stabilize before this happens. For the set up (involving two functions), see [37], where we perform calculations involving the Cauchy-Riemann equations. The motions agree until pinch off, when the topological stabilization develops.

As an example, we considered the two dimensional incompressible Euler equations, which may be written as

$$\begin{aligned} \omega_t + \vec{u} \cdot \nabla \omega &= 0 \\ \nabla \times \vec{u} &= \omega \\ \nabla \cdot \vec{u} &= 0 \end{aligned}$$

We are interested in situations in which the vorticity is initially concentrated on a set characterized by the level set function φ as follows

$$\text{Vortex patch: } \omega = H(\varphi)$$

$$\text{Vortex sheet: } \omega = \delta(\varphi), \ \text{(strength of sheet is } \frac{1}{|\nabla \varphi|}\text{)}$$

$$\text{Vortex sheet dipole: } \omega = \frac{d}{d\varphi}\delta(\varphi) = \delta'(\varphi).$$

The key observation is that φ also satisfies a simple advection equation and $\vec{u}$ and ω can be easily recovered. For example, for the vortex sheet case we solve

$$\begin{aligned} \varphi_t + \vec{u} \cdot \nabla \varphi &= 0 \\ \vec{u} &= \begin{pmatrix} -\partial y \\ \partial x \end{pmatrix} \Delta^{-1} \delta(\varphi). \end{aligned}$$

Standard Laplace solvers may be used. See [38] for results involving two and three dimensional flows. In [66] we added reinitialization and extension to this procedure and obtained improved results in the two dimensional case.

4.4 Stefan Problem

Another classical field concerns Stefan problems [24], see also [78] for an earlier, but much more involved level set based approach. Here we wish to simulate melting ice or freezing water, or more complicated crystalline growth, as in the island dynamics model discussed below.

We begin with a simplified, nondimensionalized model (see [47] for an extension as mentioned below),

$$\begin{aligned} \frac{\partial T}{\partial t} &= \nabla^2 T, \quad \vec{x} \not\in \partial\Omega = \Gamma(t) \\ v_N &= [\nabla T \cdot N], \quad \vec{x}\ \varepsilon\ \Gamma(t) \end{aligned}$$

where $[\cdot]$ denotes the jump across the boundary, and

$$T = -\bar{\varepsilon}_c \kappa (1 - A\cos(k_A\theta + \theta_0)) + \bar{\varepsilon}_v v_n (1 - A\cos(k_A\theta + \theta_0))$$

on $\Gamma(t)$, and where κ is the curvature, $\theta = \cos^{-1}\frac{\varphi_x}{|\nabla\varphi|}$, and the constants $A, k_A, \theta_0, \bar{\varepsilon}_c$, and $\bar{\varepsilon}_v$ depend on the material being modeled.

We directly discretize the boundary conditions at Γ: To update T at grid nodes near the boundary, if the stencil for the heat equation would cross Γ (as indicated by nodal sign change in φ), we merely use dimension by dimension one sided interpolation and the given boundary T value at an imaginary node placed at $\varphi = 0$ (found by interpolation on φ) to compute T_{xx} and or T_{yy}, (never interpolating across the interface) rather than the usual three point central stencils. The level set function φ is updated and then reinitialized to be equal to the signed distance to Γ. Note that the level set update uses v_N that has been extended off the interface. See [24] for details.

We note that one can easily extend this to

$$\frac{\partial T}{\partial t} = \nabla \cdot (k \nabla T)$$

where k is a different positive constant inside and outside of Ω and

$$v_N = \left[k\nabla T \cdot \vec{N}\right], \ \vec{x}\ \varepsilon\ \Gamma(t).$$

as was recently done in [47].

An important observation is that our finite differencing at the interface leads to a nonsymmetric matrix inversion when applying implicit discretization in time, although the method does have nice properties such as second

order accuracy and a maximum principle. This lack of symmetry is a bit problematic for a fast implementation, especially for very large values of k. Fortunately, an extension of GFM can be used to derive a different spatial discretization producing a symmetric matrix that can be inverted rather easily using fast methods. This was originally proposed by Fedkiw [31] and is described below.

It is sufficient to explain how the spatial derivatives are derived with respect to one variable, since there are no mixed partial derivative terms. Suppose the interface point, x_f, falls in between two grid points x_i and x_{i+1}. From ϕ, the distances between x_i, x_{i+1} and x_f can be estimated by

$$x_f - x_i \approx \frac{-\phi_i \ \Delta x}{(\phi_{i+1} - \phi_i)} = \theta_1 \Delta x \tag{13}$$

$$x_{i+1} - x_f \approx \frac{\phi_{i+1} \ \Delta x}{(\phi_{i+1} - \phi_i)} = \theta_2 \Delta x \tag{14}$$

To avoid numerical errors caused by division by 0, θ_1 or θ_2 are not used if either is less than Δx^2. If $\theta_1 < \Delta x^2$, then x_f is assumed equal to x_i. If $\theta_2 < \Delta x^2$, then x_f is assumed equal to x_{i+1}. Either assumption is effectively a second order perturbation of the interface location leading to second order accurate spatial discretization. The nonsymmetric second order accurate discretization for T_{xx} given in [24] is

$$(T_{xx})_i \approx \frac{\left(\frac{T_f - T_i}{\theta_1 \Delta x}\right) - \left(\frac{T_i - T_{i-1}}{\Delta x}\right)}{\frac{1}{2}(\theta_1 \Delta x + \Delta x)} \tag{15}$$

$$(T_{xx})_{i+1} \approx \frac{\left(\frac{T_{i+2} - T_{i+1}}{\Delta x}\right) - \left(\frac{T_{i+1} - T_f}{\theta_2 \Delta x}\right)}{\frac{1}{2}(\Delta x + \theta_2 \Delta x)} \tag{16}$$

where T_f denotes the value of T at x_f and is determined from the boundary condition. Instead of using the nonsymmetric equations (15) and (16), Fedkiw [31] proposed using

$$(T_{xx})_i \approx \frac{\left(\frac{T_f - T_i}{\theta_1 \Delta x}\right) - \left(\frac{T_i - T_{i-1}}{\Delta x}\right)}{\Delta x} \tag{17}$$

$$(T_{xx})_{i+1} \approx \frac{\left(\frac{T_{i+2} - T_{i+1}}{\Delta x}\right) - \left(\frac{T_{i+1} - T_f}{\theta_2 \Delta x}\right)}{\Delta x} \tag{18}$$

which leads to a symmetric linear system when using implicit time discretization. Equation (17) is derived using linear extrapolation of T from one side

of the interface to the other, obtaining

$$T_G = T_f + (1 - \theta_1) \left(\frac{T_f - T_i}{\theta_1} \right) \tag{19}$$

as a ghost cell value for T at x_{i+1}. The standard second order discretization of $\frac{\partial^2 T}{\partial x^2}$ at x_i using T_G at x_{i+1} is

$$(T_{xx})_i \approx \frac{\left(\frac{T_G - T_i}{\Delta x} \right) - \left(\frac{T_i - T_{i-1}}{\Delta x} \right)}{\Delta x} \tag{20}$$

and the substitution of equation (19) into equation (20) leads directly to (17). Equation (18) is derived similarly.

Formulas (17) and (18) have $O(1)$ errors using formal truncation error analysis. However, they are second order accurate on a problem where the interface has been perturbed by $O(\Delta x^2)$, making them second order accurate in the interface location. Assume that the standard second order accurate discretization is used to obtain the standard linear system of equations for T at every grid point except for those adjacent to the interface, that is except for x_i and x_{i+1}. Since the linear system of equations for the nodes to the left and including x_i is independent of the system for the nodes to the right including x_{i+1}, only the linear system to the left is discussed here. Equation (20) is used to write a linear equation for T_i introducing a new unknown T_G, and the system is closed with equation (19) for T_G. In practice, equations (19) and (20) are combined to obtain equation (17) and a symmetric linear system of equations. This linear system of equations results in well determined values (up to some prescribed tolerance near roundoff error levels) of T at each grid node as well as a well determined value of T_G (from equation (19)). For the sake of reference, designate $\vec{T}$ as the solution vector containing the values of T at each grid point to the left and including x_i as well as the value of T_G at x_{i+1} which are obtained by solving this symmetric linear system. Below, $\vec{T}$ is shown to be a second order accurate solution to our problem by showing that it is the second order accurate solution to a modified problem where the interface location has been perturbed by $O(\Delta x^2)$.

Consider the modified problem where a Dirichlet boundary condition of $T = T_G$ is specified at x_{i+1} where T_G is chosen to be the value of T_G from $\vec{T}$ defined above. This modified problem can be exactly discretized to second order accuracy everywhere using the standard discretization at every node except x_i where equation (20) is used. We note that equation (20) *is* the

standard second order accurate discretization when a Dirichlet boundary condition of $T = T_G$ is applied at x_{i+1}. This new linear system can be discretized and solved in a standard fashion to obtain a second order accurate solution at each grid node. Then the realization that $\vec{T}$ is an exact solution to *this* linear system implies that $\vec{T}$ is a second order accurate solution to this modified problem. Next consider the interface location dictated by the modified problem. Since $\vec{T}$ is a second order accurate solution to the modified problem, $\vec{T}$ can be used to obtain the interface location to second order accuracy. The linear interpolant that uses T_i at x_i and T_G at x_{i+1} predicts an interface location of *exactly* x_f which is the true interface location. Since higher order interpolants (higher than linear) can contribute at most an $O(\Delta x^2)$ perturbation of the interface location, the interface location dictated by the modified problem is at most an $O(\Delta x^2)$ perturbation of the true interface location, x_f.

In [25], we used this strategy to obtain a second order accurate symmetric discretization of the variable coefficient Poisson equation

$$\nabla \left(k \nabla T \right) = f$$

on irregular domains in as many as three spatial dimensions. Then, in a straightforward way, we obtained second order accurate symmetric discretizations of the heat equation on irregular domains using backward Euler time stepping with $\triangle t = (\triangle x)^2$ and Crank-Nicolson time stepping with $\triangle t = \triangle x$.

4.5 Kinetic Crystal Growth

For an initial state consisting of any number of growing crystals in R^d, d arbitrary, moving outward with given normal growth velocity $\vec{v}(\vec{N}) > 0$ which depends on the angle of the unit surface normal $\vec{N}$, the asymptotic growth shape is a single (kinetic) Wulff-construct crystal. This result was first conjectured by Gross in (1918) [35]. This shape is also known to minimize the surface integral of $\vec{v}(\vec{N})$ for a given volume. We gave a proof of this result [62], see also [81], using the level set formulation and the Hopf-Bellman formulas [6] for the solution of a Hamilton-Jacobi equation. Additionally, with the help of the Brunn-Minkowski inequality, we showed that if we evolve a convex surface under the motion described in (3), then the ratio to be minimized monotonically decreases to its minimum as time

increases, which provides a new proof that the Wulff construction solves the generalized isoperimetric problem as well. Thus there is a new link between this hyperbolic surface evolution and this (generally nonconvex) energy minimization. This also provides a convenient framework for numerically computing anisotropic kinetic crystal growth [67]. The theoretical and numerical results of this work are illustrated in the Uniform Density Island Dynamics models of [15] and [36]. That model describes crystals growing in two dimensions with an anisotropic velocity.

An interesting spinoff of this work came in [67] in which we proved that any two dimensional Wulff shape can be interpreted precisely as the solution of a Riemann problem for a scalar conservation law – contact discontinuities correspond to jumps in the angle of the normal to the shape, smoothly varying non flat faces correspond to rarefaction waves and planar facets correspond to constant states, which develop because of kinks in the conservation law's flux function. These kinks are also seen in the convexified Wulff energy.

4.6 Epitaxial Growth of Thin Films

A new continuum model for the epitaxial growth of thin films has been developed. Molecular Beam Epitaxy (MBE) is a method for growing extremely thin films of material. The essential aspects of this growth process are as follows: under vacuum conditions a flux of atoms is deposited on a substrate material, typically at a rate that grows one atomic monolayer every several seconds. When deposition flux atoms hit the surface, they bond weakly rather than bounce off. These surface "adatoms" are relatively free to hop from lattice site to site on a flat (atomic) planar surface. However, when they hop to a site at which there are neighbors at the same level, they form additional bonds which hold them in place. This bonding could occur at the "step edge" of a partially formed atomic monolayer, which contributes the growth of that monolayer. Or, it could occur when two adatoms collide with each other. If the critical cluster size is one, the colliding adatoms nucleate a new partial monolayer "island" that will grow by trapping other adatoms at its step edges.

By these means, the deposited atoms become incorporated into the growing thin film. Each atomic layer is formed by the nucleation of many isolated monolayer islands, which then grow in area, merge with nearby islands, and ultimately fill in to complete the layer. Because the deposition flux is continually raining down on the entire surface, including the tops of the islands,

a new monolayer can start growing before the previous layer is completely filled. Thus islands can form on top of islands in a "wedding cake" fashion, and the surface morphology during growth can become quite complicated.

The Island Dynamics model is a continuum model designed to capture the longer length scale features that are likely to be important for the analysis and control of monolayer thin film growth. It is also intended to model the physics relevant to studying basic issues of surface morphology, such as the effects of noise on growth, the long time evolution of islands, and the scaling relationships between surface features (mean island area, step edge length, etc) in various growth regimes (precoalescence, coalescence). Refer to the classic work of [14] for useful background on the modeling of the growth of material surfaces. Our present discussion of the Island Dynamics Model is an abridged version of what was discussed in [54]. We shall present this new model in some detail because, although it has many of the features of the Stefan problem, it also requires some new level set technology. This includes a "wedding cake" formulation involving several level sets of the same function, nucleation of new islands, and nontrivial numerical treatment of the interface to obtain rapid convergence of implicit time marching schemes.

In the Island Dynamics model, we treat each of the islands present as having a unit height, but a continuous (step edge) boundary on the surface. This represents the idea that the films are atomic monolayers, so that height is discrete, but they cover relatively large regions on the substrate, so x and y are continuum dimensions. The adatoms are modeled by a continuous adatom density function on the surface. This represents the idea that they are very mobile, and so they effectively occupy a given site for some fraction of the time, with statistical continuity, rather than discretely.

Thus, the domain for the model is the $x-y$ region originally defined by the substrate, and the fundamental dynamical variables for this model are:

- The island boundary curves $\Gamma_i(t),\ i = 1, 2, \ldots, N$
- The adatom density on the surface $\rho(x, y, t)$

The adatom density ρ obeys a surface diffusive transport equation, with a source term for the deposition flux

$$\frac{\partial \rho}{\partial t} = \nabla \cdot (D \nabla \rho) + F,$$

where $F = F(x, y, t)$ is specified. During most phases of the growth, it is simply a constant. This equation may also include additional small loss

terms reflecting adatoms lost to the nucleation of new islands, or lost to de-absorption off the surface. This equation must be supplemented with boundary conditions at the island boundaries. In the simplest model of Irreversible Aggregation, the binding of adatoms to step edges leaves the adatom population totally depleted near island boundaries, and the boundary condition is

$$\rho|_\Gamma = 0.$$

More generally, the effects of adatom detachment from boundaries, as well as the energy barriers present at the boundary, lead to boundary conditions of the form

$$\left[A\rho + B\frac{\partial \rho}{\partial n}\right] = C$$

where C is given and $[\cdot]$ denotes the local jump across the boundary. In particular, note that ρ itself can have a jump across the boundary, even though it satisfies a diffusive transport equation. This simply reflects that fact that the adatoms on top of the island are much more likely to incorporate into the step edge than to hop across it and mix with the adatoms on the lower terrace, and vice versa.

The island boundaries Γ_i move with velocities $\vec{v} = v_N\vec{N}$, where the normal velocity v_n reflects the island growth. This is determined simply by local conservation of atoms: the total flux of atoms to the boundary from both sides times the effective area per atom, a^2, must equal the local rate of growth of the boundary, v_N:

$$v_N = -a^2[\vec{q}\cdot\vec{N}]$$

(this assumes there is no particle transport along the boundary; more generally, there is a contribution from this as well) where $\vec{q}$ is the surface flux of adatoms to the island boundary and $\vec{N}$ is the local outward normal. In general, the net atom flux $\vec{q}$ can be expressed in terms of the diffusive transport, as well as attachment and detachment probabilities, all of which can be directly related to the parameters of Kinetic Monte Carlo models. In the special case of Irreversible Aggregation, $\vec{q}$ is simply the surface diffusive flux of adatoms

$$\vec{q} = -D\nabla\rho.$$

To complete the model we include a mechanism for the nucleation of new islands. If islands nucleate by random binary collisions between adatoms

(and if the critical cluster size is one), we expect the probability that an island is nucleated at a time t, at a site (x, y), scales like

$$P[dx, dy, dt] = \epsilon\rho(x, y, t)^2 dt\ dx\ dy.$$

This model describes nucleation as a site-by-site, timestep-by-timestep random process. A simplifying alternative is to assume the nucleation occurs at the continuous rate obtained by averaging together the probabilistic rates at each site. In this case, if we let $n(t)$ denote the total number of islands nucleated prior to time t, we have the deterministic equation

$$\frac{dn}{dt} = \langle \epsilon\rho^2 \rangle$$

where $\langle \cdot \rangle$ denotes the spatial average. In this formulation, at each time when $n(t)$ reaches a new integer value, we nucleate a new island in space. This is carried out by placing it randomly on the surface with a probability weighted by ρ^2, so that the effect of random binary collisions is retained.

This basic model also has natural extensions to handle more complex thin film models. For example, additional continuum equations can be added to model the dynamics of the density of kink sites on the island boundaries, which is a microstructural property that significantly influences the local adatom attachment rates (see [15]). Also, we can couple this model to equations for the elastic stress that results from the "lattice mismatch" between the size of the atoms in the growing layers and the size of the atoms in the substrate.

Conversely, the above model has a particular interesting extreme simplification. We can go to the limit where the adatoms are so mobile on the surface ($D \to \infty$) that the adatom density is spatially uniform, $\rho(x, y, t) = \rho(t)$. In this case, the loss of adatoms due to the absorbing boundaries is assumed to take on a limiting form proportional to the adatom density and the total length L of all the island boundaries, which can be written as a simple sink term

$$\frac{d\rho}{dt} = F - \lambda L\rho.$$

(This equation can be derived systematically from the conservation law for the total number of adatoms, $\int \rho$, that follows from the adatom diffusion equation. The above loss term is just a simplified model for the net loss of adatoms to the island boundaries.) Further, it is assumed the velocity takes on a given normal dependent limiting form, $v_N = v_N(\vec{N})$ (which implies

that growing islands will rapidly assume the associated "Wulff shape" for this function $v_N(\vec{N})$ (as in [62])). We have used this "Uniform Density" model to prototype the numerical methods, and to develop an understanding of how the island dynamics models are related to the continuum "rate equation" models that describe island size distribution evolution while using no information at all about the spatial interactions of the islands.

Much of the above model is formally a Stefan problem and many of the level set techniques required for this were developed in [24] and can similarly be applied here. In addition, the internal boundary condition discretization of the adatom diffusion equation can be implemented using the symmetric matrix version of the discretization proposed by Fedkiw [31] and discussed earlier in this paper.

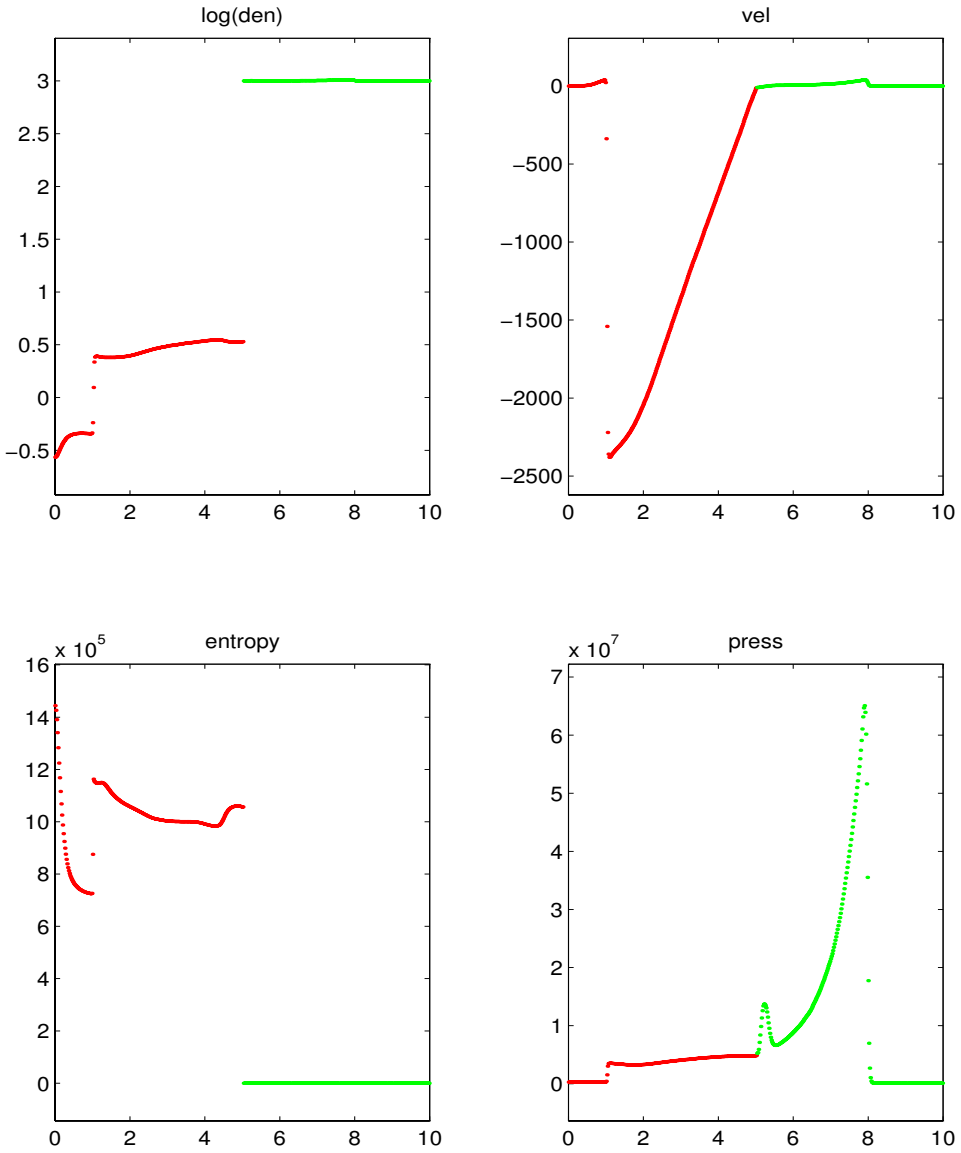


Figure 7: Two phase compressible flow calculated with the Ghost Fluid Method. Air on the left and water on the right. Reprinted from [32].

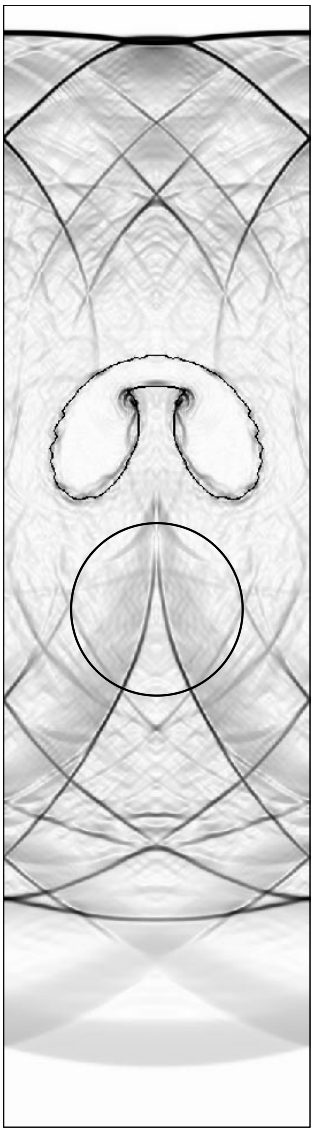


Figure 8: Mach 1.22 air shock collapse of a helium bubble. Reprinted from [32].

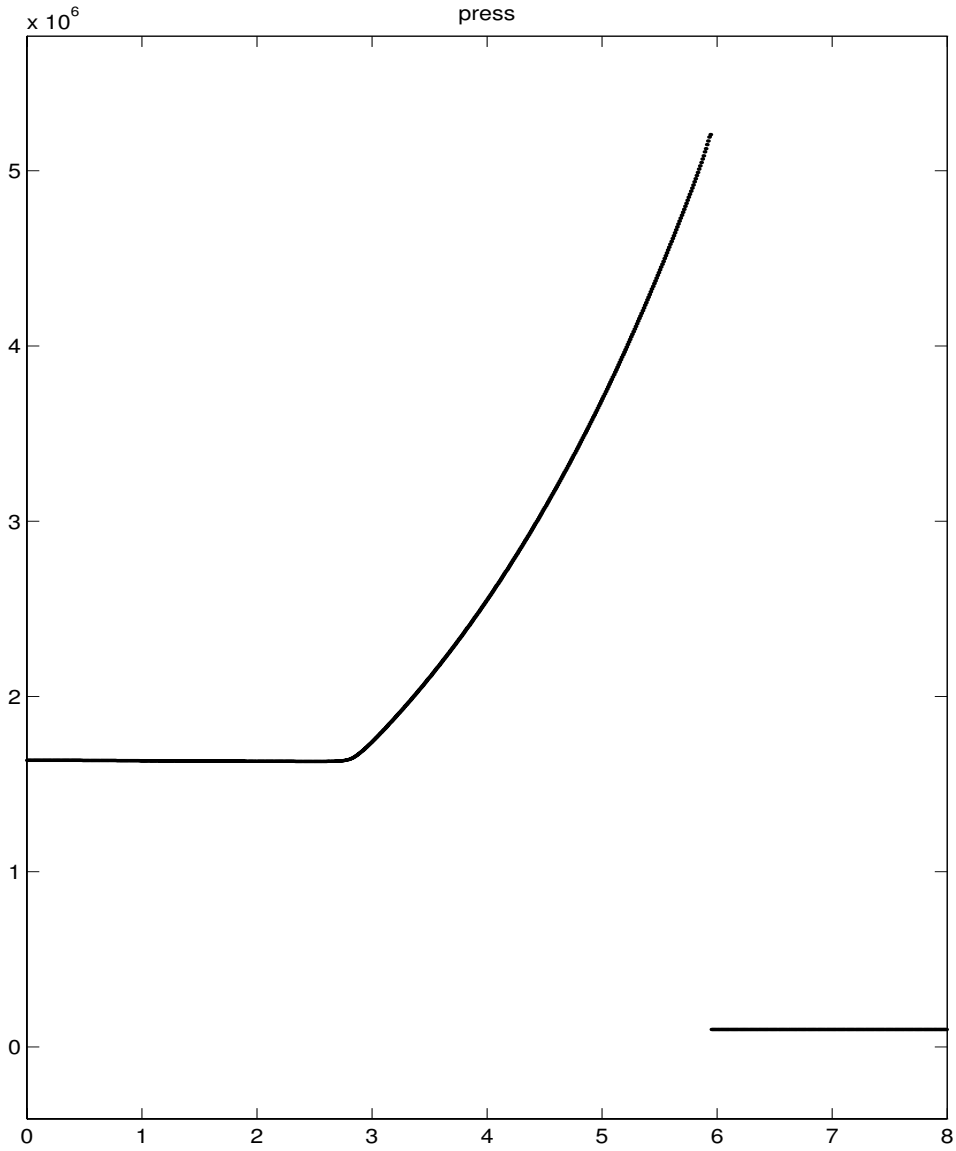


Figure 9: Nonsmeared detonation wave traveling away from a solid wall. Reprinted from [33].

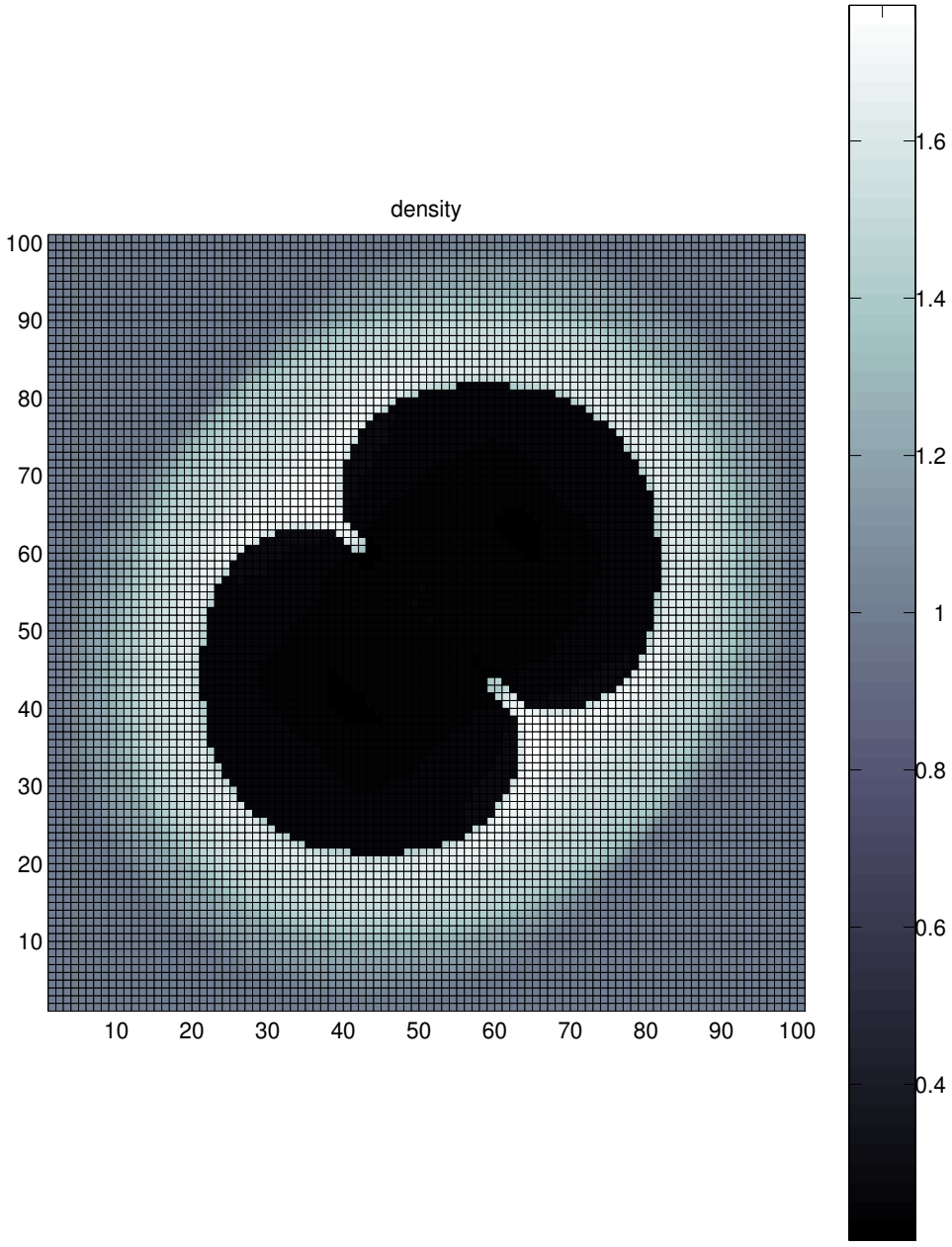


Figure 10: Two deflagration fronts depicted shortly after merging. Reprinted from [33].

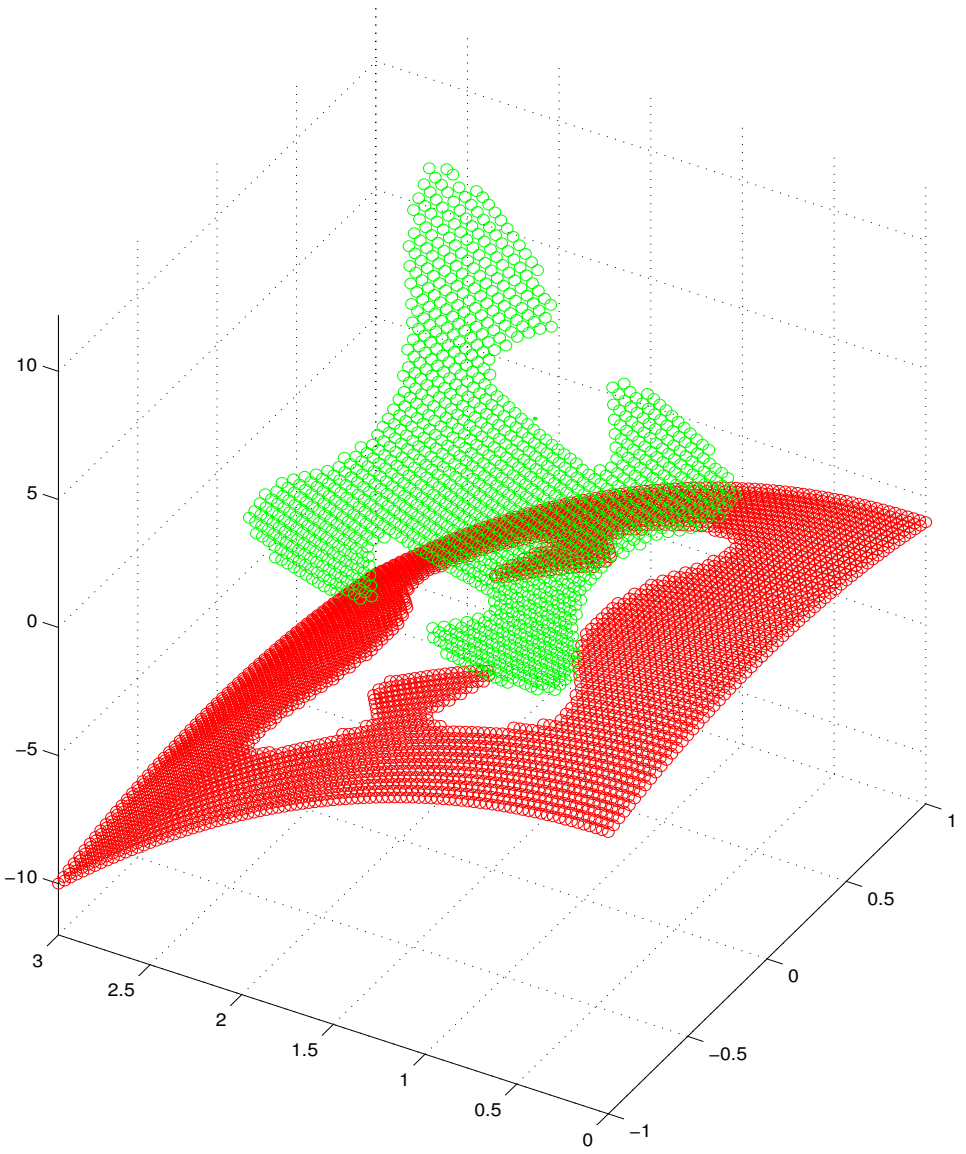


Figure 11: Two spatial dimensions, $\nabla \cdot (\frac{1}{\rho}\nabla p) = f(x,y)$, $[p] = g(x,y)$, $[\frac{1}{\rho}\nabla p \cdot \vec{N}] = h(x,y)$. Reprinted from [49].

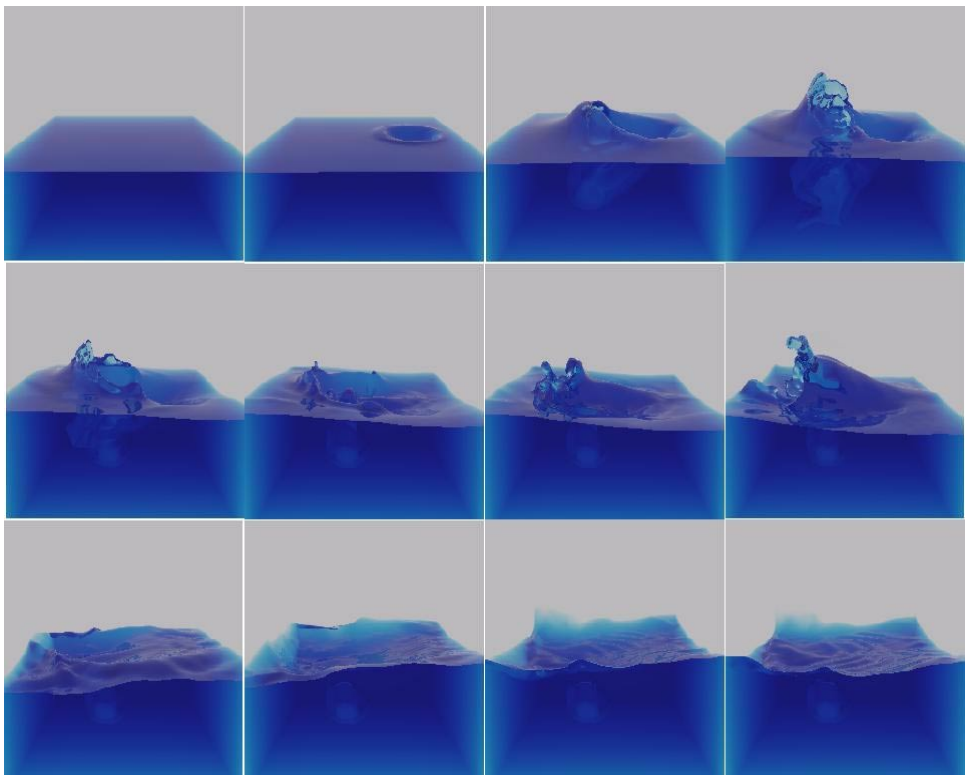


Figure 12: Water waves generated by the impact of an (invisible) solid object. Reprinted from [44].

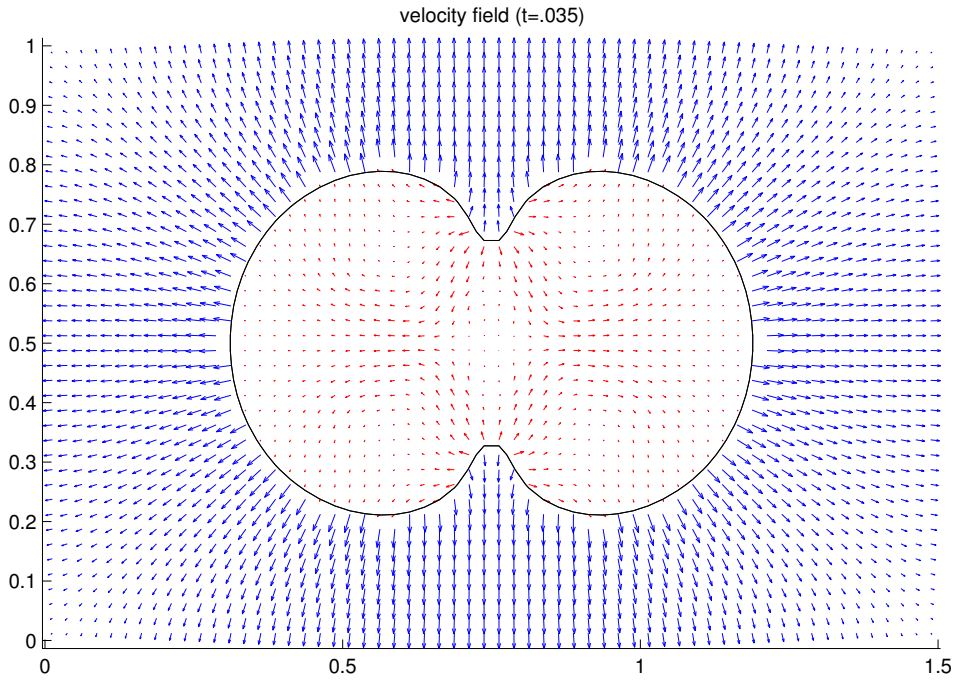


Figure 13: Two phase incompressible flames depicted shortly after merging (2 spatial dimensions). Reprinted from [57].

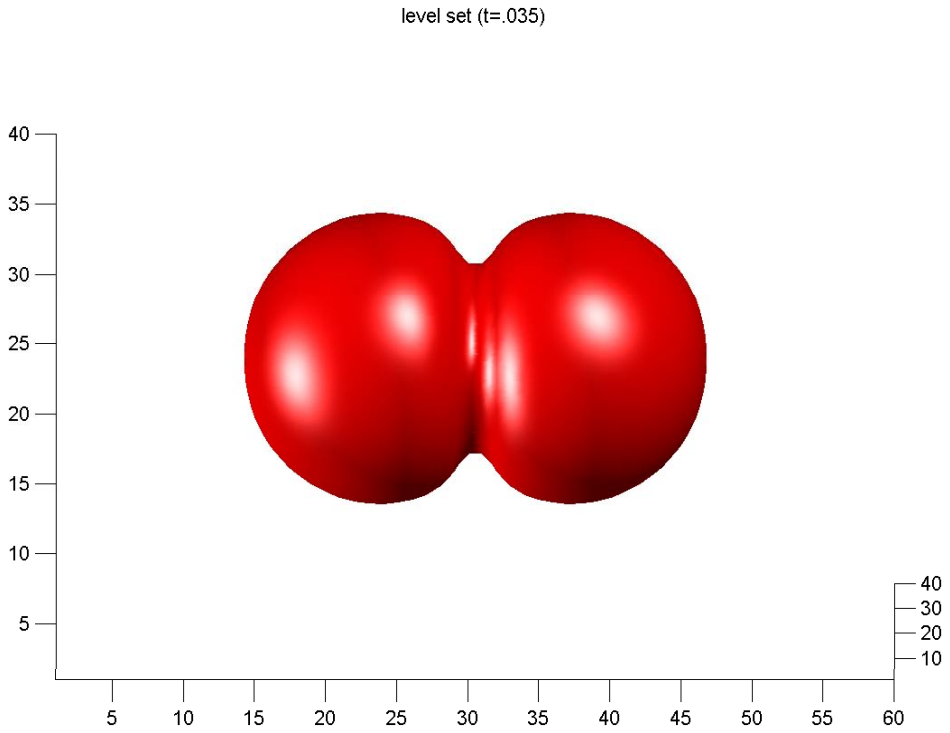


Figure 14: Two phase incompressible flames depicted shortly after merging (3 spatial dimensions). Reprinted from [57].

5 A Variational Approach with Applications to Multiphase Motion

In many situations, e.g., crystal growth, a material is composed of three or more phases. The interfaces between the phases move according to some law. If the material is a metal and its grain orientation is different in each region, then an isotropic surface energy means that the velocity is the mean curvature of the interface. Or the velocities of the interfaces may depend on the pair of phases in contact; e.g. a different constant velocity on each interface.

Several fixed grid approaches to this problem have been used. Merriman, Bence and Osher [53] have extended the level set method to compute the motion of multiple junctions. Also in that paper, and in [51] and [52], a simple method based on the diffusion of characteristic functions of each set Ω_i, followed by a certain reassignment step, was shown to be appropriate for the motion of multiple junctions in which the bulk energies are zero (and hence, the constants $e_i = 0,\ i = 1, \ldots, n$) and the $f_{i,j}$ are all equal to the same positive constant, i.e., pure mean curvature flow. See equations (21) below.

Another method using an "influence matrix" was designed in [75]. However, as cautioned by the author, the method is expensive and complex.

More general motion involving somewhat arbitrary functions of curvature, perhaps different for each interface, was proposed in [53] as well. This was implemented basically by decoupling the motions, and then using a reassignment step. Again each region has its own private level set function. This function moves each level set with a normal velocity depending on the proximity to the nearest interface, thus vacuum and overlapping regions generally develop. Then a simple reassignment step is used, removing all the spurious regions. For details see [53]. In that paper there was no restriction to gradient flows. However, the general method in [53] lacks (so far) a clean theoretical basis to guide the design of numerical algorithms. These difficulties were rectified by the following method.

In [88] we developed the variational level set approach inspired by [68]. Given a disjoint family Ω_i of regions in R^2 with the common boundary between Ω_i and Ω_j denoted by $\Gamma_{i,j}$, we associate to this geometry an energy function of the form

$$E \;=\; E_1 + E_2$$

$$
\begin{aligned}
E_1 &= \sum_{1\le i\le j\le n} f_{i,j} \text{ length } (\Gamma_{i,j}) \qquad (21)\\
E_2 &= \sum_{1\le i\le n} e_i \text{ area } (\Omega_i)
\end{aligned}
$$

where E_1 is the energy of the interface (surface tension). E_2 is bulk energy, and n is the number of phases. The gradient flow induces motion such that the normal velocity of each interface is defined in (22). At triple points (which can be seen geometrically by the triangle inequality to be the only stable junctions if all the $f_{i,j} > 0$), the angles are determined by (23) throughout the motion.

$$\text{Normal velocity of } \Gamma_{i,j} = (v_N)_{i,j} = f_{i,j}\kappa_{i,j} + (e_i - e_j). \qquad (22)$$

$$\frac{\sin\theta_1}{f_{2,3}} = \frac{\sin\theta_2}{f_{3,1}} = \frac{\sin\theta_3}{f_{1,2}}. \qquad (23)$$

This could be rewritten as:

$$
\begin{aligned}
E &= E_1 + E_2\\
E_1 &= \sum_{i=1}^{n} \gamma_i \int\int \delta(\varphi_i(x,y,t))|\nabla\varphi_i(x,y,t)|dxdy \qquad (24)\\
E_2 &= \sum_{i=1}^{n} e_i \int\int H(\varphi_i(x,y,t))dxdy,
\end{aligned}
$$

where

$$f_{i,j} = \gamma_i + \gamma_j, \quad 1 \le i < j \le n.$$

In the (most interesting) case when $n = 3$ we can solve uniquely for the γ_i.

Now our problem becomes:

Minimize E subject to the constraint that

$$\sum_{i=1}^{n} H(\varphi_i(x,y)) - 1 \equiv 0. \qquad (25)$$

This infinite set of constraints prevents the development of overlapping regions and/or vacuum. It requires that the level curves $\{(x,y)|\varphi_i(x,y,t) = 0\}$ match perfectly.

The implementation of (24) with the infinite set of constraints (25) is computationally demanding. Instead we try to replace the constraint (25) by a single constraint

$$\int\int \frac{(\sum H(\varphi_i(x,y,t)) - 1)^2}{2} dxdy = \epsilon \tag{26}$$

where $\epsilon > 0$ is as small as we can manage numerically.

The gradient projection method leads us to an interesting coupled system which involves motion of level contours of each φ with normal velocity $a + b\kappa$ together with a term enforcing the no overlap/vacuum constraint. We find that $\epsilon \approx \Delta x$ in real calculations. See [88] for details.

We have used this technique to reproduce the general behavior of complicated bubble and droplet motions in two and three dimensions [90]. The problems included soap bubble colliding and merging, drops falling or remaining attached to a generally irregular ceiling (see figure 15), liquid penetrating through an asymmetric funnel opening (see figure 16), and mercury sitting on the floor (see figure 17).

This variational approach has also been found to have many applications in computer vision – this will be discussed in the next section.

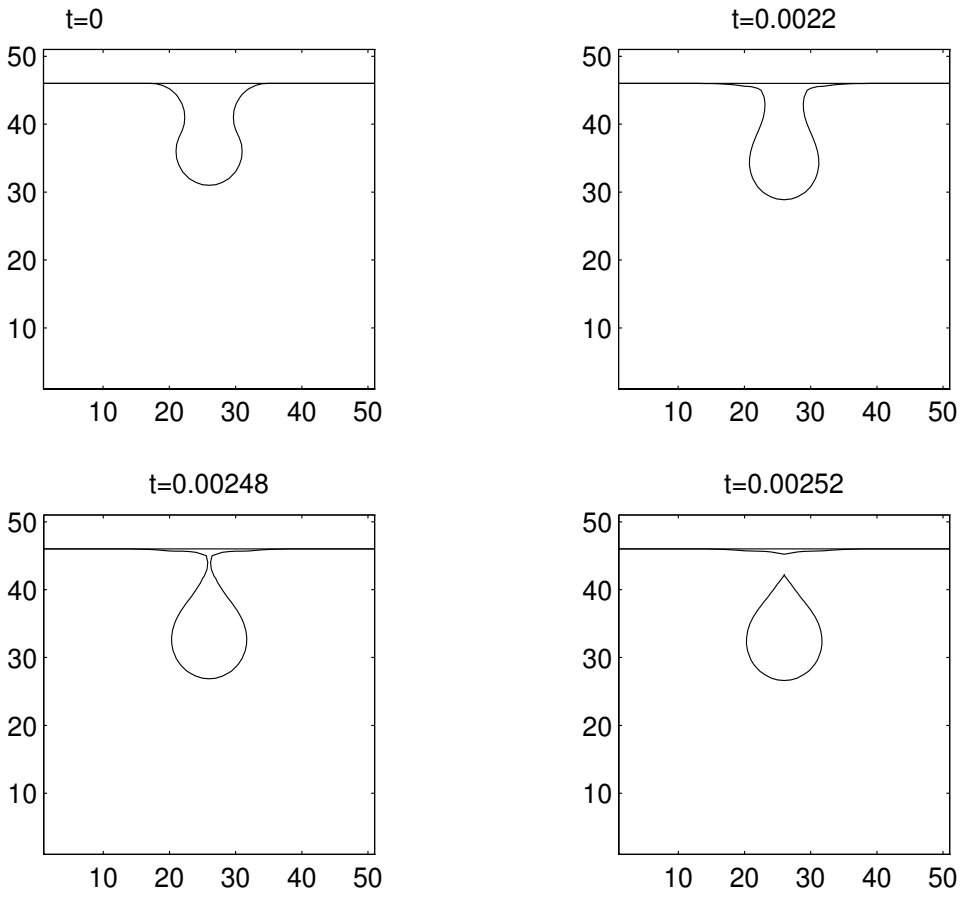


Figure 15: Three dimensional drop falling from ceiling. Reprinted from [90].

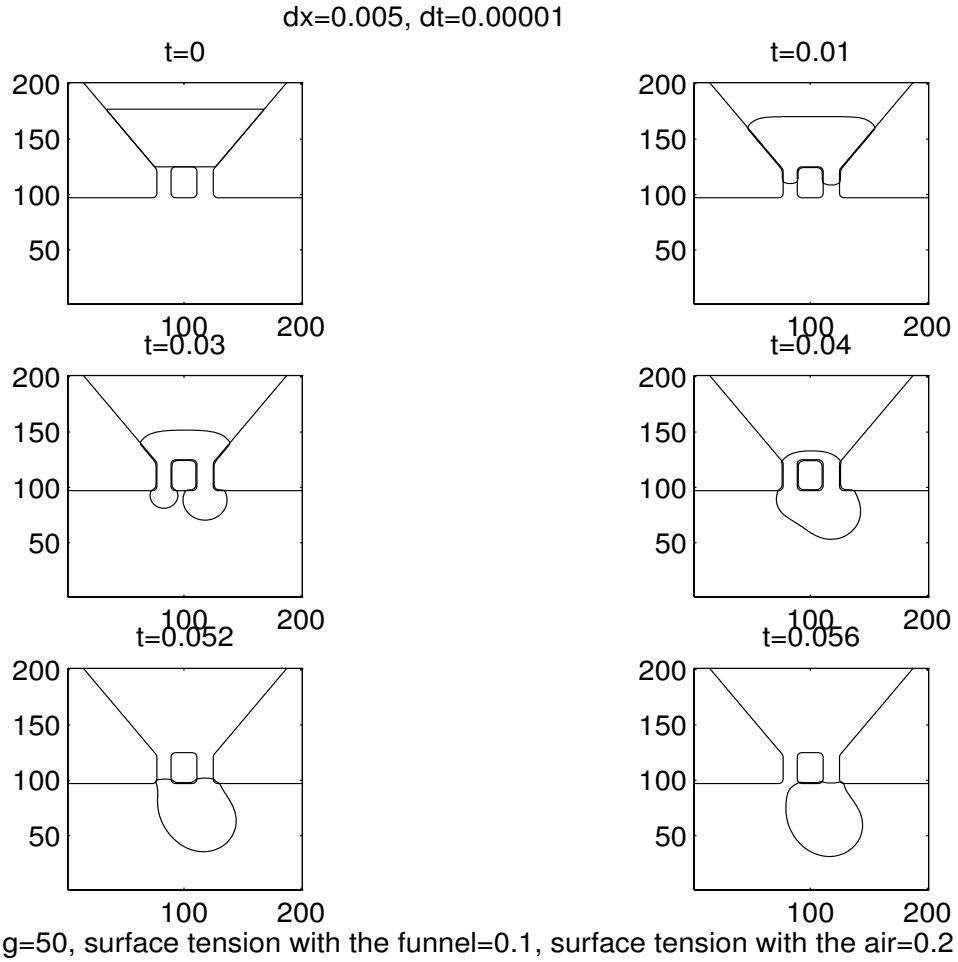


Figure 16: Liquid falling through funnel opening. Reprinted from [90].

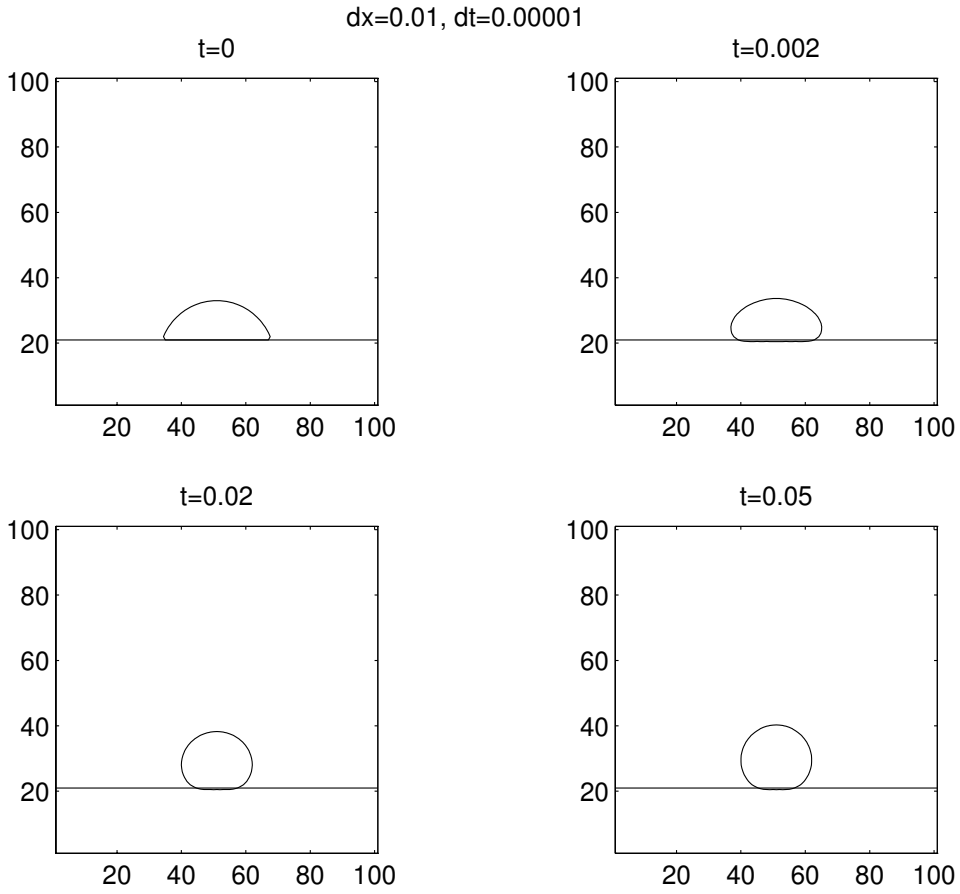


Figure 17: Mercury droplet responding to surface tension. Reprinted from [90].

6 Applications to Computer Vision and Image Processing

The use of PDE's and level set motion in image analysis and computer vision has exploded in recent years. Good references include [18] and [58].

One basic idea is to view an image as $u_0(x, y)$, a function defined on a square, and obtain a (usually second order) flow equation of the form

$$\begin{aligned} u_t &= F(u, Du, D^2u, x, t) \\ u(x, y, 0) &= u_0(x, y) \end{aligned} \tag{27}$$

which, for positive t, processes the image.

For example, if one solves the heat equation with $F(u, Du, D^2u, x, t) = \Delta u$, then $u(x, y, t)$ is the same as convolution of u_0 with a Gaussian of variance t.

L.I. Rudin, in his Ph.D. thesis [70], made the point that images are largely characterized by singularities, edges, boundaries, etc, and thus nonlinearity, especially ideas related to shock propagation, should play a role. This led to the very successful total variation based image restoration algorithms of [72] and [71]. Briefly, if we are presented with a noisy blurred image

$$u_0 = j * u + n \tag{28}$$

where j is a given convolution kernel, and the mean and variance of the noise are given, we wish to obtain the "best" restored image. This leads us (see [72] and [71]) to the evolution equation

$$u_t = \nabla \cdot \frac{\nabla u}{|\nabla u|} - \lambda j * (j * u - u_0) \tag{29}$$

to be solved for $t > 0$, where $u(x, y, 0)$ is given, and $\lambda(t) > 0$ is obtained as a Lagrange multiplier, or is set to be a fixed constant. If $j * u = u$, this becomes a pure denoising problem. The (very interesting) geometric interpretation of this procedure is that each level contour of u is moved normal to itself with velocity equal to its curvature, divided by the norm of the gradient of u, then "pulled back" in an attempt to deconvolve (28). The results are state-of-the-art for many problems. Noisy regions can be thought of as corresponding to contours having very high curvature, while edges have finite curvature and infinite gradients.

Here the motion of level sets is just used to interpret the dynamics. In [4], it was shown that reasonable axioms of image processing lead to the

remarkable fact that motion of level contours by a function of curvature is fundamental to the subject. The artificial time t is actually the scale parameter [4].

We would like to describe a few new applications of this set of ideas. In [10], we have considered the problem of processing of images defined on manifolds. The technique actually can be used to solve a wide class of elliptic equations on manifolds, without triangulation, using only a local Cartesian grid, for very general situations.

Given a manifold in R^3, defined by $\psi(x, y, z) = 0$, we can define the projection matrix

$$P_{\nabla\psi} = I - \frac{\nabla\psi}{|\nabla\psi|} \otimes \frac{\nabla\psi}{|\nabla\psi|}. \tag{30}$$

If u is an image defined on $\psi = 0$ we can use our level set calculus to extend it constant normal to the manifold, in some neighborhood of the manifold.

If u_0 is the original noisy image, the energy to be minimized is

$$E(u) = \int_{R^3} |P_{\nabla\psi}\nabla u|\delta(\psi)|\nabla\psi| d\vec{x} + \frac{\lambda}{2}\int (u - u_0)^2\delta(\psi)|\nabla\psi| d\vec{x}.$$

Using the gradient descent algorithm, i.e. following the general procedure of [72] and [88] leads us to

$$u_t = \frac{1}{|\nabla\psi|}\nabla \cdot \left(\frac{P_{\nabla\psi}\nabla u}{|P_{\nabla\psi}\nabla u|}|\nabla\psi|\right) - \lambda(u - u_0).$$

This corresponds to total variation denoising. This is done using the local level set method [66] which allows great flexibility in geometry, while always using a Cartesian grid. See [10] for denoising and deblurring results.

The technique is quite general – both variational problems and PDE's defined on manifolds can be solved in a reasonably straightforward fashion, without restrictions on the manifold and without complicated triangulation – just by using a fixed Cartesian grid.

Another basic image processing task is to detect objects hidden in an image u_0. A popular technique is called active contours or snakes, in which one evolves a curve, subject to constraints until the curve surrounds the image.

The level set method was first used in [16] as a very convenient tool to follow the motion of active contours in order to surround hidden objects.

This was an important step since topological changes could easily be handled, a variational approach could be easily used [17] and stable, easy to program algorithms resulted.

The curve is moved with a velocity which vanishes when the object is surrounded. Thus edge detectors are traditionally used to stop the evolving curve. For example, one might use

$$g(|\nabla u_0|) = \left(\frac{1}{1+|\nabla j_\sigma * u_0|}\right)^2$$

where j_σ is a Gaussian of variance σ.

In [20] the authors developed a model which was not based on edges, using a scale parameter, based on a simplification of the Mumford-Shah [56] energy based segmentation. The implementation is done through the variational level set approach [88] and the results are remarkable. The method has a denoising capability as well as the ability to perform a multiscale segmentation. See [21] and [20] for details. Here we just present the evolution equation for the level set function φ:

$$\varphi_t = |\nabla\varphi|\left[\mu\nabla\cdot\frac{\nabla\varphi}{|\nabla\varphi|} - \nu - \lambda(u_0-c_1)^2 + \lambda(u_0-c_2)^2\right]$$

for parameters $\mu, \nu, \lambda \geq 0$, where c_1 and c_2 are the averages of u_0 over the region for which $\varphi \geq 0$ and $\varphi \leq 0$ respectively. ν corresponds to the bulk energy of the area for which $\varphi \geq 0$, μ corresponds to the surface tension of the interface, and λ is the penalty for the L^2 error between u_0 and its mean over each region. Figure 18 shows an active contour segmenting a MRI brain image from its backgound.

A somewhat related problem as discussed in [89] is the following. Given a collection of unorganized points, and/or curves, and/or surface patches, find a surface which can be regarded as its shape. This is a fundamental visualization problem which arises in computer graphics, visualization and simulation. No assumptions about the ordering, connectivity or topology of the data sets or of the true shape is given. The input is the general distance to the data set which is given on a (usually logically rectangular) grid. Additionally, we may also input the values of the normal to the surface at the same or different data points.

The key idea is to find a function φ whose zero level set is the interpolating surface, φ changes sign as one goes from inside to outside the surface. The output is the discrete values of φ, which can be reinitialized to be signed distance to this surface.

We set up a variational problem, which basically minimizes the integral over the unknown surface, of the pth power of distance to the data set. We may include information about the normals in analogous fashion.

Gradient descent (as in the image restoration and active contour problems) gives us a weighted motion by curvature plus convection algorithm. The results are very promising as shown figure 19. For more details, see [89].

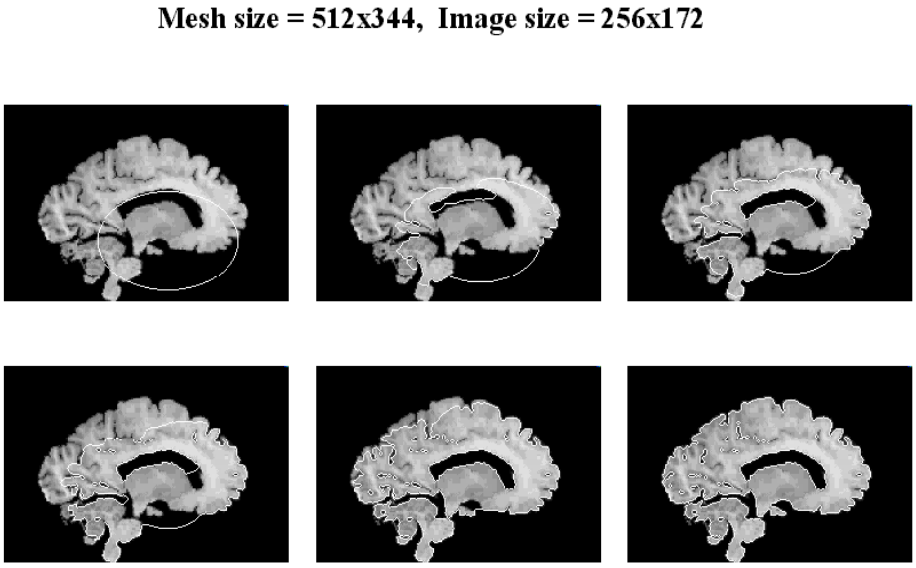


Figure 18: Active contour segmentation of an MRI brain image from its backgound. Reprinted from [19].

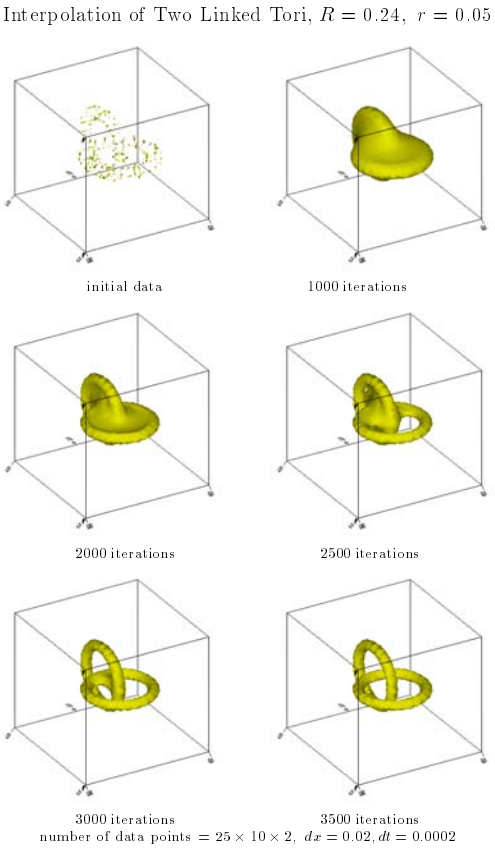


Figure 19: Interpolation of two linked tori. Reprinted from [89].

7 Conclusion

The idea of using a level set to represent an interface is a very old one. The level set method itself has antecedents, for example, in the G equation approach of Markstein [50]. What is new is the level set method technology, theoretical justification through viscosity solutions, and the enormous number of wide ranging applications that are now available, with new applications developing quite frequently.

References

[1] Adalsteinsson, D. and Sethian, J.A., *The Fast Construction of Extension Velocities in Level Set Methods*, J. Comput. Phys. 148, 2-22 (1999).

[2] Adalsteinsson, D. and Sethian, J.A. *A Fast Level Set Method for Propagating Interfaces*, J. Comput. Phys. 118, 269-277 (1995).

[3] Adalsteinsson, D. and Sethian, J.A., *A Level Set Approach to a Unified Model for Etching, Deposition, and Lithography II: Three Dimensional Simulations*, J. Comput. Phys. 122, 348-366 (1995).

[4] Alvarez, L., Guichard, F., Lions, P.-L., and Morel, J.-M., *Axioms and Fundamental Equations of Image Processing*, Arch. Ration. Mech. and Anal. 123, 199-257, (1993).

[5] Ambrosio, L. and Soner, H.M., *Level Set Approach To Mean Curvature Flow in Arbitrary Codimension*, J. Diff. Geom. 43 (4) 693-737 (1996).

[6] Bardi, M. and Evans, L.C., *On Hopf's Formulas for Solutions of Hamilton-Jacobi Equations*, Nonlinear Analysis TMA 8, 1373-1381 (1984).

[7] Bardi, M. and Osher, S., *The Nonconvex Multidimensional Riemann Problem for Hamilton-Jacobi Equations*, SIAM J. on Anal. 22, 344-351 (1991).

[8] Barles, G., *Solutions de Viscosite des Equations de Hamilton-Jacobi*, Springer-Verlag, Berlin (1966).

[9] Bellettini, G., Novaga, M. and Paolini, M., *An Example of Three Dimensional Fattening for Linked Space Curves Evolving by Curvature*, Comm. of Partial Diff. Equations (in press).

[10] Bertalmio, M., Cheng, L.T., Osher, S., and Sapiro, G., *Variational Problems and Partial Differential Equations on Implicit Surfaces: The Framework and Examples in Image Processing and Pattern Formation*, UCLA CAM Report 00-23, J. Comput. Phys. (in review).

[11] Boue, M. and Dupuis, P., *Markov Chain Approximations for Deterministic Control Problems with Affine Dynamics and Quadratic Cost in the Control*, SIAM J. Numer. Anal. 36 (3), 667-695 (1999).

[12] Brackbill, J.U., Kothe, D.B. and Zemach, C., *A Continuum Method for Modeling Surface Tension*, J. Comput. Phys. 100, 335-354 (1992).

[13] Burchard, P., Cheng, L.-T., Merriman, B., and Osher, S., *Motion of Curves in Three Spatial Dimensions Using a Level Set Approach*, UCLA CAM Report 00-29, J. Comput. Phys. (in review).

[14] Burton, W.K., Cabrera, N. and Frank, F.C., *The Growth of Crystals and the Equilibrium Structure of Their Surfaces*, Phil. Trans. Roy. Soc. London, Ser. A, pp. 243-299, (1951).

[15] Caflisch, R.E., Gyure, M., Merriman, B., Osher, S., Ratsch, C., Vvedensky, D. and Zinck, J., *Island Dynamics and the Level Set Method for Epitaxial Growth*, Appl. Math. Lett. 12, 13-22 (1999).

[16] Caselles, V., Catté, F., Coll, T., and Dibo, F., *A Geometric Model for Active Contours in Image Processing*, Numerische Mathematik 66, 1-31 (1993).

[17] Caselles, V., Kimmel, R. and Sapiro, G., *Geodesic Active Contours*, Int. J. Comput. Vision 22, 61-79 (1997).

[18] Caselles, V., Morel, J.-M., Sapiro, G., and Tannenbaum, A. Editors, Special Issue on Partial Differential Equations and Geometry-Driven Diffusion in Image Processing and Analysis, IEEE Transactions on Image Processing, (1998), v. 7, pp. 269-473.

[19] Chan, T., Fedkiw, R., Kang, M. and Vese, L., *Improvements in the Efficiency and Robustness of Active Contour Algorithms*, (in preparation).

[20] Chan, T. and Vese, L., *Active Contours Without Edges*, UCLA CAM Report 98-53 (1998).

[21] Chan, T. and Vese, L., *An Active Contour Model Without Edges*, in Lecture Notes in Comp. Sci., v. 1687, eds. M. Neilsen, P. Johansen, O.F. Olsen and J. Weickert, pp. 141-151, 1999.

[22] Chang, Y.C., Hou, T.Y., Merriman, B. and Osher, S., *A Level Set Formulation of Eulerian Interface Capturing Methods for Incompressible Fluid Flows*, J. Comput. Phys. 124, 449-464 (1996).

[23] Chen, Y.G., Giga, Y. and Goto, S., *Uniqueness and Existence of Viscosity Solutions of Generalized Mean Curvature Flow Equations*, J. Diff. Geom. 33, 749-786 (1991).

[24] Chen, S., Merriman, B., Osher, S. and Smereka, P., *A simple level set method for solving Stefan problems*, J. Comput. Phys. 135, 8-29 (1997).

[25] Cheng, L.T., Fedkiw, R.P., Gibou, F. and Kang, M., *A Symmetric Method for Implicit Time Discretization of the Stefan Problem*, J. Comput. Phys. (in review).

[26] Colella, P., Majda, A., and Roytburd, V., *Theoretical and Numerical Structure for Reacting Shock Waves*, SIAM J. Sci. Stat. Comput. 7 (4), 1059-1080 (1986).

[27] Crandall, M.G., Ishii, H. and Lions, P.-L., *User's Guide to Viscosity Solutions of Second Order Partial Differential Equations*, Amer. Math. Soc. Bull. 27, 1-67 (1992).

[28] DeGiorgi, E., *Barriers, Boundaries, Motion of Manifolds*, Lectures in Pavia, Italy, 1994.

[29] Evans, Y.C. Soner, H.M. and Souganidis, P.E., *Phase Transitions and Generalized Motion by Mean Curvature*, Comm. Pure and Applied Math. 65, 1097-1123 (1992).

[30] Evans, Y.C. and Spruck, J., *Motion of Level Sets by Mean Curvature I*, J. Diff. Geom. 33, 635-681 (1991).

[31] Fedkiw. R.P. *A Symmetric Spatial Discretization for Implicit Time Discretization of Stefan Type Problems*, (unpublished) June 1998.

[32] Fedkiw, R., Aslam, T., Merriman, B., and Osher, S., *A Non-Oscillatory Eulerian Approach to Interfaces in Multimaterial Flows (The Ghost Fluid Method)*, J. Comput. Phys. 152 (2), 457-492 (1999).

[33] Fedkiw, R., Aslam, T., and Xu, S., *The Ghost Fluid Method for Deflagration and Detonation Discontinuities*, J. Comput. Phys. 154 (2), 393-427 (1999).

[34] Fedkiw, R. and Liu, X.-D., *The Ghost Fluid Method for Viscous Flows*, Progress in Numerical Solutions of Partial Differential Equations, Arcachon, France, edited by M. Hafez, July 1998.

[35] Gross, R., *Zur Theorie des Washstrums und Losungsforganges Kristalliner Materie*, Abhandl. Math.-Phys. Klasse Kongl. Sachs, Wiss, 35, pp. 137-202 (1918).

[36] Gyure, M., Ratsch, C., Merriman, B., Caflisch, R.E., Osher, S., Zinck, J. and Vvedensky, D. *Level Set Methods for the Simulation of Epitaxial Phenomena*, Phys. Rev. E 59, R6927-6930 (1998).

[37] Harabetian, E. and Osher, S., *Regularization of Ill-Posed Problems via the Level Set Approach*, SIAM J. Appl. Math. 58, 1689-1706 (1998).

[38] Harabetian, E., Osher, S. and Shu, C.-W., *An Eulerian Approach for Vortex Motion Using a Level Set Approach*, J. Comput. Phys. 127, 15-26 (1996).

[39] Helenbrook, B.T., Martinelli, L. and Law, C.K. *A Numerical Method for Solving Incompressible Flow Problems with a Surface of Discontinuity*, J. Comput. Phys. 148, 366-396 (1999).

[40] Helmsen, J., Puckett, E., Colella, P. and Dorr, M., *Two New Methods for Simulating Photolithography Development in 3D*, Proc. SPIE 2726, 253-261 (1996).

[41] Hou, T., *Numerical Solutions To Free Boundary Problmes*, ACTA Num. 4, 335-416 (1995).

[42] Hou, T., Li, Z., Osher, S. and Zhao, H.-K., *A Hybrid Method for Moving Interface Problems with Application to the Hele-Shaw Flow*, J. Comput. Phys. 134, 236-252 (1997).

[43] Jiang, G.-S. and Peng, D., *Weighted ENO Schemes for Hamilton Jacobi Equations*, SIAM J. Sci. Comput. 21, 2126-2143 (2000).

[44] Kang, M., Fedkiw, R., and Liu, X.-D., *A Boundary Condition Capturing Method for Multiphase Incompressible Flow*, UCLA CAM Report 99-21, J. Comput. Phys. (in review).

[45] Karni, S., *Hybrid multifluid algorithms*, SIAM J. Sci. Comput. 17 (5), 1019-1039 (1996).

[46] Karni, S., *Multicomponent Flow Calculations by a Consistent Primitive Algorithm*, J. Comput. Phys 112, 31-43 (1994).

[47] Kim, Y.-T., Goldenfeld, N. and Dantzig, J., *Computation of Dendritic Microstructures using a Level Set Method*, Physical Review E 62, (2000).

[48] Kobayashi, R., *Modeling and Numerical Simulations of Dendritic Crystal Growth*, Physica D 63, 410 (1993).

[49] Liu, X.-D., Fedkiw, R.P. and Kang, M., *A Boundary Condition Capturing Method for Poisson's Equation on Irregular Domains*, J. Comput. Phys. 160, 151-178 (2000).

[50] Markstein, G.H., *Nonsteady Flame Propagation*, Pergamon Press, Oxford 1964.

[51] Mascarenhas, P., *Diffusion Generated Motion by Mean Curvature*, UCLA CAM Report 92-33, 1992.

[52] Merriman, B., Bence, J. and Osher, S., *Diffusion Generated Motion by Mean Curvature*, in AMS Select Lectures in Math., The Comput. Crystal Grower's Workshop, edited by J. Taylor, AMS Providence, 1993, pp. 73-83.

[53] Merriman, B., Bence, J. and Osher, S., *Motion of Multiple Junctions: A Level Set Approach*, J. Comput. Phys. 112 (2), 334-363 (1994).

[54] Merriman, B., Caflisch, R. and Osher, S., *Level Set Methods with an Application to Modelling the Growth of Thin Films*, in Free Boundary Value Problems, Theory and Applications, pp. 51-70, edited by I. Athanasopoulos, G. Makrikis, and J.F. Rodriguez, CRC Press, Boca Raton, FL 1999.

[55] Mulder, W., Osher, S., and Sethian, J.A., *Computing Interface Motion in Compressible Gas Dynamics*, J. Comput. Phys. 100, 209-228 (1992).

[56] Mumford, D., and Shah, J., *Optimal Approximation by Piecewise Smooth Functions and Associated Variational Problems*, Comm. Pure Appl. Math. 42, 577-685 (1989).

[57] Nguyen, D., Fedkiw, R.P. and Kang, M. *A Boundary Condition Capturing Method for Incompressible Flame Discontinuities*, UCLA CAM Report 00-19, J. Comput. Phys. (in review).

[58] Nielsen, M., Johansen, P., Olsen, O.F., and Weickert, J., editors, *Scale Space Theories in Computer Vision*, in Lecture Notes in Computer Science, v. 1682, Springer-Verlag, Berlin, (1999).

[59] Nochetto, R.H., Paolini, M. and Verdi, C., *An Adaptive Finite Element Method for Two Phase Stefan Problems in Two Space Dimensions, Part II: Implementation and Numerical Experiments*, SIAM J. of Sci. Comput. 12, p. 1207 (1991).

[60] Noh, W.F. and Woodward, P.R., *SLIC (Simple Line Interface Construction)*, in Lecture Notes in Physics, v. 59, edited by A. van de Vooren and P.J. Zandbergen, Springer-Verlag, Berlin (1976), p. 330.

[61] Osher, S. and Helmsen, J., *A Generalized Fast Algorithm with Applications to Ion Etching*, (in progress).

[62] Osher, S. and Merriman, B., *The Wulff Shape as the Asymptotic Limit of a Growing Crystalline Interface*, Asian J. Math. 1 (3), 560-571 (1997).

[63] Osher, S., *A Level Set Formulation for the Solution of the Dirichlet Problem for Hamilton-Jacobi Equations*, SIAM J. on Anal. 24, 1145-1152 (1993).

[64] Osher, S. and Sethian, J.A., *Fronts Propagating with Curvature Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations*, J. Comput. Phys. 79, 12-49 (1988).

[65] Osher, S. and Shu, C.W., *High Order Essentially Non-Oscillatory Schemes for Hamilton-Jacobi Equations*, SIAM J. Numer. Anal. 28 (4), 907-922 (1991).

[66] Peng, D., Merriman, B., Osher, S., Zhao, H.-K., and Kang, M., *A PDE-Based Fast Local Level Set Method*, J. Comput. Phys. 155, 410-438 (1999).

[67] Peng, D., Osher, S., Merriman, B. and Zhao, H.-K., *The Geometry of Wulff Crystal Shapes and Its Relations with Riemann Problems*, in Contemporary Mathematics 238, 251-303, edited by G.-Q. Chen and E. DeBenedetto, AMS, Providence, RI, 1999.

[68] Reitich, F. and Soner, H.M., *Three Phase Boundary Motions under Constant Velocities I, the Vanishing Surface Tension Limit*, Proc. Royal Soc. Edinburgh 126A, 837-865 (1996).

[69] Rouy, E. and Tourin, A., *A Viscosity Solutions Approach to Shape-From-Shading*, SIAM J. Num Anal. 29 (3) 867-884 (1992).

[70] Rudin, L.I., *Images, Numerical Analysis of Singularities, and Shock Filters*, Ph.D. thesis, Computer Science Dept., Caltech, #5250:TR:87 (1987).

[71] Rudin, L.I. and Osher, S., *Total Variation Based Restoration with Free Local Constraints*, Proc. ICIP, IEEE Int'l Conf. on Image Processing, Austin, TX, (1994), pp. 31-35.

[72] Rudin, L.I., Osher, S. and Fatemi, E., *Nonlinear Total Variation Based Noise Removal Algorithms*, Physica D 60, 259-268 (1992).

[73] Ruuth, S., Merriman and Osher, S., *A Fixed Grid Method for Capturing the Motion of Self-Intersecting Interfaces and Related PDEs*, UCLA CAM Report 99-22, 1999, J. Comput. Phys. (in review).

[74] Ruuth, S., Merriman, B., Xin, J. and Osher, S., *Diffusion-Generated Motion for Mean Curvature of Filaments*, UCLA CAM Report 98-47, 1998, Comm. Pure and Applied Math (in review).

[75] Sethian, J.A., *Algorithms for Tracking Interfaces in CFD and Materials Science*, Ann. Rev. of Comput. Fluid Mech. (1995).

[76] Sethian, J.A., *Fast Marching Level Set Methods for Three Dimensional Photolithography Development*, Proc. SPIE 2726, 261-272 (1996).

[77] Sethian, J.A., *Fast Marching Methods*, SIAM Review 41, 199-235 (1999).

[78] Sethian, J.A. and Strain, J., *Crystal Growth and Dendritic Solidification*, J. Comput. Phys. 98, 231-253 (1992).

[79] Schwarz, K.W., *Simulations of Dislocations on the Mesoscopic Scale. I. Methods and Examples*, J. Applied Phys. 85, 108-119 (1999).

[80] Schwarz, K.W., *Simulations of Dislocations on the Mesoscopic Scale. II. Application to Strained-Layer Relaxation*, J. Applied Phys. 85, 120-129 (1999).

[81] Soravia, P., *Generalized Motion of a Front Propagating Along Its Normal Direction: A Differential Games Approach*, Nonlinear Analysis TMA 22, 1247-1262 (1994).

[82] Steinhoff, J., Fan, M. and Wang, L., *A New Eulerian Method for the Computation of Propagating Short Acoustic and Electromagnetic Pulses*, J. Comput. Phys. 157, 683-706 (2000).

[83] Sussman, M., Fatemi, E., Smereka, P. and Osher, S., *An Improved Level Set Method for Incompressible Two-Phase Flow*, Computers and Fluids 27, 663-680 (1998).

[84] Sussman, M., Smereka, P. and Osher, S., *A Level Set Approach for Computing Solutions to Incompressible Two-Phase Flow*, J. Comput. Phys. 114, 146-159 (1994).

[85] Tsai, R., Zhao, H.-K. and Osher, S. *Fast Sweeping Algorithms for a Class of Hamilton-Jacobi Equations*, (in preparation).

[86] Tsitsiklis, J.N., *Efficient Algorithms for Globally Optimal Trajectories*, IEEE Transactions on Automatic Control 40 (9), 1528-1538 (1995).

[87] Unverdi, S.O. and Tryggvason, G., *A Front-Tracking Method for Viscous, Incompressible, Multi-Fluid Flows*, J. Comput. Phys. 100, 25-37 (1992).

[88] Zhao, H.-K., Chan, T., Merriman, B. and Osher, S., *A Variational Level Set Approach to Multiphase Motion*, J. Comput. Phys. 127, 179-195 (1996).

[89] Zhao, H.-K., Merriman, B., Osher, S. and Kang, M., *Implicit Nonparametric Shape Reconstruction from Unorganized Points using a Variational Level Set Method*, UCLA CAM Report 98-7, 1998, Computer Vision and Image Understanding (to appear).

[90] Zhao, H.-K., Merriman, B., Osher, S. and Wang, L., *Capturing the Behavior of Bubbles and Drops Using the Variational Level Set Approach*, J. Comput. Phys. 143, 495-518 (1998).

Shock Capturing, Level Sets and PDE Based Methods in Computer Vision and Image Processing: A Review on Osher's Contribution

Written on the occasion of Stanley Osher's 60th birthday

Ronald P. Fedkiw[1], Guillermo Sapiro[2] and Chi-Wang Shu[3]

ABSTRACT

In this paper we review the algorithm development and applications in high resolution shock capturing methods, level set methods and PDE based methods in computer vision and image processing. The emphasis is on Stanley Osher's contribution in these areas and in the impact of his work. We will first review the linear stability results for hyperbolic systems. This will be followed by shock capturing methods and we will review the Engquist-Osher scheme, TVD schemes, entropy conditions, ENO and WENO schemes and numerical schemes for Hamilton-Jacobi type equations. Among level set methods we will review implicit surfaces, the setup of level set methods, numerical techniques, fluids and materials, variational approach, high codimension motion, geometric optics, and the computation of discontinuous solutions to Hamilton-Jacobi equations. Among computer vision and image processing we will review the total variation model for image denoising, images on implicit surfaces, and the level set method in image processing and computer vision.

Key Words: shock capturing method, level set method, computer vision, image processing, linear stability.

[1]Department of Computer Science, Stanford University, Stanford, CA 94305, E-mail: fedkiw@cs.stanford.edu.

[2]Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455, E-mail: guille@mail.ece.umn.edu.

[3]Division of Applied Mathematics, Brown University, Providence, RI 02912, E-mail: shu@cfm.brown.edu.
Research supported by ARO grant DAAD19-00-1-0405, NSF grants DMS-9804985 and ECS-9906606, NASA Langley grant NCC1-01035 and AFOSR grant F49620-99-1-0077.

1 Introduction

This paper is written on the occasion of Stanley Osher's 60th birthday and serves as a review article on a few selected areas in linear stability, high resolution shock capturing schemes, level set methods, and PDE based methods in computer vision and image processing. The emphasis is on Stanley Osher's contribution in these areas and in the impact of his work.

The study of linear stability for finite difference and other numerical methods for hyperbolic, parabolic, and other types of PDEs is very important. Many important results related to linear stability, especially those for initial-boundary value problems, were obtained in the 60s and 70s. Even today, linear stability results are still crucial for linear and nonlinear problems, for they provide a necessary condition for any scheme to perform nicely.

Shock capturing numerical methods have seen revolutionary developments over the past 20 years. These are methods which deal with the numerical solutions of PDEs with discontinuous solutions. Such PDEs include nonlinear hyperbolic systems such as Euler equations of compressible gas dynamics. The problems are difficult because traditional linear numerical methods are either too diffusive, or give unphysical oscillations near the discontinuities which can lead to nonlinear instabilities. The class of high resolution numerical methods overcomes this difficulty to a large extent.

Level set methods have seen tremendously expanded applications in many areas over the past 15 years. This has been made possible by the flexibility of the level set formulation in dealing with dynamic evolutions and topological changes of curves and surfaces, and by the mathematical theory and numerical tools developed in the past 15 years in studying these methods.

PDE based methods in computer vision and image processing have been actively studied in the past few years. Again, the rapid development of mathematical models, solution tools such as level set methods, and high resolution numerical schemes has made PDE based method one of the major tools in computer vision and image processing.

Stanley Osher has made influential contributions to all these fields. A distinctive feature

of his research is that he emphasizes both fundamental problems in algorithm design and analysis, and practical considerations for the applications of the algorithms. This seems also to be the objective of the Journal of Computational Physics. It is thus not a surprise that a significant portion of Osher's journal publications have appeared in the Journal of Computational Physics. This is particularly the case for Osher's work over the past 15 years. Osher's work has been highly influential according to citation statistics. For example, according to the ISI database, which lists papers in selected journals of high impact since 1975, the 82 papers of Osher listed there have been collectively cited 2,386 times (as of November 20, 2001, the same below). Among these, 11 papers have been cited over 100 times each. The top five highly cited papers are: the paper of Osher and Sethian [147] on level set methods, cited 472 times; the paper of Harten, Engquist, Osher and Chakravarthy [76] on ENO schemes, cited 314 times; the two papers of Shu and Osher [171, 172] on ENO schemes, cited 235 and 231 times respectively; and the paper of Harten and Osher [75] on UNO schemes, cited 189 times. We remark that the top four among these five most highly cited papers of Osher were published in the Journal of Computational Physics. The other papers of Osher having a citation over 100 include: the paper of Osher and Solomon on upwind schemes [149], cited 180 times; the paper of Sussman, Smereka and Osher on level set methods for incompressible two phase flows [177], cited 137 times; the paper of Osher on Riemann solvers and entropy conditions [135], cited 131 times; the paper of Rudin, Osher and Fatemi on total variation based denoising in image processing [159], cited 126 times; the paper of Osher and Chakravarthy on high resolution schemes and the entropy condition [139], cited 108 times, and the paper of Engquist and Osher on a monotone scheme (later referred to as the Engquist-Osher, or EO, scheme in the literature) [42], cited 107 times.

The organization of this paper is as follows. In section 2 we review the earlier work of Osher related to linear stability results. Section 3 is devoted to high resolution shock capturing methods for problems with discontinuous or otherwise nonsmooth solutions. Section 4 contains a review of the very popular level set methods, and finally in section 5 we address

PDE based methods in computer vision and image processing.

2 Linear stability results

The study of linear stability for finite difference and other numerical methods for hyperbolic, parabolic and other types of PDEs is very important. For linear methods approximating smooth solutions, a linear stability analysis (plus some dissipation) is usually enough to guarantee convergence, following the Lax equivalence theorem and Strang's result. Even for nonlinear methods and for methods approximating nonsmooth solutions, linear stability is often an important necessary condition for the algorithms to be useful.

For initial value problems, a von Neumann analysis (via Fourier transform) can be easily performed on a finite difference approximation as a necessary and often also sufficient condition for stability. However, stability for initial-boundary value problems is more difficult to analyze.

Osher's work on linear stability and linear methods was mainly done in the early dates. In [126], following up on a seminal paper of Kreiss [99], Osher used Toeplitz matrices in an elegant way to derive what was later called the GKS condition, i.e., the normal mode condition guaranteeing stability of approximations to initial-boundary value problems for linear hyperbolic equations. This line of work was initiated by Godunov and Ryabenkii [64]. It was made uniform by Kreiss [99], followed up by Osher [126], and generalized by Gustafsson, Kreiss and Sundstrom [70]. In [127] Osher provided more general conditions using similar Toeplitz matrix ideas.

In [128], Osher obtained stability conditions for initial-boundary value problems for parabolic equations, generalizing the work of Varah [190]. References [101, 129, 131] were an attempt to analyze and obtain conditions guaranteeing well posedness of initial boundary value problems for linear hyperbolic equations in regions with corners in the boundaries. Reference [130] showed that the Green's function for the biharmonic equation corresponding to a clamped plate near a right angle corner changes sign an infinite number of times.

In [112], Majda and Osher extended Kreiss' well posedness condition for initial-boundary value problems for hyperbolic equations to those with uniformly characteristic boundaries. In [111], Majda and Osher analyzed the reflection of singularities at the boundary for non-grazing reflection for hyperbolic equations. In [113], Majda and Osher showed how error propagates globally within the domain of dependence for numerical approximations to coupled hyperbolic systems. The paper [110] by Majda, McDonough and Osher was the first to recommend the use of smooth cutoff functions on the frequency domain for spectral methods to confine errors to local regions near propagating discontinuities and for stability. Sharp estimates on the region of propagation were obtained. These cutoffs are now widely used in the literature and the paper is still frequently cited, 45 times total, including many in recent years.

Osher in [132] obtained well-posedness results for linear boundary value problems of mixed elliptic-hyperbolic type; in [33], Deacon and Osher made the method into a finite element approximation for such equations.

In [44], Engquist, Osher and Zhong obtained wavelet based fast algorithms for linear hyperbolic and parabolic equations. Finally, in [41, 50, 49], Engquist, Fatemi and Osher considered numerical methods for high frequency asymptotics for geometric optics. These might be considered nonlinear, since the eikonal equation is.

3 High resolution shock capturing methods

Shock capturing methods refer to a class of numerical methods for solving problems containing discontinuities (shocks, contact discontinuities or other discontinuities), which can automatically "capture" these discontinuities without special effort to track them. A typical situation would be the solution of a hyperbolic conservation law, either a scalar equation or a system, either in one spatial dimension

$$u_t + f(u)_x = 0 \tag{3.1}$$

or in multiple (say, three) spatial dimensions:

$$u_t + f(u)_x + g(u)_y + h(u)_z = 0. \tag{3.2}$$

A main ingredient of shock capturing methods is the conservation form of a scheme, namely, a scheme approximating (3.1) is in the form

$$\frac{du_j}{dt} + \frac{1}{\Delta x}\left(\hat{f}_{j+\frac{1}{2}} - \hat{f}_{j-\frac{1}{2}}\right) = 0 \tag{3.3}$$

where u_j is an approximation to either the point value $u(x_j, t)$ or the cell average $\bar{u}(x_j, t) = \frac{1}{\Delta x}\int_{x_j-\frac{\Delta x}{2}}^{x_j+\frac{\Delta x}{2}} u(x,t)dx$ of the exact solution of (3.1), and $\hat{f}_{j+\frac{1}{2}}$ is a numerical flux which typically depends on a few neighboring points

$$\hat{f}_{j+\frac{1}{2}} = \hat{f}(u_{j-k}, u_{j-k+1}, ..., u_{j+m})$$

and satisfies the following two conditions: it is consistent with the physical flux $f(u)$ in the sense $\hat{f}(u, u, ..., u) = f(u)$, and it is at least Lipschitz continuous with respect to all its arguments. Notice that (3.3) is written in a semi-discrete method of lines form, while in practice the time variable t must also be discretized. Conservative schemes in the form of (3.3) are especially suitable for computing solutions with shocks, because of the important Lax-Wendroff theorem, which states that solutions to such schemes, if convergent, would converge to a weak solution of (3.1). In particular, this means that the computed shocks will propagate with the correct speed. Almost all shock capturing schemes, including those developed by Osher and his collaborators, are of the conservation form (3.3). However, there are certain situations where a relaxation on the strict conservation would be beneficial and would not hurt the convergence to weak solutions under suitable additional assumptions. The work of Osher and Chakravarthy [138] on the "weak conservation form" for schemes on general curvilinear coordinates, and the work of Fedkiw et al. on "ghost fluid" method [56], which treats the fluid interface in a non-conservative fashion, are such examples.

3.1 First Order Monotone Schemes

In the late 70s and early 80s, designing good first order monotone schemes for (3.1) and (3.2), which give monotone shock transitions and can be proven to converge to the physically relevant weak solutions (e.g. Crandall and Majda [32]), was an active research area. The Godunov scheme is a scheme with the least numerical dissipation among first order monotone schemes, however it is costly to evaluate for complex flux functions $f(u)$, and its flux is only Lipschitz continuous but not smoother. The Lax-Friedrichs scheme is easy to evaluate and very smooth but is excessive dissipative.

In [42] and [43], Engquist and Osher designed monotone schemes for the transonic potential equations and for general scalar conservation laws, which are relatively easy to evaluate, are C^1 smooth, and have a small dissipation almost comparable with Gudonov schemes. The main idea is to approximate everything by rarefaction waves (multi-valued solutions suitably integrated over for shocks). These Engquist-Osher schemes soon became very popular, especially for implicit type methods and steady state calculations, for which the extra smoothness of the numerical fluxes helped a lot. Similar schemes for Hamilton-Jacobi equations were given by Osher and Sethian [147].

Later, Osher [133] and Osher and Solomon [149] generalized it to systems of conservation laws, obtaining what was later referred to as Osher scheme in the literature. The Osher scheme for systems has a closed form formula (for Euler equations of gas dynamics and many other systems), hence no iterations are needed, unlike the Godunov scheme. It is smoother (C^1) than the Godunov scheme and also has smaller dissipation than the simpler Lax-Friedrichs scheme. Applications of Osher schemes to the Euler equations can be found in Chakravarthy and Osher [21].

In [145], Osher and Sanders designed a conservative procedure to handle locally varying time and space grids for first order monotone schemes, and proved convergence to entropy solutions for such schemes. These ideas have been used later by Berger and Colella on their adaptive methods.

3.2 High Resolution TVD Schemes

First order monotone schemes are certainly nice in their stability and convergence to the correct entropy solutions, however they are too diffusive for most applications. One would need to use many grid points to get a reasonable resolution, which seriously restricts their usefulness for multidimensional simulations.

In the 70s and early and mid 80s, the so-called "high resolution" schemes, i.e. those schemes which are at least second order accurate and are stable when shocks appear, were developed. These started with the earlier work of, e.g., the FCT methods of Boris and Book [10], and the MUSCL schemes of van Leer [189], and moved to Harten's TVD schemes [74]. Osher and his collaborators did extensive research on TVD schemes, and contributed significantly towards the analysis of such methods, during this period. These include the schemes developed and analyzed in [135], [139], [136], and the very high order (measured by truncation errors in smooth, monotone regions) TVD schemes in [140].

3.3 Entropy Conditions

The entropy condition is an important feature for conservation laws. Because weak solutions are not unique, entropy conditions are needed to single out a unique, physically relevant solution. Osher and his collaborators did extensive research on designing and analyzing entropy condition satisfying numerical methods for conservation laws.

In [114], Majda and Osher proved that the traditional second order Lax-Wendroff scheme, although linearly stable, is not L^2 stable when solving nonlinear conservation laws with discontinuous solutions. They then provided a recipe of adding artificial viscosities, such that the scheme maintained second order accuracy yet could be proven convergent to the entropy solution. This scheme is however oscillatory, hence not very practical in applications.

In [135], Osher provided a general framework to study systematically entropy conditions for numerical schemes. This was followed by the work of Osher and Chakravarthy [139] in the study of high resolution schemes and entropy conditions, the work of Osher [136] on

generalized MUSCL schemes, the work of Osher and Tadmor [150] on entropy condition and convergence of high resolution schemes, and the work of Brenier and Osher [11] on entropy condition satisfying "maxmod" second order schemes. Entropy condition satisfying approximations for the full potential equation of transonic flow were given in [142].

3.4 ENO Schemes

In the mid 80s it was realized that TVD schemes, despite their excellent stability and high resolution properties, have serious deficiency in that they degenerate to first order at *smooth* extrema of the solution [139]. Thus, even though TVD schemes can be designed to any order of accuracy, see for example the schemes up to 13th order accurate in [140], practical TVD schemes are referred to as second order schemes since the global L^1 errors of any TVD scheme can only be second order, even for smooth, non-monotone solutions.

In [75], Harten and Osher relaxed the TVD restriction, and replaced it by a UNO restriction, in that the total number of numerical extrema does not increase and their amplitudes could be allowed to increase slightly. The UNO scheme in [75] is uniformly second order accurate including at smooth extrema. However, it was soon realized that the UNO restriction was still too strong and excluded schemes of higher than second order. Thus, the concept of ENO, or essentially non-oscillatory, schemes was first given by Harten, Engquist, Osher and Chakravarthy [76] in 1987. The clever idea is that of an adaptive stencil, which is chosen based on the local smoothness of the solution, measured by the Newton divided differences of the numerical solution. Thus the order of scheme is never reduced, however the local stencil automatically avoids crossing discontinuities. Such schemes allow both the number of numerical extrema and their amplitudes to increase, however such additional oscillations are controlled on the level of truncation errors even if the solution is not smooth. ENO schemes have been extremely successful in applications, because they are simple in concept, allow arbitrary orders of accuracy, and generate sharp, monotone (to the eye) shock transitions together with high order accuracy in smooth regions of the solution including at the extrema.

The original ENO schemes in [76] are in the cell averaged form, namely they are finite volume schemes approximating an integrated version of (3.1). Finite volume schemes have the advantage of easy handling of non-uniform meshes and general geometry in multi-space dimensions, however they are extremely costly in multi-space dimensions, when the order of accuracy is higher than two, because then one cannot confuse cell averages with point values, as they only agree up to second order accuracy, and a complex reconstruction procedure is needed to obtain point values from cell averages for evaluating the numerical fluxes. The cost is also associated with the high order numerical quadratures needed for evaluating the integration of the numerical fluxes along cell boundaries in multi-dimensions. Later, Shu and Osher [171], [172] developed ENO schemes in finite difference using point values of the numerical solution, but still in conservation form (3.3). An important observation made in [171] and [172] is that the numerical flux $\hat{f}_{j+\frac{1}{2}}$ in (3.3) is *not* a high order approximation to the physical flux at $x_{j+\frac{1}{2}}$: the difference between the numerical flux $\hat{f}_{j+\frac{1}{2}}$ and the physical flux $f(u_{j+\frac{1}{2}})$ is $O(\Delta x^2)$. This is a common mistake among practitioners of finite difference schemes. If a high order interpolation on the point values u_j is performed to obtain a high order approximation to $u_{j+\frac{1}{2}}$, and a numerical flux is chosen to approximate $f(u_{j+\frac{1}{2}})$ to a high order accuracy, then the scheme is only second order accurate. Correct choice of the numerical fluxes to obtain arbitrarily high order accuracy is given in [171] and [172]. The approach in [172] is especially simple. A detailed description of the construction and comparison of finite volume and finite difference ENO schemes can be found in the lecture notes [170].

Also in [171], a class of nonlinearly stable high order Runge-Kutta time discretization methods is developed. Termed TVD time discretizations, these Runge-Kutta methods have become very popular and have been used in many schemes. See, e.g. [65] for a review of such methods.

Analysis of ENO schemes was given in Harten et al. [77]. Applications of ENO schemes to two and three dimensional compressible flows, including turbulence and shear flow calcu-

lations, were given in Shu et al. [173]. Triangle based second order non-oscillatory schemes were given in Durlofsky et al. [37]. Non-oscillatory self-similar maximum principle satisfying high order shock capturing schemes were given in Liu and Osher [106]. Efficient characteristic projection in upwind difference schemes was given in Fedkiw et al. [59]. Convex ENO schemes without using field-by-field projection were given in Liu and Osher [107]. Chemically reactive flows were simulated in Ton et al. [182] and in Fedkiw et al. [58].

The popularity of ENO schemes is demonstrated by the citation statistics: among Osher's five mostly highly cited papers mentioned in the introduction, four of them are about ENO schemes, i.e. [76] (cited 314 times); [171] (cited 235 times); [172] (cited 231 times); and [75] (cited 189 times). The top cited paper of Osher, [147] (cited 472 times) is on level set methods but also uses second order ENO schemes for the numerical solutions and is where the construction of ENO schemes for general Hamilton-Jacobi equations began.

3.5 WENO Schemes

An improvement of ENO scheme is the WENO (weighted ENO) scheme, which was first developed by Liu, Osher and Chan [108]. Both ENO and WENO use the idea of adaptive stencils in the reconstruction procedure based on the local smoothness of the numerical solution to automatically achieve high order accuracy and non-oscillatory property near discontinuities. ENO uses just one (optimal in some sense) out of many candidate stencils when doing the reconstruction; while WENO uses a convex combination of all the candidate stencils, each being assigned a nonlinear weight which depends on the local smoothness of the numerical solution based on that stencil. WENO improves upon ENO in robustness, better smoothness of fluxes, better steady state convergence, better provable convergence properties, and more efficiency.

WENO schemes have been further developed later by Jiang and Shu [88] for fifth order accurate finite difference schemes in one and several space dimensions, by Hu and Shu [80] and Shi et al. [168] for third and fourth order accurate finite volume schemes in two space

dimensions using arbitrary triangulations, and by Balsara and Shu [5] on very high order WENO schemes. A detailed description can again be found in the lecture notes [170].

3.6 Hamilton-Jacobi Equations

We will now move to the description of Osher's work in designing schemes for solving Hamilton-Jacobi equations. Further discussions on this topic will also be given in the next section on level set methods.

In [134], Osher gave explicit formulas for solutions to the Riemann problems for non-convex conservation laws and Hamilton-Jacobi equations. These are important for numerical schemes such as Godunov schemes using such Riemann solvers as building blocks.

In [147], Osher and Sethian, in the context of discussing level set methods, provided a first order monotone scheme (an adaptation of the Engquist-Osher scheme [43]) and a second order ENO scheme based on the framework of [171] and [172]. In [148], Osher and Shu developed high order ENO schemes for solving Hamilton-Jacobi equations, using various building blocks including Lax-Friedrichs, local Lax-Friedrichs, and Roe with an entropy fix. In [102], Lafon and Osher developed high order two dimensional triangle based non-oscillatory schemes for solving Hamilton-Jacobi equations. Later, Jiang and Peng [87] designed WENO schemes for solving Hamilton-Jacobi equations on rectangular meshes and Zhang and Shu [200] designed WENO schemes for solving Hamilton-Jacobi equations on arbitrary triangular meshes.

3.7 Additional Topics

Even though it does not exactly fit the title of this section, the work of Lagnado and Osher [103], [104] is worth mentioning. These papers concern solving an inverse problem to compute the volatility in the European options Black-Scholes model, and they were the first to use PDE techniques to solve this inverse problem, via gradient descent and Tychonoff regularization, allowing the volatility, a coefficient in a parabolic equation to be a function of the independent variables, stock price and time. These papers have attracted a lot of

attention after their publication.

Also worth mentioning is the work of Fatemi, Jerome and Osher [51] on using ENO schemes to solve the hydrodynamic models of semiconductor device simulations. This was the first work of using high order shock capturing methods in semiconductor device simulations, and has led to many further developments, e.g. [86] and [20].

4 Level set methods

4.1 Implicit Surfaces

In n dimensions, consider a surface that separates R^n into separate subdomains with nonzero volumes. For $n = 3$ an explicit representation can be quite difficult to discretize. One needs to choose a number of points on the two dimensional surface and record their connectivity. If the surface and its connectivity is known, it is simple to tile the surface with triangles whose vertices lie on the interface and edges indicate connectivity. On the other hand if connectivity is not known, it can be quite difficult to determine, and even some of the most popular algorithms can produce surprisingly inaccurate surface representations, e.g. surfaces with holes. Connectivity can change for dynamic implicit surfaces, i.e. pinching and merging. Here, connectivity is not a one time issue dealt with when constructing an explicit representation of the surface. Instead, it must be resolved over and over again every time pieces of the surface merge together or pinch apart. The "interface surgery" needed for merging and pinching is complex leading to a number of difficulties. One of the nicest properties of implicit surfaces is that connectivity does not need to be determined for the discretization. A uniform Cartesian grid can be used along with straightforward generalizations of the technology from two spatial dimensions. Possibly the most powerful aspect of implicit surfaces is that it is straightforward to go from two spatial dimensions to three (or even more) spatial dimensions.

Implicit surfaces are defined as the zero isocontour of a function $\phi(\vec{x})$. Consequently implicit interface representations include some powerful geometric tools. For example, we

can determine which side of the interface a point is on simply by looking at the local sign of ϕ. That is, $\vec{x}_o$ is inside the interface when $\phi(\vec{x}_o) < 0$, outside the interface when $\phi(\vec{x}_o) > 0$ and on the interface when $\phi(\vec{x}_o) = 0$.

Implicit functions make simple Boolean operations easy to apply. If ϕ_1 and ϕ_2 are two different implicit functions, then $\phi(\vec{x}) = \min(\phi_1(\vec{x}), \phi_2(\vec{x}))$ is the implicit function representing the union of their interior regions. Similarly, $\phi(\vec{x}) = \max(\phi_1(\vec{x}), \phi_2(\vec{x}))$ represents the intersection of the interior regions. The complement of $\phi_1(\vec{x})$ is $\phi(\vec{x}) = -\phi_1(\vec{x})$. Etc.

The gradient of the implicit function is defined as

$$\nabla\phi = \left(\frac{\partial\phi}{\partial x}, \frac{\partial\phi}{\partial y}, \frac{\partial\phi}{\partial z}\right). \tag{4.1}$$

$\nabla\phi$ is perpendicular to the isocontours of ϕ pointing in the direction of increasing ϕ. Therefore, if $\vec{x}_o$ is a point on the zero isocontour of ϕ, i.e. a point on the interface, then $\nabla\phi$ evaluated at $\vec{x}_o$ is a vector that points in same direction as the local unit (outward) normal $\vec{N}$ to the interface. Thus, the unit (outward) normal is

$$\vec{N} = \frac{\nabla\phi}{|\nabla\phi|} \tag{4.2}$$

for points on the interface. Equation (4.2) can be used to define a function $\vec{N}$ everywhere on the domain embedding the normal in a function $\vec{N}$ that agrees with the normal for points on the interface. The mean curvature of the interface is defined as the divergence of the normal,

$$\kappa = \nabla \cdot \vec{N} \tag{4.3}$$

so that $\kappa > 0$ for convex regions, $\kappa < 0$ for concave regions and $\kappa = 0$ for a plane.

The characteristic function χ^- of the interior region Ω^- is defined as

$$\chi^-(\vec{x}) = \begin{cases} 1 & \text{if} \quad \phi(\vec{x}) \leq 0 \\ 0 & \text{if} \quad \phi(\vec{x}) > 0 \end{cases} \tag{4.4}$$

where we arbitrarily include the boundary with the interior region. The characteristic function, χ^+ of the exterior region Ω^+ is defined similarly as

$$\chi^+(\vec{x}) = \begin{cases} 0 & \text{if} \quad \phi(\vec{x}) \leq 0 \\ 1 & \text{if} \quad \phi(\vec{x}) > 0 \end{cases} \tag{4.5}$$

again including the boundary with the interior region. $\chi^{\pm}$ are functions of a multidimensional variable $\vec{x}$. It is often more convenient to work with functions of the scalar variable ϕ. Thus we define the one dimensional Heaviside function

$$H(\phi) = \begin{cases} 0 & \text{if} \quad \phi \leq 0 \\ 1 & \text{if} \quad \phi > 0 \end{cases} \tag{4.6}$$

where ϕ depends on $\vec{x}$, although it is not necessary to specify this dependence when working with H. Note that $\chi^+(\vec{x}) = H(\phi(\vec{x}))$ and $\chi^-(\vec{x}) = 1 - H(\phi(\vec{x}))$.

The volume integral (area integral in R^2) of a function f over the interior region Ω^- is defined as

$$\int_{\Omega} f(\vec{x})\chi^-(\vec{x})d\vec{x} \tag{4.7}$$

where the region of integration is all of Ω since χ^- prunes out the exterior region Ω^+ automatically. The one dimensional Heaviside function can be used to rewrite this volume integral as

$$\int_{\Omega} f(\vec{x})\left(1 - H\left(\phi(\vec{x})\right)\right) d\vec{x} \tag{4.8}$$

representing the integral of f over the interior region Ω^-. Similarly,

$$\int_{\Omega} f(\vec{x})H\left(\phi(\vec{x})\right)d\vec{x} \tag{4.9}$$

is the integral of f over the exterior region Ω^+.

By definition, the directional derivative of the Heaviside function H in the normal direction $\vec{N}$ is the Dirac delta function

$$\hat{\delta}(\vec{x}) = \nabla H\left(\phi(\vec{x})\right) \cdot \vec{N} \tag{4.10}$$

which is a function of the multidimensional variable $\vec{x}$. Note that this distribution is only nonzero on the interface $\partial\Omega$ where $\phi = 0$. We can rewrite equation (4.10) as

$$\hat{\delta}(\vec{x}) = H'(\phi(\vec{x}))\nabla\phi(\vec{x}) \cdot \frac{\nabla\phi(\vec{x})}{|\nabla\phi(\vec{x})|} = H'(\phi(\vec{x}))|\nabla\phi(\vec{x})| \tag{4.11}$$

using the chain rule to take the gradient of H and the definition of the normal from equation (4.2). In one spatial dimension, the delta function is defined as the derivative of the Heaviside

function

$$\delta(\phi) = H'(\phi) \tag{4.12}$$

with $H(\phi)$ defined in equation (4.6) above. $\delta(\phi)$ is identically zero everywhere except where $\phi = 0$. This allows us to rewrite equation (4.11) as

$$\hat{\delta}(\vec{x}) = \delta(\phi(\vec{x}))|\nabla\phi(\vec{x})| \tag{4.13}$$

using the one dimensional delta function $\delta(\phi)$.

The surface integral (line integral in R^2) of a function f over the boundary $\partial\Omega$ is defined as

$$\int_\Omega f(\vec{x})\hat{\delta}(\vec{x})d\vec{x} \tag{4.14}$$

where the region of integration is all of Ω since $\hat{\delta}$ prunes out everything except $\partial\Omega$ automatically. The one dimensional delta function can be used to rewrite this surface integral as

$$\int_\Omega f(\vec{x})\delta(\phi(\vec{x}))|\nabla\phi(\vec{x})|d\vec{x}. \tag{4.15}$$

Typically, volume integrals are computed by dividing up the interior region, and surface integrals are computed by dividing up the boundary $\partial\Omega$. This requires treating a complex two dimensional surface in three spatial dimensions. By embedding the volume and surface integrals in higher dimensions, equations (4.8), (4.9) and (4.15) avoid the need for identifying inside, outside or boundary regions. Instead the integrals are taken over the entire region Ω.

Consider the surface integral in equation (4.15) where the one dimensional delta function needs to be evaluated. Since $\delta(\phi) = 0$ almost everywhere, i.e. except on the lower dimensional interface which has measure zero, it seems unlikely that any standard numerical approximation based on sampling will give a good approximation to this integral. Thus, we use a first order accurate smeared out approximation of $\delta(\phi)$. First, we define the smeared out Heaviside function

$$H(\phi) = \begin{cases} 0 & \phi < -\epsilon \\ \frac{1}{2} + \frac{\phi}{2\epsilon} + \frac{1}{2\pi}\sin\left(\frac{\pi\phi}{\epsilon}\right) & -\epsilon \le \phi \le \epsilon \\ 1 & \epsilon < \phi \end{cases} \tag{4.16}$$

where ϵ is a tunable parameter that determines the size of the bandwidth of numerical smearing. A typically good value is $\epsilon = 1.5\triangle x$ making the interface width equal to three grid cells when ϕ is normalized to a signed distance function with $|\nabla\phi| = 1$. Then the delta function is defined according to equation (4.12) as the derivative of the Heaviside function

$$\delta(\phi) = \begin{cases} 0 & \phi < -\epsilon \\ \frac{1}{2\epsilon} + \frac{1}{2\epsilon}\cos\left(\frac{\pi\phi}{\epsilon}\right) & -\epsilon \leq \phi \leq \epsilon \\ 0 & \epsilon < \phi \end{cases} \tag{4.17}$$

where ϵ is determined as above. This delta function allows us to evaluate the surface integral in equation (4.15) using a standard sampling technique such as the midpoint rule. Similarly, the smeared out Heaviside function in equation (4.16) aids in the evaluation of the integrals in equations (4.8) and (4.9).

A distance function $d(\vec{x})$ is defined as

$$d(\vec{x}) = \min|\vec{x} - \vec{x}_I| \quad \text{over all} \quad \vec{x}_I \in \partial\Omega \tag{4.18}$$

implying that $d(\vec{x}) = 0$ on the boundary where $\vec{x} \in \partial\Omega$. For a given point $\vec{x}$, suppose that $\vec{x}_C$ is the point on the interface closest to $\vec{x}$. The line segment from $\vec{x}$ to $\vec{x}_C$ is the shortest path from $\vec{x}$ to the interface. In other words, the path from $\vec{x}$ to $\vec{x}_C$ is the path of steepest descent for the function d. Evaluating $-\nabla d$ at any point on the line segment from $\vec{x}$ to $\vec{x}_C$ gives a vector that points from $\vec{x}$ to $\vec{x}_C$. Furthermore, since d is Euclidean distance,

$$|\nabla d| = 1. \tag{4.19}$$

A signed distance function is an implicit function ϕ with $\phi(\vec{x}) = d(\vec{x}) = 0$ for all $\vec{x} \in \partial\Omega$, $\phi(\vec{x}) = -d(\vec{x})$ for all $\vec{x} \in \Omega^-$, and $\phi(\vec{x}) = d(\vec{x})$ for all $\vec{x} \in \Omega^+$. Given a point $\vec{x}$, and using the fact that $\phi(\vec{x})$ is the signed distance to the closest point on the interface, we can write

$$\vec{x}_C = \vec{x} - \phi(\vec{x})\vec{N} \tag{4.20}$$

to calculate the closest point on the interface where $\vec{N}$ is the local unit normal at $\vec{x}$.

4.2 Level Set Methods

Level set methods add dynamics to implicit surfaces. The key idea that started the level set fanfare was the Hamilton-Jacobi approach to numerical solutions of a time dependent equation for a moving implicit surface. This was first done in the seminal work of Osher and Sethian [147].

Suppose that the velocity of each point on the implicit surface is given as $\vec{V}(\vec{x})$. Given this velocity field, $\vec{V} = \langle u, v, w \rangle$, we wish to move all the points on the surface with this velocity. The simplest way to do this is to solve the ordinary differential equation

$$\frac{d\vec{x}}{dt} = \vec{V}(\vec{x}) \tag{4.21}$$

for every point $\vec{x}$ on the front, i.e. for all $\vec{x}$ with $\phi(\vec{x}) = 0$. This is the Lagrangian formulation of the interface evolution equation. Since there is generally an infinite number of points on the front, this means discretizing the front into a finite number of pieces. For example, one could use segments in two spatial dimensions or triangles in three spatial dimensions. This is not so hard to accomplish if the connectivity does not change and the surface elements do not distort too much. Unfortunately, even the most trivial velocity fields can cause large distortion of boundary elements and the accuracy of the method can deteriorate quickly if one does not periodically modify the discretization in order to account for these deformations by smoothing and regularizing inaccurate surface elements.

In order to avoid problems with instabilities, deformation of surface elements and complicated surgical procedures for topological repair of interfaces, Osher and Sethian [147] proposed using the implicit function ϕ both to represent the interface and to evolve the interface. The evolution of the implicit function ϕ is governed by the simple convection equation

$$\phi_t + \vec{V} \cdot \nabla\phi = 0. \tag{4.22}$$

This is an Eulerian formulation of the interface evolution since the interface is captured by the implicit function ϕ as opposed to being tracked by interface elements as is done in a

Lagrangian formulation. Equation (4.22) is sometimes referred to as the level set equation. The velocity field given in equation (4.22) can come from a number of external sources. For example, when the $\phi(\vec{x}) = 0$ isocontour represents the interface between two different fluids, the interface velocity is calculated using the two-phase Navier-Stokes equations.

In general, one does not need to specify tangential components when devising a velocity field. Since $\vec{N}$ and $\nabla\phi$ point in the same direction, $\vec{T} \cdot \nabla\phi = 0$ for any tangent vector $\vec{T}$ implying that the tangential velocity components vanish when plugged into the level set equation. For example, in two spatial dimensions with $\vec{V} = V_n\vec{N} + V_t\vec{T}$, the level set equation

$$\phi_t + \left(V_n\vec{N} + V_t\vec{T}\right) \cdot \nabla\phi = 0 \tag{4.23}$$

is equivalent to

$$\phi_t + V_n\vec{N} \cdot \nabla\phi = 0. \tag{4.24}$$

Furthermore, since

$$\vec{N} \cdot \nabla\phi = \frac{\nabla\phi}{|\nabla\phi|} \cdot \nabla\phi = \frac{|\nabla\phi|^2}{|\nabla\phi|} = |\nabla\phi| \tag{4.25}$$

we can rewrite equation (4.24) as

$$\phi_t + V_n|\nabla\phi| = 0 \tag{4.26}$$

where V_n is the component of velocity in the normal direction (the normal velocity). Equation (4.26) is *also* known as the level set equation. Equation (4.22) tends to be used for externally generated velocity fields while equation (4.26) tends to be used for (internally) self-generated velocity fields.

4.3 Numerical Techniques

This subsection is a natural continuation of the discussion of numerical methods in section 3.

Consider the one dimensional scalar conservation law

$$u_t + f(u)_x = 0 \tag{4.27}$$

where u is the conserved quantity and $f(u)$ is the flux function. A well known system of conservation laws are the Euler equations for inviscid fluid flow dynamics. The Euler equations are rather interesting because the presence of discontinuities forces one to consider weak solutions where the derivatives of solution variables can fail to exist. While a contact discontinuity is essentially linear, the nonlinear nature of a shock wave discontinuity allows it to develop as the solution progresses forward in time even if the data is initially smooth. Another interesting aspect of the Euler equations concerns the uniqueness of the solution. When more than one solution exists, an entropy condition is needed to pick out the physically correct solution. It turns out that the vanishing viscosity solution is the desired physically correct solution. For example, this vanishing viscosity solution admits a physically consistent rarefaction wave as opposed to a physically inadmissible expansion shock.

Now consider the one dimensional Hamilton-Jacobi equation

$$\phi_t + H(\phi_x) = 0 \tag{4.28}$$

which becomes

$$(\phi_x)_t + H(\phi_x)_x = 0 \tag{4.29}$$

after taking a spatial derivative of the entire equation. Setting $u = \phi_x$ in equation (4.28) results in

$$u_t + H(u)_x = 0 \tag{4.30}$$

which is a scalar conservation law. Thus in one spatial dimension, we can draw a direct correspondence between Hamilton-Jacobi equations and conservation laws. The solution u to conservation law is the derivative of a solution ϕ to a Hamilton-Jacobi equation. Conversely, the solution ϕ to a Hamilton-Jacobi equation is the integral of a solution u to a conservation law. This allows us to point out a number of useful facts. For example, since the integral of a discontinuity is a kink (discontinuity in first derivative), solutions to Hamilton-Jacobi equations can develop kinks in the solution even if the data is initially smooth. In addition, solutions to Hamilton-Jacobi equations cannot generally develop a discontinuity (unless the

corresponding conservation law solution develops a delta function). Thus, solutions ϕ to equation (4.28) are typically continuous. Furthermore, since conservation laws can have nonunique solutions, one needs to apply an entropy condition to pick out the "physically" relevant solution to equation (4.28).

Viscosity solutions for Hamilton-Jacobi equations were first proposed by Crandall and Lions [30] in order to pick out the physically relevant solution. In addition, monotone first order accurate numerical methods were first proven to converge by Crandall and Lions in [31]. Later, in [147], Osher and Sethian used the connection between conservation laws and Hamilton-Jacobi equations to construct higher order accurate artifact free numerical methods based in part on new upwind difference schemes. Even though the analogy between conservation laws and Hamilton-Jacobi equations fails in multidimensions, many Hamilton-Jacobi equations can be discretized in a dimension by dimension fashion. This cumulated in [148] where Osher and Shu proposed a general framework for the numerical solution of Hamilton-Jacobi equations using modern methods from the theory of conservation laws and the multidimensional Riemann solver of Bardi and Osher [6]. The framework in [148] allowed one to use Lax-Friedrichs, Roe-Fix or Godunov building blocks to create higher order accurate spatial discretizations using an essentially non-oscillatory (ENO) polynomial reconstruction introduced in [76] by Harten et al. for the numerical solution of conservation laws. The basic idea is to compute numerical flux functions using the smoothest possible polynomial interpolants. The actual numerical implementation of this idea was improved considerably by Shu and Osher in [171] and [172] where the numerical flux functions were constructed directly from a divided difference table of the pointwise data. [171] and [172] addressed higher order accurate Runge-Kutta discretization in time as well.

In [108], Liu et al. pointed out that the ENO philosophy of picking out exactly one of three candidate stencils is overkill in smooth regions where the data is well behaved. They proposed a Weighted ENO (WENO) method that takes a convex combination of the three ENO approximations. Of course, if any of the three approximations interpolate across a

discontinuity, it is given minimal weight in the convex combination in order to minimize its contribution and the resulting errors. Otherwise, in smooth regions of the flow, all three approximations are allowed to make a significant contribution in a way that improves the local accuracy from third order to fourth order accuracy. Later, Jiang and Shu [88] improved the WENO method by choosing the convex combination weights in order to obtain the optimal fifth order accuracy in smooth regions of the flow. In [87], following the work on HJ ENO in [148], Jiang and Peng extended WENO to the Hamilton-Jacobi framework. This Hamilton-Jacobi WENO or HJ WENO scheme turns out to be very useful as it reduces the numerical errors by more than an order of magnitude over the third order accurate HJ ENO scheme for typical applications.

Even with these high order accurate approaches to solving the Hamilton-Jacobi equations, one can obtain surprisingly inaccurate results when the level set function solution becomes too steep or too flat, i.e. discontinuous or poorly conditioned. In [29], Chopp considered an application where certain regions of the flow had level sets piling up on each other increasing the local gradient, and other regions of the flow had level sets that separated from each other flattening out ϕ. In order to reduce the numerical errors caused by both the steeping and flattening effects, [29] introduced the notion that one should reinitialize the level set function periodically throughout the calculation. In [158], Rouy and Tourin proposed a numerical method for the shape from shading problem that was later generalized into the modern day reinitialization equation of Sussman, Smereka and Osher [177], using the fact that $|\nabla d| = 1$, for d the signed or unsigned distance to a given set.

Unfortunately, this straightforward reinitialization routine can be slow, especially if it needs to be done every time step although [177] noted that just a few time iterations are usually needed. In order to obtain reasonable run times, [29] restricted the calculations of the interface motion and the reinitialization to a small band of points near the $\phi = 0$ isocontour. This idea of computing solutions to Hamilton-Jacobi equations local to the interface has been studied further in the more recent work of Adalsteinsson and Sethian [1] and Peng et

al. [153].

Local methods are important for both solving the Hamilton-Jacobi equation and for reinitializing the level sets so that they do not become discontinuous or poorly conditioned. However, at least in the reinitialization case, it is possible to construct an even faster method that only treats each grid point once while sweeping out from the zero isocontour creating a signed distance function. This algorithm was invented by Tsitsiklis in a pair of papers, [185] and [186]. The most novel part of this algorithm is the extension of Dijkstra's algorithm for computing the taxicab metric to an algorithm for computing Euclidean distance. Although this method was originally proposed by Tsitsiklis, it was later rediscovered by the level set community, see for example Sethian [166] and Helmsen et al. [78].

The great success of level set methods can in part be attributed to the role of curvature in regularizing the level set function such that the proper vanishing viscosity solution is obtained. It is much more difficult to obtain vanishing viscosity solutions with Lagrangian methods that faithfully follow the characteristics. For these methods, one usually has to delete (or add) characteristic information by hand when a shock (or rarefaction) is detected. This ability of level set methods to identify and delete merging characteristics is clearly seen in a purely geometrically driven flow where a square is advected inward normal to itself at constant speed. In the corners of the square, the flow field has merging characteristics that are appropriately deleted by the level set method. On the other hand, repeating the same calculation with a Lagrangian numerical method is difficult since characteristics will merge in the corners of the square but not be automatically deleted. One does not easily obtain the correct viscosity solution. Level set methods are not perfect however, since they tend to incorrectly delete characteristics in under resolved regions of the flow – a behavior frequently called "loss of mass" (or volume) in reference to the error it represents when level sets are used to model incompressible fluid flow. In contrast, despite a lack of explicit enforcement of mass (or volume) conservation, Lagrangian schemes are quite successful in conserving mass since they preserve material characteristics for all time, i.e. characteristics are never deleted.

The difficulty stems from the fact that the level set method cannot accurately tell if characteristics merge, separate, or run parallel in under-resolved regions of the flow. This indeterminacy leads to vanishing viscosity solutions that can incorrectly delete characteristics when they appear to be merging. In [46], Enright et al. designed a hybrid particle level set method to alleviate the mass loss issues associated with level set methods. In the case of fluid flows, knowing a priori that there are no shocks present in the fluid velocity field, one can assert that characteristic information associated with that characteristic field should never be deleted. Particles are randomly seeded near the interface and passively advected with the flow. When marker particles cross over the interface, it indicates that characteristic information has been incorrectly deleted, and these errors are fixed by locally rebuilding the level set function using the characteristic information present in these escaped marker particles.

4.4 Fluids and Materials

Chronologically, the first attempt to use the level set method for flows involving external physics was in the area of two phase inviscid compressible flow. Mulder et al. [122] appended the level set equation to the standard equations for one phase compressible flow. The level set was advected using the velocity of the compressible flow field so that the zero level set of ϕ corresponds to particle velocities and can be used to track an interface separating two different compressible fluids. Later, Karni [90] pointed out that such method suffered from spurious oscillations at the interface. This was later fixed by Fedkiw et al. [56] by creating a set of fictitious ghost cells on each side of the interface, and populating these ghost cells with a specially chosen ghost fluid that implicitly captures the Rankine-Hugoniot jump conditions across the interface. This method was referred to as the ghost fluid method. Later, Fedkiw et al. [57] extended the level set method and ghost fluid method to treat shock, detonation and deflagration waves as sharp discontinuities. Caiden et al. [15] extended these methods to couple incompressible flows to compressible flows in order to study liquid/gas interac-

tions. Fedkiw [55] extended these methods to couple Lagrangian calculations to Eulerian calculations in order to study solid/fluid interactions.

The earliest real success in the coupling of the level set method to problems involving external physics came in computing two-phase incompressible flow, in particular see Sussman et al. [177] and Chang et al. [24]. The Navier-Stokes equations were used to model the fluids on both sides of the interface. Generally, the fluids will have different densities and viscosities and the presence of surface tension forces cause the pressure to be discontinuous across the interface as well. Although these early papers smeared out these discontinuous quantities across the interface, this was later remedied by Kang et al. [89] using the methods developed by Liu et al. [109]. More recently, Nguyen et al. [124] extended these techniques to treat low speed flames.

A level set regularization procedure was proposed in Harabetian and Osher [72] for ill-posed problems such as vortex motion in incompressible flows. This regularization, coupled with non-oscillatory numerical methods for the resulting level set equations, provides a regularization which is topological and is automatically accomplished through the use of numerical schemes whose viscosity shrinks to zero with grid size. There is no need for explicit filtering, even when singularities appear in the solution. The method also has the advantage of automatically allowing topological changes such as merging of surfaces.

An application of this procedure for incompressible vortex motion was given in Harabetian, Osher and Shu [73]. An Eulerian, fixed grid, approach to solve the motion of an incompressible fluid, in two and three dimensions, in which the vorticity is concentrated on a lower dimensional set, is provided. The numerical variables for the level sets are actually smooth, thus allowing for accurate numerical simulations. Numerical examples including two and three dimensional vortex sheets, two dimensional vortex dipole sheets and point vortices, are given. This was the first three dimensional vortex sheet calculation in which the sheet evolution feeds back to the calculation of the fluid velocity, although vortex in cell calculations for three dimensional vortex sheets were done earlier by Trygvasson et al. in

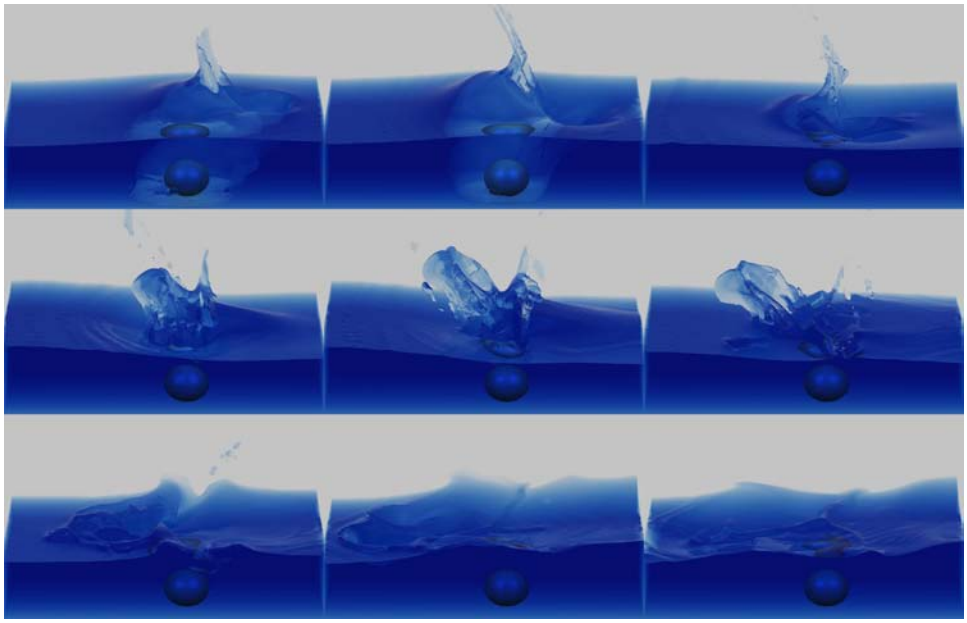


Figure 4.1: A splash is generated as a sphere is thrown into the water.

[183].

Level set methods have been applied to a variety of other problems as well. They have been used to compute solutions to Stefan problems to study crystal growth [26, 94], to simulate water for computer graphics applications [60] as shown in Figures 4.1 and 4.2, and to reconstruct three dimensional models from arbitrary unorganized data points [202] as shown in Figures 4.3 (before) and 4.4 (after).

Level set type analysis was also used to obtain rigorous results identifying the Wulff minimizing shape and the evolution of growing crystals moving with normal velocity defined as a given positive function of the normal direction, thus verifying a conjecture of Gross. Moreover it was also shown that the Wulff energy decreases monotonically under such an evolution to its minimum [143]. A spinoff came in [152] where it was proven that any two dimensional Wulff shape can be interpreted as the solution a corresponding Riemann problem for a scalar conservation law – jumps in the direction of the normal correspond to contact discontinuities, smoothly varying thin flat faces correspond to rarefaction curves and planar facets correspond to constant states. The work in [143] also motivated the derivation of a

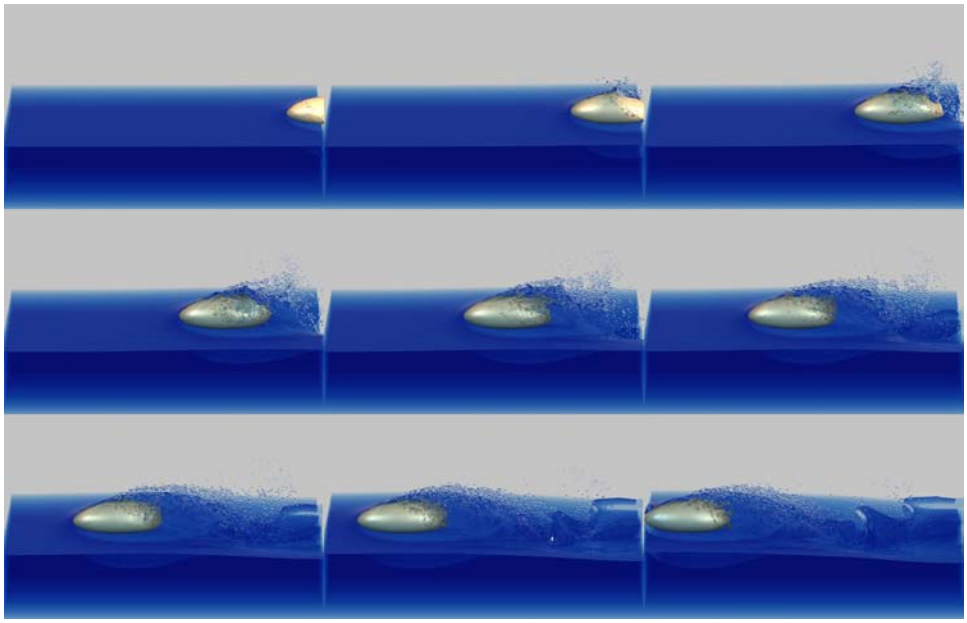


Figure 4.2: An interesting spray effect is generated as a slightly submerged ellipse slips through the water.

new class of isoperimetric inequalities for convex plane curves [67].

Molecular beam epitaxy (MBE) is a method for growing extremely thin films of material. A new continuum model for the epitaxial growth of thin films has been developed. This new island dynamics model has been designed to capture the larger length scale features. The key idea involves the level set based motion of islands of various integer levels – see for example [121, 25, 71].

4.5 A Variational Approach

In [201] a variational level set approach was developed. Key ideas were the use of a single level set function for each phase, the gradient projection method of [159] to prevent overlap and / or vacuum, and the liberal use of the level set calculus as described earlier. This general variational approach has many applications. The first was to study the behavior of bubbles and droplets in two and three dimensions [203], for example drops falling or remaining attached to a generally irregular ceiling, and mercury sitting on the floor.

Many problems in engineering design involve optimizing the geometry to maximize a

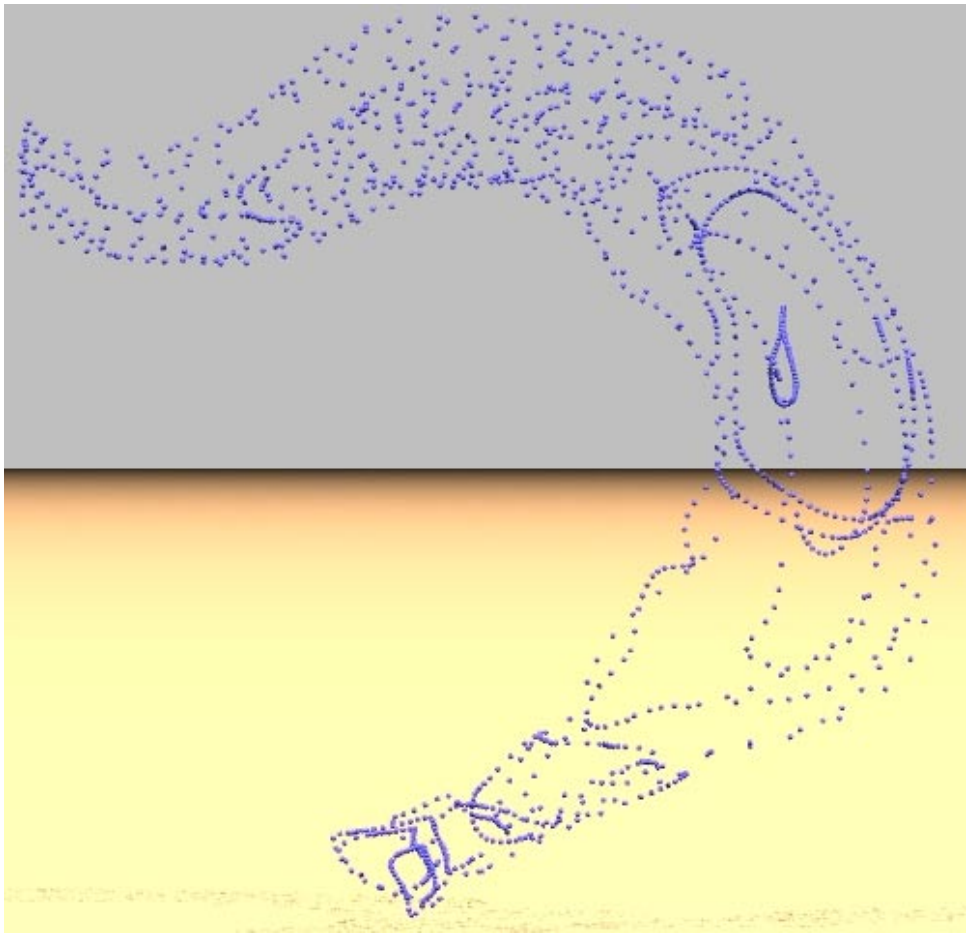


Figure 4.3: Arbitrary data points measured from a rat brain.

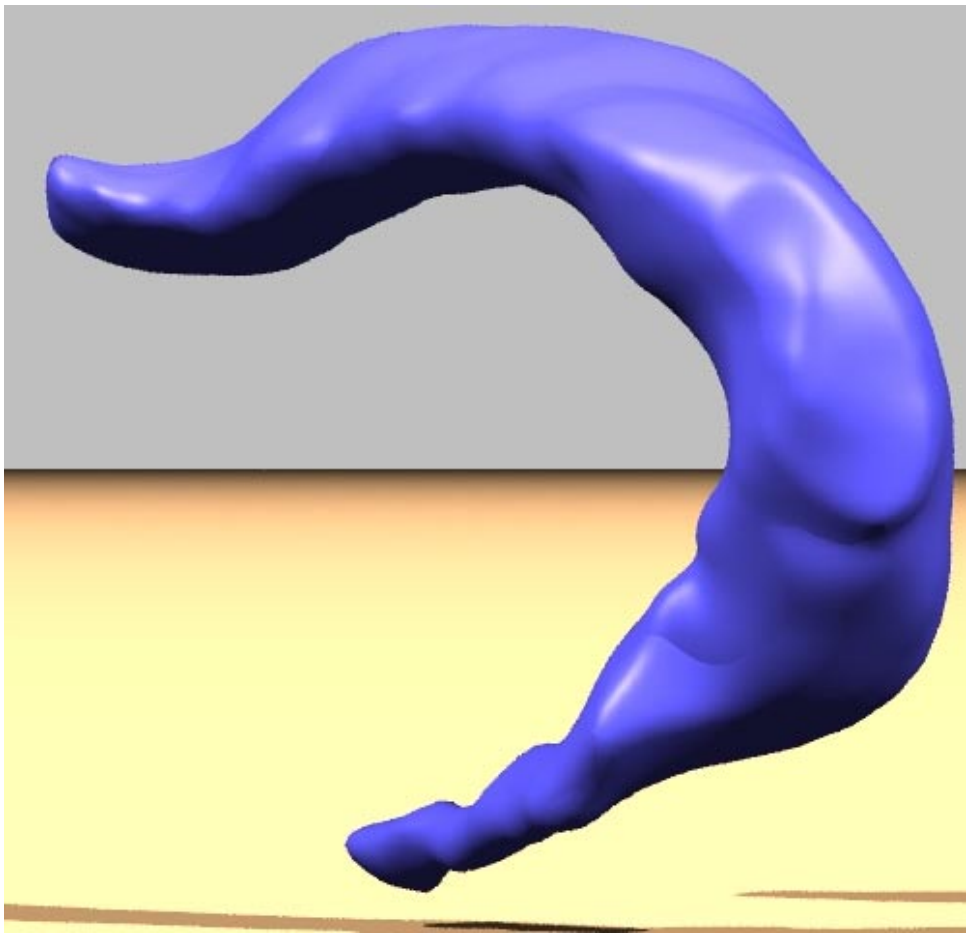


Figure 4.4: A three dimensional geometric reconstruction of the rat brain using the level set method.

certain design objective. In [146] the variational level set method was used to analyze a vibrating system whose resonant frequency or whose spectral gap is to be optimized subject to constraints on the geometry.

This variational approach has applications in computer vision as well, e.g. snakes and active contours [22]. This will be discussed further in section 5.

4.6 High Codimension Motion

Typically level set methods are used to model codimension one objects, e.g. curves in R^2 or surfaces in R^3. In [13], this technology was extended to treat codimension two objects, e.g. curves in R^3, using the intersection of the zero level sets of two functions. This means a curve is determined by

$$\Gamma(t) = \{\vec{x} | \phi_1(\vec{x}, t) = \phi_2(\vec{x}, t) = 0\}.$$

The geometry of the curve can be derived from ϕ_1 and ϕ_2. For example, the tangent to the curve is defined by

$$\vec{T} = \frac{\nabla\phi_1 \times \nabla\phi_2}{|\nabla\phi_1 \times \nabla\phi_2|}.$$

The curvature times the normal is the derivative of the tangent vector along the curve,

$$\kappa\vec{N} = \nabla\vec{T} \cdot \vec{T}. \tag{4.31}$$

The normal vectors can be defined by normalizing this quantity,

$$\vec{N} = \frac{\kappa\vec{N}}{|\kappa\vec{N}|}. \tag{4.32}$$

The binormal is

$$\vec{B} = \frac{\vec{T} \times \vec{N}}{|\vec{T} \times \vec{N}|}.$$

The torsion times the normal vector is $\tau\vec{N} = -\nabla\vec{B} \cdot \vec{T}$.

These geometric quantities are all defined numerically just as in the standard codimension one level set method. Geometric motion of a curve in R^3 is thus obtained by solving coupled

systems of two evolution equations. This is done locally near $\Gamma(t)$, saving on storage and complexity. See [13] for results involving merging and breaking which appear to agree with the reaction-diffusion limit when appropriate.

Another application of this idea comes from the following observation. If we freeze one of the functions, say ϕ_1, we can generate the motion of curves on a surface. Here the surface is defined by $\{\vec{x}|\phi_1(\vec{x}) = 0\}$ and the evolving curve is defined by the intersection of that fixed surface with $\{\vec{x}|\phi_2(\vec{x}, t) = 0\}$. This is useful for path planning on terrain data, see [28].

4.7 Geometric Optics

In [141] a level set based approach for ray tracing and for the construction of wavefronts in geometric optics was introduced. The approach automatically handles the multivalued solutions that appear and automatically resolves the wavefronts. The key idea, first introduced in [45] in a "segment projection" (rather than a level set) approach, is to use the linear Liouville equation in twice as many independent variables and solve in this higher dimensional space via the idea introduced in [13].

In two dimensional ray tracing, this involves solving for an evolving curve in x, y, θ space, where θ is the angle of the normal to the curve. This uses two level set functions and gives codimension 2 motion in 3 space dimension plus time. A local level set method can be used to make the complexity tractable – $O(n^2 \log(n))$ – for n the number of points on the curve for every time iteration. The memory requirement is $O(n^2)$.

In three dimensional ray tracing, this involves solving for an evolving two dimensional surface in x, y, z, θ, ψ space, where θ and ψ give the angle of the normal, and results in codimension 3 motion in 5 space dimension plus time. The complexity goes up by a power of n over the two dimensional case, as does the memory requirement. Again, this involves a local level set method, this time using three level set functions. The interested reader is referred to [175] and [162] for a different Eulerian approach.

4.8 Computing Discontinuous Solutions to Hamilton-Jacobi Equations

Hamilton-Jacobi equations of the form

$$\phi_t + H(\vec{x}, t, \phi, \nabla\phi) = 0 \tag{4.33}$$

have uniformly continuous solutions if H is non-decreasing in ϕ. However, there are interesting cases in which this hypothesis fails. Moreover, discontinuous initial data is appropriate for some problems in control theory and differential games. The solution devised in [63] uses the evolution of the level set of an auxiliary level set equation. The idea has antecedents in [137] where it was proven that, under reasonable circumstances, the zero level set of the viscosity solution of

$$\phi_t + H(\vec{x}, \nabla\phi) = 0$$

for H homogeneous of degree one in $\nabla\phi$ is the same as the t level set of the viscosity solution of

$$H(\vec{x}, \nabla\psi) = 1$$

i.e.

$$\{\vec{x}|\phi(\vec{x}, t) = 0\} = \{\vec{x}|\psi(\vec{x}) = t\}. \tag{4.34}$$

This idea was used in [63] to go one dimension higher in equation (4.33). This leads to new and successful numerical methods for a wide class of initial value problems for Hamilton-Jacobi equations with discontinuous solutions, see [184].

5 Image processing and computer vision

The use of partial differential equations (PDE's) and curvature driven flows in image processing and computer vision has become an active research topic in the past few years. The basic idea is to deform a given curve, surface, or image with a PDE, and obtain the desired result as the solution of this PDE. Sometimes, as in the case of color images, a system of coupled

PDE's is used. The art behind this technique is in the design, analysis, and implementation of these PDE's.

Partial differential equations can be obtained from variational problems. Assume a variational approach to an image processing problem formulated as

$$\arg\left\{\mathrm{Min}_u \mathcal{U}(u)\right\},$$

where $\mathcal{U}$ is a given energy computed over the image (or surface) u. Let $\mathcal{F}(\cdot)$ denote the Euler derivative (first variation) of $\mathcal{U}$. Since under general assumptions, a necessary condition for u to be a minimizer of $\mathcal{U}$ is that $\mathcal{F}(u) = 0$, the (local) minima may be computed via the steady state solution of the equation

$$\frac{\partial u}{\partial t} = -\mathcal{F}(u),$$

where t is an 'artificial' time marching parameter. PDE's obtained in this way have been used already for quite some time in computer vision and image processing, and the literature is large. The most classical example is the Dirichlet integral,

$$\mathcal{U}(u) = \int |\nabla u|^2(x)dx,$$

which is associated with the linear heat equation

$$\frac{\partial u}{\partial t}(x,t) = \Delta u(x).$$

Extensive research is also being done on the direct derivation of evolution equations which are not necessarily obtained from the energy approaches. The attributes of PDE's in image processing are discussed for example in [19, 163]. In the pioneering paper [2] the authors prove that a few basic image processing principles naturally lead to PDE's.

Note that when considering PDE's for image processing and numerical implementations, we are dealing with derivatives of non-smooth signals, and the right framework must be defined. As introduced by the image processing group formerly at CEREMADE [2, 3], the theory of *viscosity solutions* provides a framework for rigorously employing a partial

differential formalism, in spite of the fact that the image may not be smooth enough to give a classical sense to derivatives involved in the PDE. These works also showed with a very elegant axiomatic approach the importance of PDE's in image processing.

Ideas on the use of PDE's in image processing go back at least to Gabor [62] and to Jain [85]. The field took off thanks to the independent works of Koenderink [98] and Witkin [195]. These researchers rigorously introduced the notion of *scale-space*, that is, the representation of images simultaneously at multiple scales. In their work, the multi-scale image representation is obtained by Gaussian filtering, see below. This is equivalent to deforming the original image via the classical heat equation, obtaining in this way an isotropic diffusion flow. In the late 80's, R. Hummel [82] noted that the heat flow is not the only parabolic PDE that can be used to create a scale-space, and indeed argued that an evolution equation which satisfies the maximum principle will define a scale-space as well. The maximum principle appears to be a natural mathematical translation of *causality*. Koenderink once again made a major contribution into the PDE's arena (this time probably involuntarily, since the consequences were not clear at all in his original formulation), when he suggested to add a thresholding operation to the process of Gaussian filtering. As later suggested in [119, 120, 161], and proved by a number of groups [4, 47, 83, 84], this leads to a geometric PDE, actually, one of the most famous ones, curvature motion. In [160] this was extended to diffusion generated motion of curves in $\mathbb{R}^3$. Solving a vector heat equation and thresholding leads to moving the curve in the direction of the normal with velocity equal to its curvature.

Perona and Malik's work [154] on anisotropic diffusion, together with the work by Rudin-Osher-Fatemi on Total Variation [159] (and Osher-Rudin on shock filters [144]), have been among the most influential papers in the area, explicitly showing the importance of understanding non-linear PDE's theory to deal with images. They proposed to replace the linear Gaussian smoothing, equivalent to isotropic diffusion via the heat flow, by a selective non-linear diffusion that preserves edges, see below. Their work opened a number of theoretical and practical questions that continue to occupy the PDE image processing community, e.g.,

[3, 157]. We should also point out that about at the same time, Price *et al.* published a very interesting paper on the use of Turing's reaction-diffusion theory for a number of image processing problems [156]. Reaction diffusion equations were also suggested to create artificial texture [188, 197].

Many of the PDE's used in image processing and computer vision are based on moving curves and surfaces with curvature based velocities. In this area, the level-set numerical method developed by Osher and Sethian [147] is very influential and examples will be provided later in this section. The representation of static objects as level-sets (zero-sets) is of course not completely new to the computer vision and image processing communities, since it is one of the fundamental techniques in mathematical morphology [164]. Considering the image itself as a collection of its level-sets, and not just as the level-set of a higher dimensional function, is a key concept in the PDE's community [2]. Implicit surfaces and level-set representations appear in computer graphics as well [9, 196].

Other works, like the segmentation approach of Mumford and Shah [123] and the snakes of Kass, Witkin, and Terzopoulos [91] have been very influential in the PDE's community as well. More on this will be mentioned below.

It should be noted that a number of the above approaches rely quite heavily on a large number of mathematical advances in differential geometry for curve evolution [66] and in viscosity solutions theory for curvature motion (see e.g., [27, 48].)

The frameworks of PDE's and geometry driven diffusion have been applied to many problems in image processing and computer vision, since the seminal works mentioned above. Examples include continuous mathematical morphology, invariant shape analysis, shape from shading, segmentation, tracking, object detection, optical flow, stereo, image denoising, image sharpening, contrast enhancement, and image quantization. In this section we provide a few examples of these. Since this is a paper in honor of Stan Osher, the presentation of the examples is of course biased by his involvement and contributions in the area. Important sources of literature in the area are the excellent collection of papers in the book edited

by Bart Romeny [157], the book by Guichard and Morel [69] that contains an outstanding description of the topic from the point of view of iterated infinitesimal filters, Sethian's book on level-sets [165], Osher-Fedkiw's long expected book, Lindeberg's book, a classic in Scale-Space theory [105], Weickert's book on anisotropic diffusion in image processing [192], Kimmel's lecture notes [97], Sapiro's recent book [163], Toga's book on Brain Warping that includes a number of PDE's based algorithms for this [181], the special issue on the March 1998 issue of the IEEE Transactions on Image Processing, the special issues in the Journal of Visual Communication and Image Representation, a series of Special Sessions at a number of IEEE International Conference on Image Processing (ICIP), the Proceedings of the Scale Space Workshops, and the 2001 Workshop on Level-Set and Variational Methods. The interested reader will find in these publications some fascinating contributions in the area of PDE's in image processing and computer vision, much beyond the few introductory examples provided below.

5.1 The Total Variation Model for Image Denoising

As mentioned above, the use of PDE's for image enhancement has become one of the most active research areas in image processing [19]. In particular, diffusion equations are commonly used for image regularization, denoising, and multiscale representations (representing the image simultaneously at several scales or levels of resolution). This started with the pioneering works in [98, 195], where the authors suggested the use of the linear heat flow for this task, given by

$$\frac{\partial u}{\partial t} = \Delta u, \tag{5.1}$$

where $u : \Omega \subset I\!R^2 \rightarrow I\!R$ represents the image gray values (the original noisy image is used as initial condition). As it is well known, this equation is the gradient-descent of

$$\int_\Omega \| \nabla u \|^2 \, d\Omega, \tag{5.2}$$

An example of the effect of the linear heat flow or Laplace equation (5.1) is presented in Figure 5.1. It is clear that although this technique can be used to denoise images, it is also

blurring them. That is, not only the noise is being removed, but the edges and the relevant information is getting destroyed as well. Moreover, it can be shown that edges are destroyed faster than the actual noise is removed [8].

Figure 5.1: Example of the heat flow (isotropic diffusion). On the left we have the original image and on the right two different time steps of the diffusion flow, showing how the image is getting blurred.

Two directions were taken to address this problem. On one hand, Perona and Malik [154] suggested to replace the linear heat flow by a PDE that preserves edges. Simultaneously, Rudin, Osher, and Fatemi [159] started to look at the modification of the variational problem (5.2). In certain cases, the two directions can be shown to be equivalent, the PDE being the gradient descent of the proposed variational formulation. Rudin, Osher and Fatemi suggested to replace the linear L_2 norm in (5.2) by the edge oriented Total Variation (TV) norm in the energy, thereby obtaining

$$\int_{\Omega} \parallel \nabla u \parallel d\Omega, \tag{5.3}$$

whose gradient descent flow is given by

$$\frac{\partial u}{\partial t} = \operatorname{div}\left(\frac{\nabla u}{\parallel \nabla u \parallel}\right). \tag{5.4}$$

We notice that in comparison with the linear heat flow, the TV one has a stopping term of the form $\frac{1}{\|\nabla u\|}$. This helps to preserve edges, as can be seen in Figure 5.2. Rudin *et al* also suggested to add constraints to this minimization, in order to avoid reaching the trivial (flat) steady state, thereby improving the results in Figure 5.2. In this case the corresponding

Lagrange multiplier is evaluated via a projection method that was found to be useful in other applications as well, e.g., [146].

Figure 5.2: Example of the TV flow for anisotropic diffusion. On the left we have the original image and on the right the result of the flow, showing how the edges are much better preserved than with the isotropic flow.

From the point of view of edge preservation, the TV flow is optimal if we limit ourselves to convex functionals [8]. Motivated by the seminal work of Perona and Malik and Rudin-Osher-Fatemi, significant theoretical and practical studies have been conducted in this kind of anisotropic diffusion flows in general and the TV flow in particular. People have studied their numerics (e.g., Mulet-Chan-Golub, Weickert, Marquina-Osher) as well as their formal mathematical properties (e.g., Alvarez-Morel-Lions, Weickert, to name just a few, and more recently Caselles *et al.* with a full study of the TV flow in general dimensions). This work has also in part motivated Cohen, DeVore, and others to connect wavelets with the TV space.

5.2 Images on Implicit Surfaces

In the last section we dealt with images on the plane. There is of course more than that, and data can be defined on surfaces. In [7] the authors dealt with this issue. A framework for solving variational problems and partial differential equations for scalar and vector-valued data defined on surfaces was introduced. The key idea is to implicitly represent the static surface as the level set of a higher dimensional static function, and solve the surface equations

in a fixed Cartesian coordinate system using this new embedding function. Implicit surfaces can be obtained for example from the algorithms in [39, 61, 100, 117, 179, 186, 199, 202]. Applications of PDE's on surfaces include computer graphics [187, 188, 197], visualization [35], weathering simulation [36], vector field computation or interpolation process [155, 194], inverse problems [53], and surface parameterization [38].

We assume then that the three dimensional surface $\mathcal{S}$ of interest is given in implicit form, as the zero level set of a given function $\phi : I\!R^3 \rightarrow I\!R$. This function is negative inside the closed bounded region defined by $\mathcal{S}$, positive outside, Lipschitz continuous a.e., with $\mathcal{S} \equiv \{x \in I\!R^3 : \phi(x) = 0\}$. To ensure that the data, which needs not to be defined outside of the surface originally, is now defined in the whole band, one simple possibility is to extend this data u defined on $\mathcal{S}$ (i.e the zero level set of ϕ) in such a form that it is constant normal to each level set of ϕ. This, which is easily realizable [26], is only done if the data is not already defined in the whole embedding space.

We will exemplify the framework with the simplest case, the heat flow or Laplace equation for scalar data defined on a surface. For scalar data u defined on the plane, that is, $u(x, y) : \Omega \subset I\!R^2 \rightarrow I\!R$, as we saw before, the heat flow is given by (5.1), and its corresponding energy by (5.2). If we now want to smooth scalar data u defined on a surface $\mathcal{S}$, that is, $u(x, y) : \mathcal{S} \rightarrow I\!R$, we must find the minimizer of the energy given by

$$\frac{1}{2} \int_{\mathcal{S}} \| \nabla_{\mathcal{S}} u \|^2 \, d\mathcal{S}. \tag{5.5}$$

The equation that minimizes this energy is its gradient descent flow (e.g., [176]):

$$\frac{\partial u}{\partial t} = \Delta_{\mathcal{S}} u. \tag{5.6}$$

Here $\nabla_{\mathcal{S}}$ is the intrinsic gradient and $\Delta_{\mathcal{S}}$ the intrinsic Laplacian or Laplace-Beltrami operator.

Classically, eq. (5.6) would be implemented in a triangulated surface, giving place to sophisticated and elaborated algorithms even for such simple flows. We now show how to simplify this when considering implicit representations.

Let $\vec{v}$ be a generic three dimensional vector, and $P_{\vec{v}}$ the operator that projects a given three dimensional vector onto the plane orthogonal to $\vec{v}$. It is then easy to show that the harmonic energy (5.5) ([40]) is equivalent to (see for example [174])

$$\frac{1}{2}\int_{\mathcal{S}} \parallel P_{\vec{N}}\nabla u \parallel^2 d\mathcal{S}, \tag{5.7}$$

where $\vec{N}$ is the normal to the surface $\mathcal{S}$. In other words, $\nabla_{\mathcal{S}} u = P_{\vec{N}}\nabla u$. That is, the gradient intrinsic to the surface ($\nabla_{\mathcal{S}}$) is just the projection onto the surface of the 3D Cartesian (classical) gradient ∇. We now embed this in the function ϕ:

$$\frac{1}{2}\int_{\mathcal{S}} \parallel \nabla_{\mathcal{S}} u \parallel^2 d\mathcal{S} = \frac{1}{2}\int_{\Omega\in I\!R^3} \parallel P_{\nabla\phi}\nabla u \parallel^2 \delta(\phi) \parallel \nabla\phi \parallel dx,$$

where $\delta(\cdot)$ stands for the delta of Dirac, and all the expressions above are considered in the sense of distributions. Note that first we got rid of intrinsic derivatives by replacing $\nabla_{\mathcal{S}}$ by $P_{\vec{N}}\nabla u$ (or $P_{\nabla\phi}\nabla u$) and then replaced the intrinsic integration ($\int_{\mathcal{S}} d\mathcal{S}$) by the explicit one ($\int_{\Omega\in I\!R^3} dx$) using the delta function. Intuitively, although the energy lives in the full space, the delta function forces the penalty to be effective only on the level set of interest. The gradient descent of this energy is given by

$$\frac{\partial u}{\partial t} = \frac{1}{\parallel \nabla\phi \parallel}\nabla\cdot(P_{\nabla\phi}\nabla u \parallel \nabla\phi \parallel). \tag{5.8}$$

In other words, this equation corresponds to the intrinsic heat flow for data on an implicit surface. But all the gradients in this PDE are defined in the three dimensional Cartesian space, not in the surface $\mathcal{S}$ (this is why we need the data to be defined at least on a band around the surface). The numerical implementation is then straightforward. Once again, due to the implicit representation, classic numerics are used, avoiding elaborate projections onto discrete surfaces and discretization on general meshes, e.g., [34, 81]. The same framework can be applied to other variational formulations as well as to PDE's defined on surfaces, e.g., the ones exemplified below [7].

A particularly interesting example is obtained when we have unit vectors defined on the surface. That is, we have data of the form $u : \mathcal{S} \to S^{n-1}$. When $n = 3$ our unit

vectors lie on the sphere. Following the work [178] for color images defined on the plane, we show in Figure 5.3 how to denoise a color image painted on an implicit surface. The basic idea is to normalize the RGB vector (a three dimensional vector) to a unit vector representing the chroma, and diffuse this unit vector with the harmonic maps flow embedded on the implicit surface extending the intrinsic heat flow example presented above. The corresponding magnitude, representing the brightness, is smoothed separately via scalar diffusion flows as those presented before for images on the plane (e.g., an intrinsic TV anisotropic heat flow). That is, we have to regularize a map from the zero level-set onto S^2 (the chroma) and another one onto $\mathbb{R}$ (the brightness).

Following the same framework and the work in [187, 188, 197], we show in Figure 5.4 the result of reaction diffusion flows solved on implicit surfaces in order to generate intrinsic patterns.

Finally, inspired by the work on line integral convolution [14] and that on anisotropic diffusion [154], the authors of [35] suggested to use anisotropic diffusion to visualize flows in 2D and 3D. The basic idea is, starting from a random image, anisotropically diffuse it in the directions dictated by the flow field. The authors presented very nice results both in 2D (flows on the plane) and 3D (flows on a surface), but once again using triangulated surfaces which introduce many computational difficulties. In a straightforward fashion we can compute these anisotropic diffusion equations on the implicit surfaces with the framework here introduced, and some results are presented in Figure 5.5.

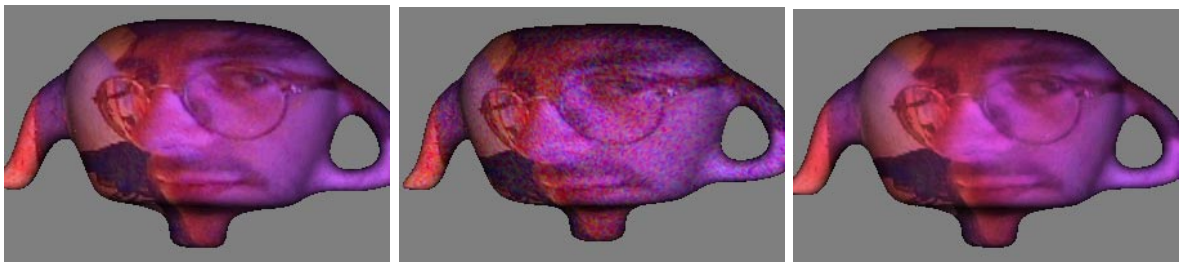


Figure 5.3: Intrinsic vector field regularization. Left: original color image. Middle: heavy noise has been added to the 3 color channels. Right: color image reconstructed after 20 steps of anisotropic diffusion of the chroma vectors.

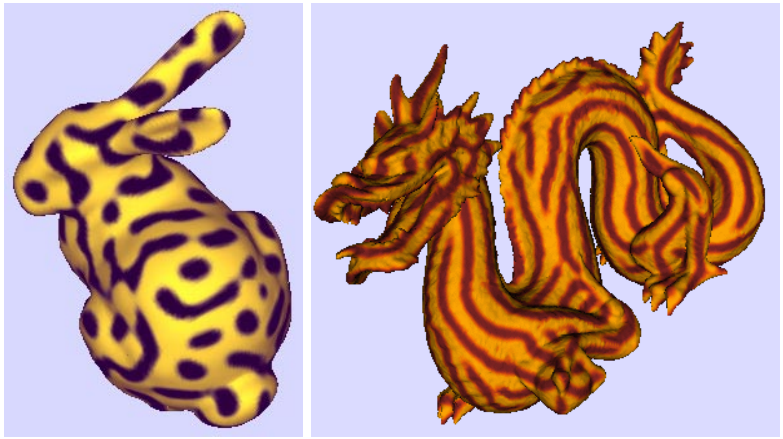


Figure 5.4: Texture synthesis via intrinsic reaction-diffusion flows on implicit surfaces. Left: isotropic. Right: anisotropic.

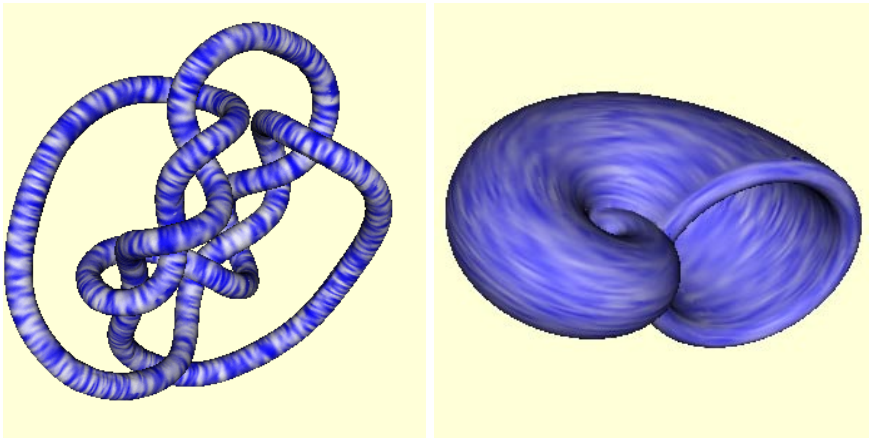


Figure 5.5: Flow visualization on implicit 3D surfaces via intrinsic anisotropic diffusion flows. Left: flow aligned with the major principal direction of the surface. Right: flow aligned with the minor principal direction of the surface. Pseudo-color representation of scalar data is used.

5.3 The Level-Set Method in Image Processing and Computer Vision

We now present a number of examples on the use of the level-set method described above for problems in image processing and computer vision.

5.3.1 Image Segmentation

One of the most popular applications of level-set methods in image processing and computer vision is for image segmentation. The contributions in this area started shortly after the work in [95] (which is one of the first papers in computer vision using the level-set method) by the works in [16, 115, 116]. These authors showed how to embed in the level-set framework the pioneering work on snakes and active contours by Terzopoulos and colleagues [91].

Consider the image on the left of Figure 5.6. Terzopoulos and colleagues suggested to detect the objects in this image (segment the image) starting with a curve that surrounds the object/s, and letting the curve deform (active-contour/snake) toward the boundary of the objects. The deformation is driven by the minimization of a given energy that penalizes non-smooth curves that do not sit at the objects boundaries. The authors of [91] proposed a Lagrangian implementation of the curve deformation process, while Caselles *et al.* and Malladi *et al.* pioneered the use of the level-set method for this approach. This added the classical topological freedom, thereby allowing the detection of multiple objects without prior knowledge of their number (later on Terzopoulos and colleagues showed a technique based on Lagrangian implementation to achieve this [118]). Following this work, in [17] (see also [18, 92, 93, 167, 180, 193] and [151] for pioneering extensions of this to object tracking), the authors showed that both approaches can be formally unified if one considers an energy given by

$$E(\mathcal{C}) = \int_{\mathcal{C}} g(\mathcal{C}) ds, \tag{5.9}$$

where ds is the Euclidean arc-length over the deforming curve $\mathcal{C} : [a, b] \to I\!R^2$ and $g(\cdot)$ is a function that penalizes curves that do not sit on the objects boundaries (a function of the image gradient for example). That is, image segmentation has been translated into finding a curve minimizing (5.9), thereby a geodesic in a space with metric $g(\cdot)$. The geodesic was computed using the level-set method. Examples are provided in Figure 5.6.

When describing image segmentation, variational problems, and PDE's, we can not avoid but think about the famous Mumford-Shah work [123], and ask ourself the relationship

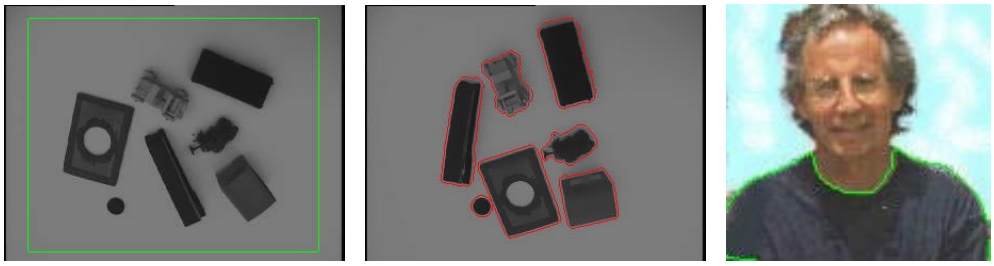


Figure 5.6: Level-set based object segmentation. The first figure on the left shows the original image and original contour, surrounding an un-known number of objects. The results of the geodesic active contours is given in the middle image. The image in the right is a result of the geodesic active contours framework implemented following Cohen and Kimmel.

between these techniques. Some of this relationship is described in [163], while additional one comes to light from recent works connecting the Mumford-Shah model and level-set techniques (see for example works by Paragios-Deriche, Yezzi *et al.*, and Chan-Vese). One of the works in this direction is presented in [23, 191]. This work is inspired in part by Zhao *et al.* [201]. In their work, multiple phases and their boundaries, represented via the level set method, evolve and interact in time, to minimize a bulk-surface energy. Combining several level set functions together, triple junctions were also represented and evolved in time. Inspired by this, Chan and Vese presented a multi-phase level set model for image segmentation. Triple junctions and complex topologies are segmented using more than one level set function. An example is provided in Figure 5.7. In this example, a multi-phase model with four phases is used, obtained by combining two level set functions. Here, the phases and their boundaries evolve in time, by minimizing an energy related to the Mumford and Shah piecewise-constant model for segmentation. We show the evolution of the curves and of the four phases, in a level set framework.

5.3.2 Stereo and 3D Reconstruction from Multiple Views

The problem of stereo is as follows: Recover the geometry of the scene when two or more images from the world are taken simultaneously. Since the internal parameters of the camera are unknown, the problem is essentially one of establishing correspondence between the

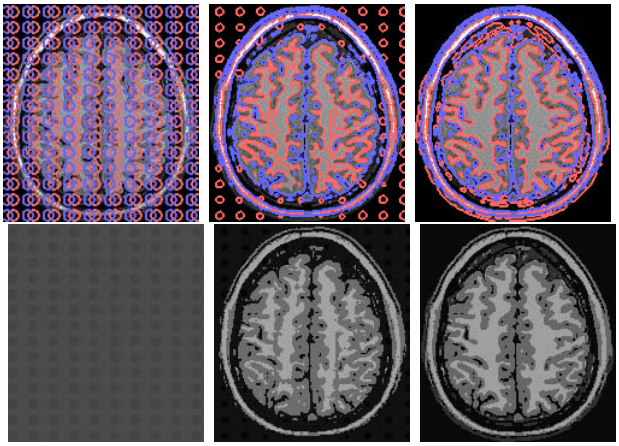


Figure 5.7: Evolution of the four-phase segmentation model from [23], using two level set functions: evolving curves (top) and phases (bottom).

views. The correspondence problem is usually addressed setting up a functional and looking for its extrema. Once the correspondence has been achieved, the 3D point is reconstructed by intersecting the corresponding optical rays. See for example [52, 68, 79] for further geometric details on the old problem of stereo.

Faugeras and Keriven pioneered the use of level-sets and the geodesic framework for this problem. They proposed to start from some initial 3D surface $\mathcal{S}_0$ and deform it toward the minimization of a given geometric functional. One of the functionals proposed in [54] reads as follows:

$$E(\mathcal{S},\vec{\mathcal{N}}) = \int\int \Psi(\mathcal{S},\vec{\mathcal{N}})da = -\sum_{i,j=1,i\neq j}^{n} \frac{\langle I_i, I_j\rangle}{\langle I_i, I_i\rangle\langle I_j, I_j\rangle}, \tag{5.10}$$

where i, j run over all the n available images I_i, and $\langle\cdot,\cdot\rangle$ is the cross correlation between the images, which includes the geometry of the perspective projection and the assumption of Lambertian surfaces [54]. Note that this approach consists basically on replacing the edge dependent 'metric' g we used for the segmentation approach by a new 'metric' Ψ that favors correlation between the collection of images. The authors work out the Euler-Lagrange equations for this formulation and embed it in the level-set framework. The example in Figure 5.8 was provided by Ronny Kimmel, and it corresponds to a simplification of the model by Faugeras-Keriven that he has recently proposed (additional examples can be found

in the home pages of Faugeras and Keriven). This work was also extended by [198], where the authors explicitly combine multi-view reconstruction with segmentation ideas as those described before. The authors suggest to find a surface whose projection properly segments the multiple given images, and an example from their work is given in Figure 5.9.

Figure 5.8: 3D reconstruction from a stereo pair using the geodesic stereo approach. First, the stereo pair is shown, followed by the reconstructed 3D shape.

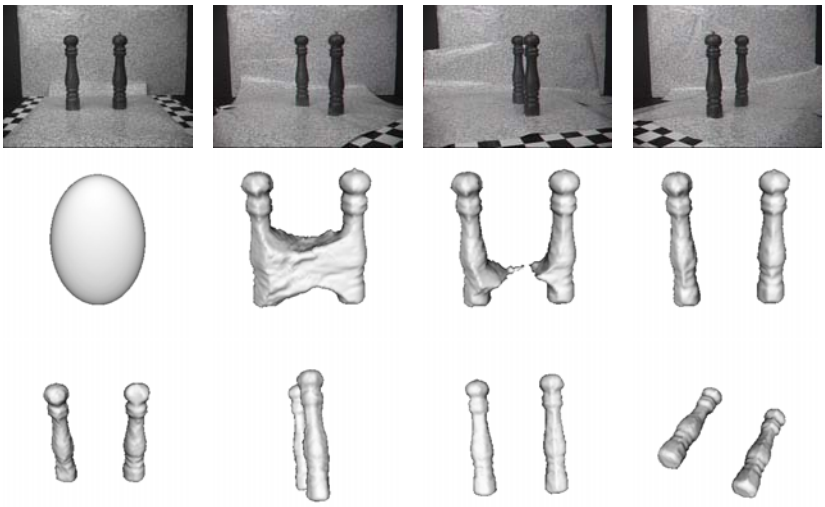


Figure 5.9: Surface reconstruction from multiple-views. The first row shows four different views. The second row shows four steps of the evolution, while the last row shows four different 3D views of the reconstructed surface.

5.4 Shape from Shading

According to the so called *Lambertian* shading rule, the 2D array of pixel gray levels, corresponding to the shading of a 3D object, is proportional to the cosine of the angle between the light source direction and the surface normal. The shape from shading problem is the inverse problem of reconstructing the 3D surface from this shading data. The history of this problem is extensive. We here described a basic technique, developed by Kimmel and Bruckstein [96] to address this problem. An outstanding contribution to the problem was done in [158], based on the theory of viscosity solutions (this is currently being extended at INRIA by Faugeras and his collaborators; see also [125].) See these references for details and an extensive literature.

Consider a smooth surface, actually a graph, given by $z(x, y)$. According to the Lambertian shading rule, the shading image $I(x, y)$ is equal (or proportional) to the inner product between the light direction $\hat{l} = (0, 0, 1)$ and the normal $\vec{\mathcal{N}}(x, y)$ to the parameterized surface. This gives the so called *irradiance equation*:

$$I(x, y) = \hat{l} \cdot \vec{\mathcal{N}} = \frac{1}{\sqrt{1 + p^2 + q^2}},$$

where $p := \partial z/\partial x$ and $q := \partial z/\partial y$. Starting from a small circle around a singular point, Bruckstein [12] observed that equal height contours $\mathcal{C}(p, t) : S \to I\!R^2$ of the surface z (t stands for the height) hold

$$\frac{\partial \mathcal{C}}{\partial t} = \frac{I}{\sqrt{1 - I^2}} \vec{n},$$

where now $\vec{n}$ is the 2D unit normal to the equal height contour (or level-set of z). This means that the classical shape from shading problem is simply a curve evolution problem, and as so, we can use all the curve evolution machinery to solve it. In particular, we can use both the level-set and the fast marching numerical techniques (the weight for the distance is always positive and given by $\sqrt{1/I^2 - 1}$). An example, courtesy of the authors of [96], is presented in Figure 5.10.

We conclude by mentioning that this work by Kimmel and Bruckstein on shape from shading using curve evolution and level-sets inspired in part the work in [137]. This presents the general connection between the unsteady and steady approaches to curve and surface evolution.

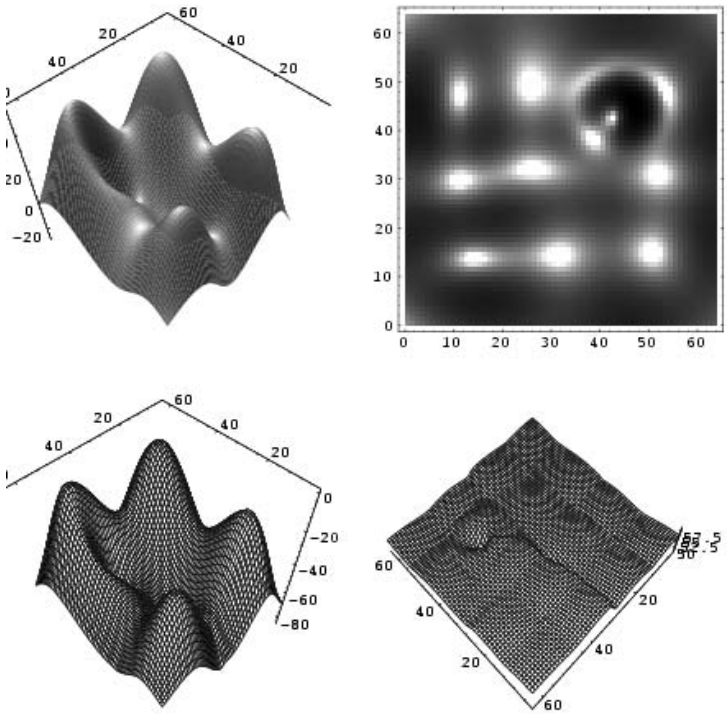


Figure 5.10: Example of shape from shading via curve evolution. The figure shows the original surface, the simulated shading, the reconstructed surface, and the reconstruction error.

References

[1] Adalsteinsson, D. and Sethian, J., *A fast level set method for propagating interfaces*, J. Comput. Phys. 118, 269-277 (1995).

[2] Alvarez, L., Guichard, F., Lions, P.L. and Morel, J.M., *Axioms and fundamental equations of image processing*, Arch. Rational Mechanics 123, 199-257 (1993).

[3] Alvarez, L., Lions, P.L. and Morel, J.M., *Image selective smoothing and edge detection by nonlinear diffusion*, SIAM J. Numer. Anal. 29, 845-866 (1992).

[4] Barles, G. and Georgelin, C., *A simple proof of convergence for an approximation scheme for computing motions by mean curvature*, SIAM J. Numer. Anal. 32, 484-500 (1995).

[5] Balsara, D. and Shu, C.-W., *Monotonicity preserving weighted essentially non-oscillatory schemes with increasingly high order of accuracy*, J. Comput. Phys. 160, 405-452 (2000).

[6] Bardi, M. and Osher, S., *The nonconvex multi-dimensional Riemann problem for Hamilton-Jacobi equations*, SIAM J. Numer. Anal. 22, 344-351 (1991).

[7] Bertalmio, M, Cheng, L.T., Osher, S. and Sapiro, G., *Variational problems and partial differential equations on implicit surfaces*, J. Comput. Phys., to appear.

[8] Black, M., Sapiro, G., Marimont, D. and Heeger, D., *Robust anisotropic diffusion*, IEEE Trans. Image Processing 7:3, 421-432 (1998).

[9] Bloomenthal, J., *Introduction to Implicit Surfaces*, Morgan Kaufmann Publishers, Inc., San Francisco, California, 1997.

[10] Boris, J.P. and Book, D.L., *Flux corrected transport I, SHASTA, a fluid transport algorithm that works*, J. Comput. Phys. 11, 38-69 (1973).

[11] Brenier, Y. and Osher, S., *The discrete one-sided Lipschitz condition for convex scalar conservation laws*, SIAM J. Numer. Anal. 25, 8-23 (1988).

[12] Bruckstein, A.M., *On shape from shading*, Comp. Vision Graph. Image Processing 44, 139-154 (1988).

[13] Burchard, P., Cheng, L.-T., Merriman, B. and Osher, S., *Motion of curves in three spatial dimensions using a level set approach*, J. Comput. Phys. 165, 463-502 (2001).

[14] Cabral, B. and Leedom, C., *Imaging vector fields using line integral convolution*, ACM Computer Graphics (SIGGRAPH '93) 27:4, 263-272 (1993).

[15] Caiden, R., Fedkiw, R. and Anderson, C., *A numerical method for two-phase flow consisting of separate compressible and incompressible regions*, J. Comput. Phys. 166, 1-27 (2001).

[16] Caselles, V., Catte, F., Coll, T. and Dibos, F., *A geometric model for active contours*, Numerische Mathematik 66, 1-31 (1993).

[17] Caselles, V., Kimmel, R. and Sapiro, G., *Geodesic active contours*, International Journal of Computer Vision 22:1, 61-79 (1997).

[18] Caselles, V., Kimmel, R., Sapiro, G. and Sbert, C., *Minimal surfaces based object segmentation*, IEEE-PAMI 19:4, 394-398 (1997).

[19] Caselles, V., Morel, J.M., Sapiro, G. and Tannenbaum, A., Editors, *Special Issue on Partial Differential Equations and Geometry-Driven Diffusion in Image Processing and Analysis*, IEEE Trans. Image Processing 7, March 1998.

[20] Cercignani, C., Gamba, I., Jerome, J. and Shu, C.-W., *Device benchmark comparisons via kinetic, hydrodynamic, and high-field models*, Comput. Meth. Appl. Mech. Engin. 181, 381-392 (2000).

[21] Chakravarthy, S. and Osher, S., *Numerical experiments with the Osher upwind scheme for the Euler equations*, AIAA J. 21, 1241-1248 (1983).

[22] Chan, T. and Vese, L., *Active contour model without edges*, in Lect. Notes in C.S., edited by M. Neilsen, P. Johansen, O.F. Olson and J. Weickert, Springer-Verlag, Berlin/New York, v. 1687, p. 141 (1999).

[23] Chan, T. and Vese, L., *Image segmentation using level sets and the piecewise-constant Mumford-Shah model*, UCLA CAM Report 00-14, 2000.

[24] Chang, Y., Hou, T., Merriman, B. and Osher, S., *A level set formulation of Eulerian interface capturing methods for incompressible fluid flows*, J. Comput. Phys. 124, 449-464 (1996).

[25] Chen, S., Merriman, B., Kang, M., Caflisch, R., Ratsch, C., Guyre, M. Fedkiw, R., Anderson, C. and Osher, S., *A level set method for thin film epitaxial growth*, J. Comput. Phys. 167, 475-500 (2001).

[26] Chen, S., Merriman, B., Osher, S. and Smereka, P., *A simple level set method for solving Stefan problems*, J. Comput. Phys. 135, 8-29 (1997).

[27] Chen, Y. G., Giga, Y. and Goto, S., *Uniqueness and existence of viscosity solutions of generalized mean curvature flow equations*, J. Diff. Geom. 33, 749-786 (1991).

[28] Cheng, L.-T., Burchard, P., Merriman, B. and Osher, S., *Motion of curves constrained on surfaces using a level set approach*, UCLA CAM Report 00-32, J. Comput. Phys., to appear 2001.

[29] Chopp, D., *Computing minimal surfaces via level set curvature flow*, J. Comput. Phys. 106, 77-91 (1993).

[30] Crandall, M. and Lions, P.-L., *Viscosity solutions of Hamilton-Jacobi equations*, Trans. Amer. Math. Soc. 277, 1-42 (1983).

[31] Crandall, M. and Lions, P.-L., *Two approximations of solutions of Hamilton-Jacobi equations*, Math. Comp. 43, 1-19 (1984).

[32] Crandall, M. and Majda, A., *Monotone difference approximations for scalar conservation laws*, Math. Comput. 34, 1-21 (1980).

[33] Deacon, A.G. and Osher, S., *A finite element method for a boundary value problem of mixed type*, SIAM J. Numer. Anal. 16, 756-778 (1979).

[34] Desbrun, M., Meyer, M., Schröder, P., and Barr, A.H., *Discrete differential-geometry operators in nD*, Multi-res modeling group TR, Caltech, September 2000 (obtained from www.multires.caltech.edu).

[35] Diewald, U., Preufer, T. and Rumpf, M., *Anisotropic diffusion in vector field visualization on Euclidean domains and surfaces*, IEEE Trans. Visualization and Computer Graphics 6, 139-149 (2000).

[36] Dorsey J. and Hanrahan, P., *Digital materials and virtual weathering*, Scientific American 282:2, 46-53 (2000).

[37] Durlofsky, L.J., Engquist, B. and Osher, S., *Triangle based adaptive stencils for the solution of hyperbolic conservation laws*, J. Comput. Phys. 98, 64-73 (1992).

[38] Eck, M., DeRose, T., Duchamp, T., Hoppe, H., Lounsbery, M. and Stuetzle, W., *Multi-resolution analysis of arbitrary meshes*, Computer Graphics (SIGGRAPH '95 Proceedings), 173-182, 1995.

[39] Eck, M. and Hoppe, H., *Automatic reconstruction of B-spline surfaces of arbitrary topological type*, Computer Graphics, 1996.

[40] Eells, J. and Lemarie, L., *A report on harmonic maps*, Bull. London Math. Soc. 10:1, 1-68 (1978).

[41] Engquist, B., Fatemi, E. and Osher, S., *Numerical solution of the high frequency asymptotic expansion for hyperbolic equations*, Proc. 10th Ann. Review of Progress in Applied Comp. Electromagnetics, Monterey, CA, 32-45 (1994).

[42] Engquist, B. and Osher, S., *Stable and entropy satisfying approximations for transonic flow calculations*, Math. Comp. 34, 45-57 (1980).

[43] Engquist, B. and Osher, S., *One-sided difference approximations for nonlinear conservation laws*, Math. Comp. 36, 321-351 (1981).

[44] Engquist, B., Osher, S. and Zhong, S., *Fast wavelet based algorithms for linear evolution equations*, SIAM J. Sci. Stat. Comput. 15:4, 755-775 (1994).

[45] Engquist, B., Runborg, O. and Tornberg, A.-K., *High frequency wave propagation by the segment projection method*, UCLA CAM Report 01-13, (2001), submitted to J. Comput. Phys.

[46] Enright, D., Fedkiw, R., Ferziger, J. and Mitchell, I. *A hybrid particle level set method for improved interface capturing*, J. Comput. Phys., in review.

[47] Evans, L.C., *Convergence of an algorithm for mean curvature motion*, Indiana University Mathematics Journal 42, 553-557 (1993).

[48] Evans, L.C. and Spruck, J., *Motion of level sets by mean curvature, I*, J. Differential Geometry 33, 635-681 (1991).

[49] Fatemi, E., Engquist, B. and Osher, S., *Numerical solution of the high frequency asymptotic expansion of the scalar wave equation*, J. Comput. Phys. 120, 145-155 (1995).

[50] Fatemi, E., Engquist, B. and Osher, S., *Finite difference methods for the nonlinear equations of perturbed geometric optics*, ACES J. 11, 90-98 (1996).

[51] Fatemi, E., Jerome, J. and Osher, S., *Solution of the hydrodynamic device model using high order nonoscillatory shock capturing algorithms*, IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems 10, 232-243 (1991).

[52] Faugeras, O.D. *Three-Dimensional Computer Vision: A Geometric Viewpoint*, MIT Press, Cambridge, MA, 1993.

[53] Faugeras, O., Clément, F., Deriche, R., Keriven, R., Papadopoulo, T., Gomes, J., Hermosillo, G., Kornprobst, P. Lingrad, D., Roberts, J. Viéville, T. and Devernay, F., *The inverse EEG and MEG problems: The adjoint state approach I: The continuous case*, INRIA Research Report 3673, June 1999.

[54] Faugeras, O.D. and Keriven, R., *Variational principles, surface evolution, PDE's, level-set methods, and the stereo problem*, IEEE Trans. Image Processing 7:3, 336-344 (1998).

[55] Fedkiw, R., *Coupling an Eulerian fluid calculation to a Lagrangian solid calculation with the ghost fluid method*, J. Comput. Phys., in review.

[56] Fedkiw, R., Aslam, T., Merriman, B. and Osher, S., *A non-oscillatory Eulerian approach to interfaces in multimaterial flows (the ghost fluid method)*, J. Comput. Phys. 152, 457-492 (1999).

[57] Fedkiw, R., Aslam, T., and Xu, S., *The ghost fluid method for deflagration and detonation discontinuities*, J. Comput. Phys. 154, 393-427 (1999).

[58] Fedkiw, R., Merriman, B. and Osher, S., *High accuracy numerical methods for thermally perfect gas flows with chemistry*, J. Comput. Phys. 132, 175-190 (1997).

[59] Fedkiw, R., Merriman, B. and Osher, S., *Efficient characteristic projection in upwind difference schemes for hyperbolic systems; the complementary projection method*, J. Comput. Phys. 141, 22-36 (1998).

[60] Foster, N. and Fedkiw, R., *Practical animation of liquids*, Siggraph 2001 Annual Conference (2001).

[61] Frisken, S.F., Perry, R.N., Rockwood, A. and Jones, T., *Adaptively sampled fields: A general representation of shape for computer graphics*, Computer Graphics (SIGGRAPH), New Orleans, July 2000.

[62] Gabor, D., *Information theory in electron microscopy*, Laboratory Investigation 14, 801-807 (1965).

[63] Giga, Y. and Sato, M.-H., *A level set approach to semicontinuous solutions for Cauchy problems*, to appear in Comm. in PDE (2001).

[64] Godunov, S.K. and Ryabenkii, V.S., *Spectral criteria for the stability of boundary problems for non self-adjoint difference equations*, Uspekhi Mat. Nauk 18, (1963).

[65] Gottlieb, S., Shu, C-W. and Tadmor, E., *Strong stability preserving high order time discretization methods*, SIAM Review 43, 89-112 (2001).

[66] Grayson, M., *The heat equation shrinks embedded plane curves to round points*, J. Diff. Geom. 26, 285-314 (1987).

[67] Green, M. and Osher, S., *Steiner polynomials, Wulff flows and some new isoperimetric inequalities for convex plane curves*, Asian J. Math. 3, 659-676 (1999).

[68] Grimson, W.E.L., *From Images to Surfaces*, MIT Press, Cambridge, MA, 1981.

[69] Guichard. F. and Morel, J.M., *Introduction to Partial Differential Equations in Image Processing*, Tutorial Notes, *IEEE Int. Conf. Image Proc.*, Washington, DC, October 1995.

[70] Gustafsson, B., Kreiss, H.-O. and Sundstrom, A., *Stability theory of difference approximations for mixed initial boundary value problems. II*, Math. Comp. 26, 649-686 (1972).

[71] Gyure, M., Ratsch, C., Merriman, B., Caflisch, R., Osher, S., Zinck, J. and Vvedensky, D., *Level set Methods for the simulation of epitaxial phenomena*, Phys. Rev. E. 58, 6927-6930 1998.

[72] Harabetian, E. and Osher, S., *Regularization of ill-posed problems via the level set approach*, SIAM J. Appl. Math. 58, 1689-1706 (1998).

[73] Harabetian, E., Osher, S. and Shu, C.-W., *An Eulerian approach for vortex motion using a level set regularization procedure*, J. Comput. Phys. 127, 15-26 (1996).

[74] Harten, A., *High resolution schemes for hyperbolic conservation laws*, J. Comput. Phys. 49, 357-393 (1983).

[75] Harten, A. and Osher, S., *Uniformly high-order accurate non-oscillatory schemes, I*, SIAM J. Numer. Anal. 24, 279-304 (1987).

[76] Harten, A., Engquist, B., Osher, S. and Chakravarthy, S., *Uniformly high-order accurate essentially non-oscillatory schemes III*, J. Comput. Phys. 71, 231-303 (1987).

[77] Harten, A., Osher, S., Engquist, B. and Chakravarthy, S., *Some results on uniformly high order accurate essentially non-oscillatory schemes*, Appl. Numer. Math. 2, 347-377 (1986).

[78] Helmsen, J., Puckett, E., Colella, P. and Dorr, M., *Two new methods for simulating photolithography development in 3D*, Proc. SPIE 2726, 253-261 (1996).

[79] Horn, B.K.P., *Robot Vision*, MIT Press, Cambridge, MA, 1986.

[80] Hu, C. and Shu, C.-W., *Weighted essentially non-oscillatory schemes on triangular meshes*, J. Comput. Phys. 150, 97-127 (1999).

[81] Huiskamp, G., *Difference formulas for the surface Laplacian on a triangulated surface*, J. Comput. Phy. 95, 477-496 (1991).

[82] Hummel, R.A., *Representations based on zero-crossings in scale-space*, Proc. IEEE CVPR, 204-209 (1986).

[83] Ishii, H., *A generalization of Bence, Merriman, and Osher algorithm for motion by mean curvature*, In A. Damlamian, J. Spruck, and A. Visintin, Editors, *Curvature Flows and Related Topics*, pp. 111-127, Gakkôtosho, Tokyo, 1995.

[84] H. Ishii, G. E. Pires, and P. E. Souganidis, *Threshold dynamics type schemes for propagating fronts*, J. Math. Soc. Japan 51(2), 267-308 (1999).

[85] Jain, A.K., *Partial differential equations and finite-difference methods in image processing, part 1: Image representation*, J. of Optimization Theory and Applications 23, 65-91 (1977).

[86] Jerome, J. and Shu, C.-W., *Transport effects and characteristic modes in the modeling and simulation of submicron devices*, IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems 14, 917-923 (1995).

[87] Jiang, G.-S. and Peng, D., *Weighted ENO schemes for Hamilton Jacobi equations*, SIAM J. Sci. Comput. 21, 2126-2143 (2000).

[88] Jiang, G.-S. and Shu, C.-W., *Efficient implementation of weighted ENO schemes*, J. Comput. Phys. 126, 202-228 (1996).

[89] Kang, M., Fedkiw, R., and Liu, X.-D., *A boundary condition capturing method for multiphase incompressible flow*, J. Sci. Comput. 15, 323-360 (2000).

[90] Karni, S., *Multicomponent flow calculations by a consistent primitive algorithm*, J. Comput. Phys. 112, 31-43 (1994).

[91] Kass, M., Witkin, A. and Terzopoulos, D., *Snakes: Active contour models*, International Journal of Computer Vision 1, 321-331 (1988).

[92] Kichenassamy, S., Kumar, A., Olver, P., Tannenbaum, A. and Yezzi, A., *Gradient flows and geometric active contour models*, Proc. Int. Conf. Comp. Vision '95, 810-815, Cambridge, June 1995.

[93] Kichenassamy, S., Kumar, A., Olver, P., Tannenbaum, A. and Yezzi, A., *Conformal curvature flows: from phase transitions to active vision*, Archive for Rational Mechanics and Analysis 134, 275-301 (1996).

[94] Kim, Y.-T., Goldenfeld, N. and Dantzig, J., *Computation of dendritic microstructures using a level set method*, Phys. Rev. E 62, 2471-2474 (2000).

[95] Kimia, B. B., Tannenbaum, A. and Zucker, S. W. *Toward a computational theory of shape: An overview*, Lecture Notes in Computer Science 427, 402-407, Springer-Verlag, New York, 1990.

[96] Kimmel, R. and Bruckstein, A.M., *Tracking level sets by level sets: A method for solving the shape from shading problem*, CVIU 62(1), 47-58 (1995).

[97] Kimmel, R., *Numerical geometry of images: Theory, algorithms, and applications*, Technion CIS Report 9910, October 1999.

[98] Koenderink, J.J. *The structure of images*, Biological Cybernetics 50, 363-370 (1984).

[99] Kreiss, H.-O., *Difference approximations for the initial-boundary value problem for hyperbolic differential equations*, in Numerical Solutions of Nonlinear Differential Equations. Ed., D. Greenspan, John Wiley, New York, 141-166 (1966).

[100] Krishnamurthy, V. and Levoy, M., *Fitting smooth surfaces to dense polygon meshes*, Proc SIGGRAPH '96 (New Orleans, LA), in Computer Graphics Proceedings, ACM SIGGRAPH, 313-324 (1996).

[101] Kupka, I.A.K. and Osher, S., *On the wave equation in a multi-dimensional corner*, Comm. Pure Appl. Math. 24, 381-393 (1971).

[102] Lafon, F. and Osher, S., *High order two dimensional nonoscillatory methods for solving Hamilton-Jacobi equations*, J. Comput. Phys. 123, 235-253 (1996).

[103] Lagnado, R. and Osher, S., *Reconciling differences*, Risk Magazine 10, 79-83 (1997).

[104] Lagnado, R. and Osher, S., *A technique for calibrating derivative security pricing models, numerical solution of an inverse problem*, J. Comput. Finance 1, 13-25 (1997).

[105] Lindeberg, T., *Scale-Space Theory in Computer Vision*, Kluwer, 1994.

[106] Liu, X.-D. and Osher, S., *Nonoscillatory high order accurate self similar maximum principle satisfying shock capturing schemes*, SIAM J. Numer. Anal. 33, 760-779 (1996).

[107] Liu, X.-D. and Osher, S., *Convex ENO high-order multidimensional schemes without field by field projection or staggered grids*, J. Comput. Phys. 142, 304-330 (1998).

[108] Liu, X.-D., Osher, S. and Chan, T. *Weighted essentially non-oscillatory schemes*, J. Comput. Phys. 115, 200-212 (1994).

[109] Liu, X.-D., Fedkiw, R. and Kang, M., *A boundary condition capturing method for Poisson's equation on irregular domains*, J. Comput. Phys. 160, 151-178 (2000).

[110] Majda, A., McDonough, J. and Osher, S., *The Fourier method for nonsmooth initial data*, Math. Comp. 32, 1041-1081 (1978).

[111] Majda, A. and Osher, S., *Reflections of singularities at the boundary*, Comm. Pure Appl. Math. 28, 479-499 (1975).

[112] Majda, A. and Osher, S., *Initial-boundary value problems for hyperbolic equations with uniformly characteristic boundary*, Comm. Pure Appl. Math. 28, 607-675 (1975).

[113] Majda, A. and Osher, S., *Propagation of error into regions of smoothness for accurate difference approximations to hyperbolic equations*, Comm. Pure Appl. Math. 30, 671-705 (1977).

[114] Majda, A. and Osher, S., *A systematic approach for correcting nonlinear instabilities: the Lax-Wendroff scheme for scalar conservation laws*, Numer. Math. 30, 429-452 (1978).

[115] Malladi, R., Sethian, J.A. and Vemuri, B.C., *Evolutionary fronts for topology independent shape modeling and recovery*, Proc. of the 3rd ECCV, Stockholm, Sweden, pp.3-13, 1994.

[116] Malladi, R., Sethian, J.A. and Vemuri, B.C., *Shape modeling with front propagation: A level set approach*, IEEE Trans. on PAMI 17, 158-175 (1995).

[117] Mauch, S., *Closest point transform*, www.ama.caltech.edu/∼seanm/software/cpt/cpt.html.

[118] McInerney, T. and Terzopoulos, D., *Topologically adaptable snakes*, Proc. ICCV, Cambridge, MA, June 1995.

[119] Merriman, B., Bence, J. and Osher, S., *Diffusion generated motion by mean curvature*, in J. E. Taylor, Editor, *Computational Crystal Growers Workshop*, pp. 73-83, American Mathematical Society, Providence, Rhode Island, 1992.

[120] Merriman, B., Bence, J. and Osher, S., *Motion of multiple junctions: A level-set approach*, J. Comput. Phys. 112, 334-363 (1994).

[121] Merriman, B., Caflisch, R. and Osher, S. *Level set methods with an application to modeling the growth of thin films*, in "Free boundary value problems, theory and applications", eds. I. Athanasopoulos, G. Makreakis and J. Rodrigues (CRC Press, Boca Raton), pp. 51-70 (1999).

[122] Mulder, W., Osher, S., and Sethian, J., *Computing interface motion in compressible gas dynamics*, J. Comput. Phys. 100, 209-228 (1992).

[123] Mumford, D. and Shah, J., *Optimal approximations by piecewise smooth functions and variational problems*, Comm. Pure Appl. Math. 42, 577-685 (1989).

[124] Nguyen, D., Fedkiw, R. and Kang, M., *A boundary condition capturing method for incompressible flame discontinuities*, J. Comput. Phys. 172, 71-98 (2001).

[125] Oliensis, J. and Dupuis, P., *Direct method for reconstructing shape from shading*, Proceedings SPIE Conf. 1570 on Geometric Methods in Computer Vision, 116-128, 1991.

[126] Osher, S., *Systems of difference equations with general homogeneous boundary conditions*, Trans. Amer. Math. Soc. 137, 177-201 (1969).

[127] Osher, S., *On systems of difference equations with wrong boundary conditions*, Math. Comp. 23, 335-340 (1969).

[128] Osher, S., *Stability of parabolic difference approximations to certain mixed initial boundary value problems*, Math. Comp. 26, 13-39 (1972).

[129] Osher, S., *Initial boundary value problems for hyperbolic systems in regions with corners, I.* Trans. Amer. Math. Soc. 176, 141-164 (1973).

[130] Osher, S., *On Green's function for the biharmonic equation in a right angle wedge*, J. Math. Anal. Appl. 43, 705-716 (1973).

[131] Osher, S., *Initial boundary value problems for hyperbolic systems in regions with corners, II.* Trans. Amer. Math. Soc. 198, 151-175 (1974).

[132] Osher, S., *Boundary value problems for equations of mixed type: I. The Lavrentev-Bitsadze model*, Comm. in P.D.E. 2, 499-547 (1977).

[133] Osher, S., *Numerical solution of singular perturbation problems and one-sided difference schemes*, North Holland Math. Studies 47, 1981.

[134] Osher, S., *The Riemann problem for nonconvex scalar conservation laws and the Hamilton-Jacobi equations*, Proc. Amer. Math. Soc. 89, 641-646 (1983).

[135] Osher, S., *Riemann solvers, the entropy condition and difference approximations*, SIAM J. Numer. Anal. 21, 217-235 (1984).

[136] Osher, S., *Convergence of generalized MUSCL schemes*, SIAM J. Numer. Anal. 22, 947-961 (1985).

[137] Osher, S., *A level set formulation for the solution of the Dirichlet problem for Hamilton-Jacobi equations*, SIAM J. Anal. 24, 1145-1152 (1993).

[138] Osher, S. and Chakravarthy, S., *Upwind schemes and boundary conditions with applications to Euler equations in general geometries*, J. Comput. Phys. 50, 447-481 (1983).

[139] Osher, S. and Chakravarthy, S., *High resolution schemes and the entropy condition*, SIAM J. Numer. Anal. 21, 955-984 (1984).

[140] Osher, S. and Chakravarthy, S., *Very high order accurate TVD schemes*, in IMA Volumes in Mathematics and Its Applications, 2, 229-274 (1986), Springer-Verlag.

[141] Osher, S., Cheng, L.-T., Kang, M., Shim, H. and Tsai, Y.-H., *Geometric optics in a phase space and Eulerian framework*, report of LS, Inc #LS-01-01, (Sept. 2001), submitted to J. Comput. Phys.

[142] Osher, S., Hafez, M. and Whitlow Jr., W., *Entropy condition satisfying approximations for the full potential equation of transonic flow*, Math. Comp. 44, 1-29 (1985).

[143] Osher, S. and Merriman, B., *The Wulff shape as the asymptotic limit of a growing crystalline interface*, Asian J. Math. 1, 560-571 (1997).

[144] Osher, S. and Rudin, R.T., *Feature-oriented image enhancement using shock filters*, SIAM J. Numer. Anal. 27, 919-940 (1990).

[145] Osher, S. and Sanders, R., *Numerical approximations to nonlinear conservation laws with locally varying time and space grids*, Math. Comp. 41, 321-336 (1983).

[146] Osher, S. and Santosa, F., *Level set method for optimization problems involving geometry and constraints, I. Frequencies of a two-density inhomogeneous drum*, J. Comput. Phys. 171, 277-288 (2001).

[147] Osher, S. and Sethian, J., *Fronts propagating with curvature dependent speed: algorithms based on Hamilton-Jacobi formulations*, J. Comput. Phys. 79, 12-49 (1988).

[148] Osher, S. and Shu, C.-W., *High order essentially non-oscillatory schemes for Hamilton-Jacobi equations*, SIAM J. Numer. Anal. 28, 902-921 (1991).

[149] Osher, S. and Solomon, F., *Upwind difference schemes for hyperbolic systems of conservation laws*, Math. Comp. 38, 339-374 (1982).

[150] Osher, S. and Tadmor, E., *On the convergence of difference approximations to scalar conservation laws*, Math. Comp. 50, 19-51 (1988).

[151] Paragios, N. and Deriche, R., *A PDE-based level-set approach for detection and tracking of moving objects*, Proc. Int. Conf. Comp. Vision '98, Bombay, India, January 1998.

[152] Peng, D., Osher, S., Merriman, B, and Zhao, H.-K., *The geometry of Wulff crystal shapes and its relations with Riemann problems*, in Contemporary Math., edited by G.Q. Chen and E. DeBenedetto, AM. Math. Soc. Providence, R.I., v. 238, p. 251 (1999).

[153] Peng, D., Merriman, B., Osher, S., Zhao, H.-K., and Kang, M., *A PDE-based fast local level set method*, J. Comput. Phys. 155, 410-438 (1999).

[154] Perona, P. and Malik, J., *Scale-space and edge detection using anisotropic diffusion*, IEEE Trans. Pattern. Anal. Machine Intell. 12, 629-639 (1990).

[155] Praun, E., Finkelstein, A. and Hoppe, H., *Lapped textures*, ACM Computer Graphics (SIGGRAPH), New Orleans, July 2000.

[156] Price, C.B., Wambacq, P. and Oosterlink, A. *Image enhancement and analysis with reaction-diffusion paradigm*, IEE Proc. 137, 136-145 (1990).

[157] Romeny, B., Editor, *Geometry Driven Diffusion in Computer Vision*, Kluwer, 1994.

[158] Rouy, E. and Tourin, A., *A viscosity solutions approach to shape-from-shading*, SIAM J. Num. Anal. 29, 867-884 (1992).

[159] Rudin, L.I., Osher, S. and Fatemi, E., *Nonlinear total variation based noise removal algorithms*, Phyica D 60, 259-268 (1992).

[160] Ruuth, S., Fedkiw, R., Xin, J. and Osher, S., *A diffusion generated approach to the curvature motion of filaments*, J. Nonlinear Science, to appear.

[161] Ruuth, S., Merriman, B. and Osher, S., *Convolution generated motion as a link between cellular automata and continuum pattern dynamics*, J. Comput. Phys. 151, 836-861 (1999).

[162] Ruuth, S., Merriman, B. and Osher, S., *A fixed grid method for capturing the motion of self-intersecting interfaces and related PDEs*, J. Comput. Phys. 163, 1-21 (2000).

[163] Sapiro, G., *Geometric Partial Differential Equations and Image Processing*, Cambridge University Press, New York, January 2001.

[164] Serra, J., *Image Analysis and Mathematical Morphology, vol. 2: Theoretical Advances*, Academic Press, 1988.

[165] Sethian, J.A., *Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision and Materials Sciences,* Cambridge University Press, Cambridge-UK, 1996.

[166] Sethian, J.A., *A fast marching method for three dimensional photolithography development*, Proc. SPIE 2726, p. 261, (1996).

[167] Shah, J., *A common framework for curve evolution, segmentation, and anisotropic diffusion*, Proc. CVPR, San Francisco, June 1996.

[168] Shi, J., Hu, C. and Shu, C.-W., *A technique of treating negative weights in WENO schemes*, J. Comput. Phys., to appear.

[169] Shu, C.-W., *TVB uniformly high-order schemes for conservation laws*, Math. Comp., 49, 105-121 (1987).

[170] Shu, C.-W., *Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws*, in *Advanced Numerical Approximation of Nonlinear Hyperbolic Equations*, B. Cockburn, C. Johnson, C.-W. Shu and E. Tadmor (Editor: A. Quarteroni), Lecture Notes in Mathematics, volume 1697, Springer, 1998, pp.325-432.

[171] Shu, C.-W. and Osher, S., *Efficient implementation of essentially non-oscillatory shock capturing schemes*, J. Comput. Phys. 77, 439-471 (1988).

[172] Shu, C.-W. and Osher, S., *Efficient implementation of essentially non-oscillatory shock capturing schemes II*, J. Comput. Phys. 83, 32-78 (1989).

[173] Shu, C.-W., Zang, T.A., Erlebacher, G., Whitaker, D. and Osher, S., *High-order ENO schemes applied to two- and three-dimensional compressible flow*, Appl. Numer. Math. 9, 45-71 (1992).

[174] Simon, L., *Lectures on Geometric Measure Theory*, Australian National University, Australia, 1984.

[175] Steinhoff, J., Fang, M. and Wang, L., *A new Eulerian method for the computation of propagating short acoustic and electromagnetic pulses*, J. Comput. Phys. 157, 683-706 (2000).

[176] Struwe, M., *On the evolution of harmonic mappings of Riemannian surfaces*, Comment. Math. Helvetici 60, 558-581 (1985).

[177] Sussman, M., Smereka, P. and Osher, S., *A level set approach for computing solutions to incompressible two-phase flow*, J. Comput. Phys. 114, 146-159 (1994).

[178] Tang, B., Sapiro, G. and Caselles, V., *Diffusion of general data on non-flat manifolds via harmonic maps theory: The direction diffusion case*, Int. Journal Computer Vision 36:2, 149-161 (2000).

[179] Taubin, G., *Estimation of planar curves, surfaces, and nonplanar space curves defined by implicit equations with applications to edge and range image segmentation*, IEEE Trans. PAMI 13:11, 1115-1138 (1991).

[180] Tek, H and Kimia, B.B., *Image segmentation by reaction-diffusion bubbles*, Proc. ICCV'95, 156-162, Cambridge, June 1995.

[181] Toga, A.W., *Brain Warping*, Academic Press, New York, 1998.

[182] Ton, V.T., Karagozian, A.R., Osher, S. and Engquist, B., *Numerical simulation of high speed chemically reactive flow*, Ther. Comput. Fluid Dyn. 6, 161-179 (1994).

[183] Trygvasson, G., Dahm, W.J.A. and Sbeih, K., *Fine structure of vortex sheet rollup by viscous and inviscid simulation*, J. Fluids Engin. 113, 31-36 (1991).

[184] Tsai, Y.-H., Giga, Y. and Osher, S., *A level set approach for computing discontinuous solutions of Hamilton-Jacobi equations*, to appear in Math. Comp.

[185] Tsitsiklis, J., *Efficient algorithms for globally optimal trajectories*, Proceedings of the 33rd Conference on Decision and Control, Lake Buena Vista, LF, 1368-1373, December 1994.

[186] Tsitsiklis, J., *Efficient algorithms for globally optimal trajectories*, IEEE Transactions on Automatic Control 40, 1528-1538 (1995).

[187] Turing, A., *The chemical basis of morphogenesis*, Philosophical Transactions of the Royal Society B 237, 37-72 (1952).

[188] Turk, G., *Generating textures on arbitrary surfaces using reaction-diffusion*, Computer Graphics 25:4, 289-298 (1991).

[189] van Leer, B., *Towards the ultimate conservative difference scheme V. A second order sequel to Godunov's method*, J. Comput. Phys. 32, 101-136 (1979).

[190] Varah, J., *Stability of difference approximations to the mixed initial boundary value problem for parabolic systems*, SIAM J. Numer. Anal. 8, 598-615 (1971).

[191] Vese, L.A. and Chan, T.F., *A multiphase level set framework for image segmentation using the Mumford and Shah model*, UCLA Math CAM Report 01-25, September 2001

[192] Weickert, J., *Anisotropic Diffusion in Image Processing*, ECMI Series, Teubner-Verlag, Stuttgart, Germany, 1998.

[193] Whitaker, R.T., *Algorithms for implicit deformable models*, Proc. ICCV'95, 822-827, Cambridge, June 1995.

[194] Winkenbach, G. and Salesin, D.H., *Rendering parametric surfaces in pen and ink*, Computer Graphics (SIGGRAPH 96), 469-476 (1996).

[195] Witkin, A.P., *Scale-space filtering*, Int. Joint. Conf. Artificial Intelligence 2, 1019-1021 (1983).

[196] Witkin, A. and Heckbert, P., *Using particles to sample and control implicit surfaces*, Computer Graphics (SIGGRAPH), 269-278 (1994).

[197] Witkin, A. and Kass, M., *Reaction-diffusion textures*, Computer Graphics (SIGGRAPH) 25:4, 299-308 (1991).

[198] Yezzi, A. and Soatto, S., *Stereoscopic segmentation*, IEEE Intl. Conf. on Computer Vision, 59-66, Vancouver, July 2001.

[199] Yngve, G. and Turk, G., *Creating smooth implicit surfaces from polygonal meshes*, Technical Report GIT-GVU-99-42, Graphics, Visualization, and Usability Center. Georgia Institute of Technology, 1999 (obtained from www.cc.gatech.edu/gvu/geometry/publications.html).

[200] Zhang, Y.-T. and Shu, C,-W., *High order WENO schemes for Hamilton-Jacobi equations on triangular meshes*, SIAM J. Sci. Comput., submitted.

[201] Zhao, H.-K., Chan, T., Merriman, B. and Osher, S., *A variational level set approach to multiphase motion*, J. Comput. Phys. 127, 179-195 (1996).

[202] Zhao, H.-K., Osher, S., Merriman, B. and Kang, M., *Implicit nonparametric shape reconstruction from unorganized points using a variational level set method*, Comput. Vision Image Understanding 80, 295-314 (2000).

[203] Zhao, H.-K., Merriman, B., Osher, S. and Wang, C., *Capturing the behavior of bubbles and drops using the variational level set approach*, J. Comput. Phys. 143, p. 495 (1998).

Session 2

Numerical Methods and Applications

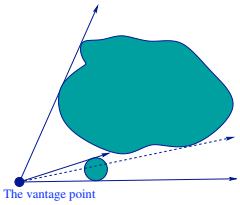
Visibility, its Dynamics, and related Variational Problems in an Implicit Framework

Yen-Hsi Richard Tsai
Institute for Advanced Study and
Department of Mathematics and PACM
Princeton University

Stanley Osher
Department of Mathematics,
UCLA

The Problem

Surfaces (occluders) in space.



Find portions of space and surfaces that are visible/invisible from a given view point.

* Find $\psi_{\mathbf{x}_o}$ such that $\{\psi_{\mathbf{x}_o} < 0\}$ describes the invisible regions

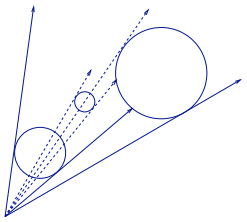
Applications: ray tracing, navigation problems, shape reconstruction, and etching.

I will discuss:

- Basic algorithms for stationary vantage points
- Visibility interpolation
- Variational problems

Our solution

Again: Find $\psi_{\mathbf{x}_o}$ such that $\{\psi_{\mathbf{x}_o} < 0\}$ describes the invisible regions

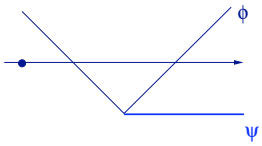


Basic algorithm: $\psi_{\mathbf{x}_o}(y) := \min_{\xi \in L(x_0,y)} \phi(\xi)$

KEY: "propagate" the visibility incrementally in radial direction, starting from the vantage point, using

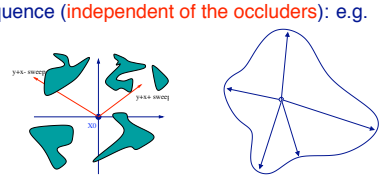
$$\psi(\mathbf{x}) = \min(\phi(x), \psi(\tilde{x})),$$

where $\tilde{x}$ a point "before" $\mathbf{x}$, depending on the grid geometry.



What we do:

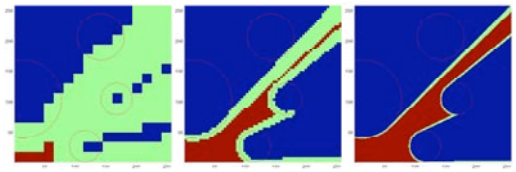
Visit the grid points in the grid either by sweeping or by a "star-shaped" updating sequence (independent of the occluders): e.g.

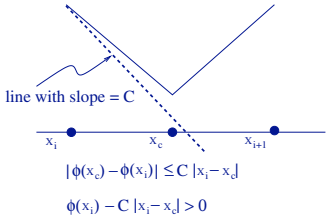


For each grid point x:

1. Solve $\nabla\psi \cdot r = 0$ at x;
2. Update: $\psi(x) = \min(\phi(x), \psi(x))$.

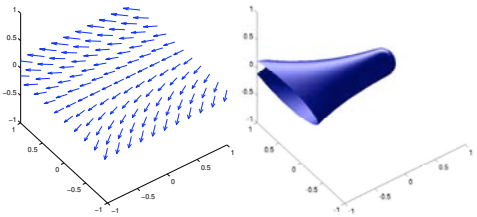
Multi-level algorithm





Skipping through large regions on which ψ does not change signs!

Visibility in bending ray fields



Solved by the fast sweeping algorithm as described in [Tsai-Cheng-Osher-Zhao 2002, to appear in SINUM].

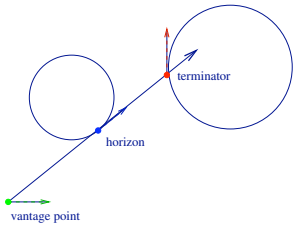
Benefits

- ✔ Algorithm independent of the topology and geometry of the occluders
- ✔ Algorithm extendable to more compicated scenarios
- ✔ Solution is Lipschitz continuous
  Shaper description of the shadow boundary using grids
  Suitable for PDE methods on the grid
- ✔ Build-in stability of the algorithm
- ✔ Reciprocity of Solution. $\psi_{\mathbf{x}_o}(y) := \min_{\xi \in L(x_0,y)} \phi(\xi)$, where $L(x_0, y)$ is the line segment connecting x_0 and y:

$$\psi_x(y) = \psi_y(x)$$

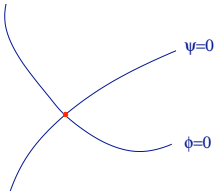
Dynamic visibility

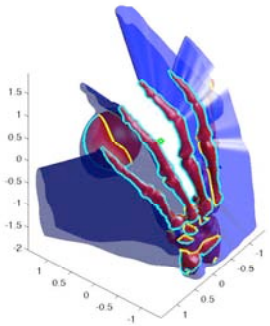
What determines the visibility when the observer is moving?
How does the shadow move?



Characterizing the horizons and terminators

We characterize the horizon and its terminator by the intersections of level set functions.





Motion laws

Horizon motion:

$$\begin{cases} \phi(\mathbf{x}(t)) = 0 \\ (\mathbf{x} - \mathbf{x_o}) \cdot \nabla\phi(\mathbf{x}) = 0 \end{cases} \tag{1}$$

$$\dot{\mathbf{x}} = \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \frac{1}{\kappa} \frac{\dot{\mathbf{x}}_\mathbf{o} \cdot n(\mathbf{x})}{|\mathbf{x} - \mathbf{x}_o|} \nu(\mathbf{x}). \tag{2}$$

$$\dot{x} = \frac{II(x - x_0)}{|II(x - x_0)|^2} \left(\dot{x}_0 \cdot \frac{\nabla\phi}{|\nabla\phi|} \right), \tag{3}$$

where II is the second fundamental form, $II = \frac{1}{|\nabla\phi|} P_{\nabla\phi} \nabla^2\phi P_{\nabla\phi}$.

Motion laws

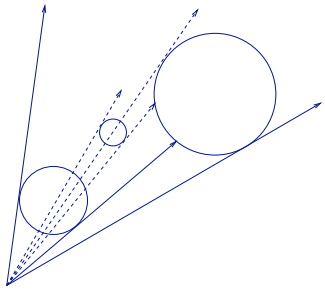
Cast horizon motion: (Let $\mathbf{y}$ be the cast of the horizon point $\mathbf{x}$)

$$\begin{cases} \phi(\mathbf{y}) = 0, \\ \frac{\mathbf{y} - \mathbf{x}_o}{|\mathbf{y} - \mathbf{x}_o|} = \frac{\mathbf{x} - \mathbf{x}_o}{|\mathbf{x} - \mathbf{x}_o|}. \end{cases} \tag{4}$$

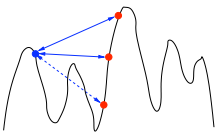
$$\dot{\mathbf{y}} = \frac{1}{\nu \cdot \mathbf{n}(\mathbf{y})} \left(\frac{|\mathbf{r}|}{|\mathbf{r}^*|} \dot{\mathbf{r}}^* \cdot (\nu^*)^\perp + \dot{\mathbf{x}}_\mathbf{o} \cdot \nu^\perp \right) \mathbf{n}^\perp(\mathbf{y}), \tag{5}$$

where $\mathbf{r} = \mathbf{y} - \mathbf{x_o}$, and $\mathbf{r}^* = \mathbf{x} - \mathbf{x_o}$, $\nu = \mathbf{r}/|\mathbf{r}|$.

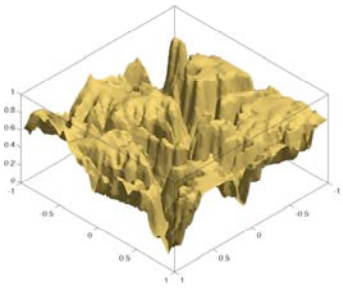
Hidden objects may emerge:

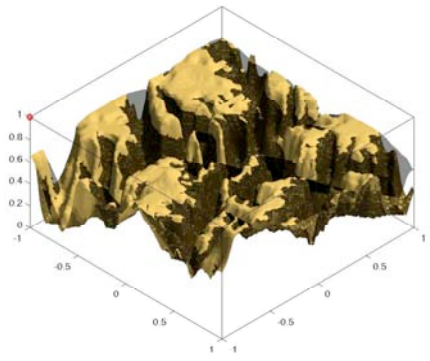


- ✔ These motion laws can be used also by the Lagrangian tracking formulations
- ✔ Prediction of objects appearance and disappearance
- ✔ Can easily be extended to morphing surfaces
- ✔ Important for illumination type problems; i.e. etching, melting ice, shape-from-shading

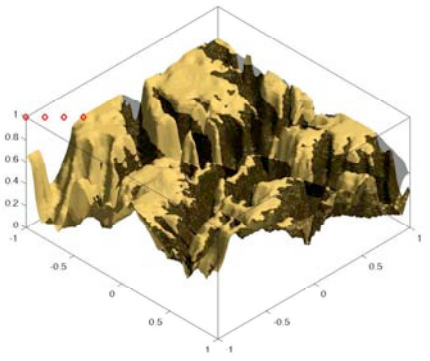


Result — Terrain fly-through

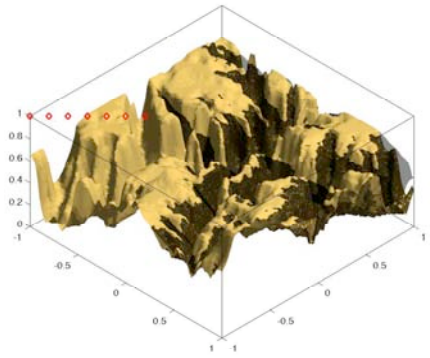




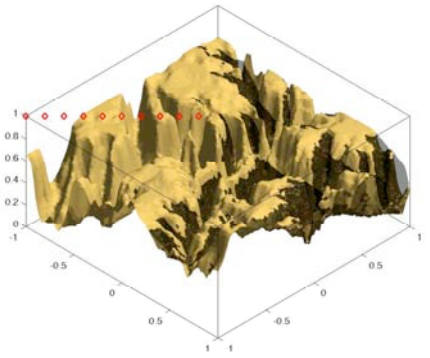
18



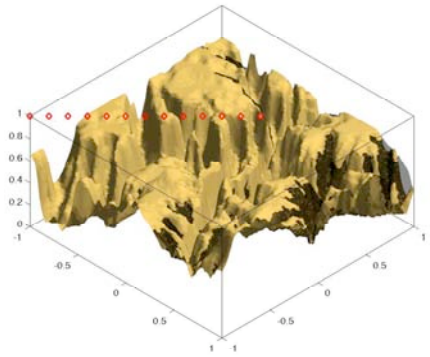
19



20

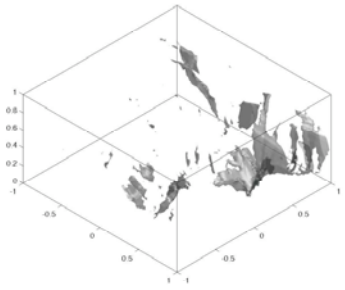


21



22

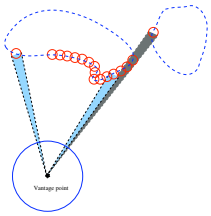
Accumulative result: invisible regions



23

Surface Interpolation of Point Clouds

Given $\{z_i\}$ sampled from the occluders. What is visible?



A piecewise constant approximation to:

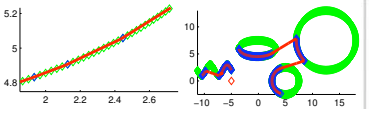
$$\rho(\mathbf{p}) = \begin{cases} \min_{\mathbf{x}\in\mathbb{R}^d}\{|\mathbf{x}-\mathbf{x_o}| : \nu(\mathbf{x_o},\mathbf{x}) = \mathbf{p}, \phi(\mathbf{x}) \le 0\} & \text{if exists} \\ \infty & \text{otherwise} \end{cases} \quad (6)$$

Reconstruction

- Filter the "occluded" points:

$$\tilde{\rho}_{\mathbf{x}_o}(\mathbf{z}) = \min\left(\rho_{\mathbf{x}_o}(z), |\mathbf{x}_o - y_i|\right), \quad \text{for every } \mathbf{z} \in \pi_{\mathbf{x}_o} B(y_i, \epsilon).$$

- Use higher order reconstruction (e.g. ENO)
- Obtain other geometrical quantities; such as k



Variational problems

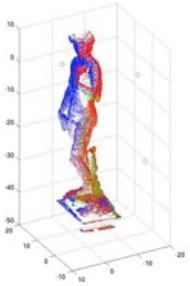
1. Variational energy depends on a fixed visibility function
   - Certain scattering problems: e.g. etching, melting-ice problems
   - Shape reconstruction from gray scale images
2. Finding the location of vantage point(s) maximizing visibility
   - The "I want to see everything" problem (complete visibility)
3. Path planning under visibility constraints: $H(\nabla u) = r(x, \phi, \psi)$. (with time dependence.)
   - The "I don't want to be seen" problem.

Working with: $\psi_x(y) = \psi(x, y)$, where x is the vantage point location.

Steoreoscopic shape reconstruction

[Jin et al.]

I want to see every thing!



Surveillance: a fly-through over DC area

(see movies)

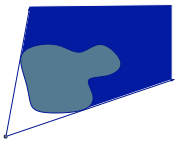
Lets try to see as much as possible.

Seeing everything translates into:

$$\bigcap_{\alpha\in A}\{\psi(x_\alpha,\cdot)\le 0\}=\{\phi\le 0\}\Longleftrightarrow \Pi_{\alpha\in A}H(\psi)=H(\phi)$$

Find x minimizing the area of invisible region outside of the occluders:

$$\min_x A(x)=\min_x\int_\Omega H(-\psi(x,y))H(\phi)dy$$



Gradient descent:

$$\frac{dx}{dt}=-\nabla_x A(x)=\int_\Omega H(\phi)\frac{\nabla_x\psi(x,y)}{|\nabla_y\psi(x,y)|}\delta(-\psi(x,y))|\nabla_y\psi(x,y)|dy.$$

Can be used to grow a path maximizing visibility.

Generalization:

$$\min_x A(x)=\min_x\int_\Omega w(\phi,\psi,\Psi,y)H(-\psi(x,y))dy,$$

$w(\phi,\psi,\Psi,y)$ is a weight function; e.g. near sighted — $w(d)$, where $d=$distance to the occluder.

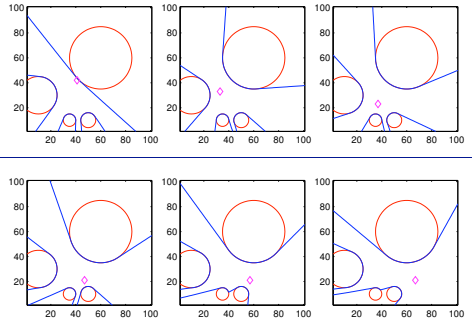
Coupled system of vantage points

Vantage points $\{x_n\}$:

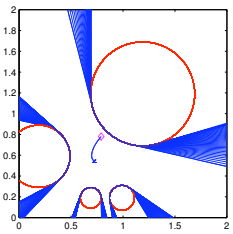
$$\min_{\mathbf{x}=\{x_n\}} A(\mathbf{x})=\min_{\mathbf{x}}\int_\Omega \omega(\mathbf{x},\phi,\cdots)\,\Pi_n H(-\psi(x_n,y))H(\phi)dy.$$

Can be used to generate a visibility maximizing path.

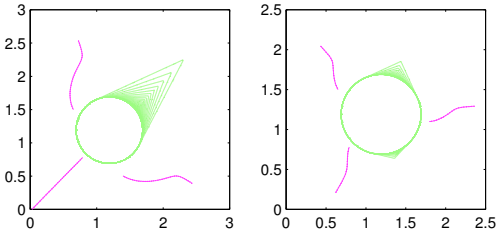
Example: extra weight on the region [50,100]x[50,100]



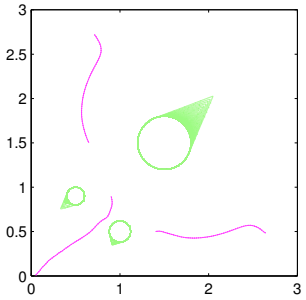
Same configuration with equal weights



Collective visibility of several vantage points



Collective visibility of several vantage points (2)



Dynamic Visibility in an Implicit Framework

Richard Tsai*†, Li-Tien Cheng‡, Paul Burchard§,
Stanley Osher*¶, and Guillermo Sapiro‖

February 2002

Abstract

We investigate the problem of determining visible regions in two or three dimensional space given a set of obstacles and a moving vantage point. This is of importance in several fields of study including rendering in computer graphics, etching in materials construction, and navigation. Our approach to this problem is through an implicit framework, where the obstacles are represented by a level set function. An efficient generic multiscale level set method is developed to generate the visible and invisible regions in space. Furthermore, we study the dynamics of shadow boundaries on the surfaces of the obstacles using special level set techniques when the vantage point moves with a given trajectory. In all of these situations, topological changes such as merging and breaking occur in the regions of interest. These are automatically handled by the level set framework here proposed. Finally, we obtain additional useful information through simple operations in the level set framework.

*Research supported by ONR N00014-97-1-0027, DARPA/NSF VIP grant NSF DMS 9615854 and ARO DAAG 55-98-1-0323

†Department of Mathematics, University of California Los Angeles, Los Angeles, California 90095, email:ytsai@math.ucla.edu

‡Department of Mathematics, UCSD, La Jolla, CA 92093-0112. Email: lcheng@math.ucsd.edu

§Department of Mathematics, University of California Los Angeles, Los Angeles, California 90095, email:burchard@math.ucla.edu

¶Department of Mathematics, University of California Los Angeles, Los Angeles, California 90095, email:sjo@math.ucla.edu

‖Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455, email: guille@ece.umn.edu. Supported by ONR,NSF, PEC ASE, and CAREER.

Contents

1 Introduction

In this paper, we consider the visibility problem described as follows: given a collection of hypersurfaces representing the surfaces of objects, called the occluders, in two or three dimensional space, determine the regions of space or on the surfaces visible to a given observer. In real world applications, this problem must be solved quickly and efficiently, preferably in real time. Generalizations of the visibility problem are just as, if not more, important, such as the case of a moving rather than static observer and the determination of regions visible for all time or invisible for all time in this situation. However, we begin with the basic visibility problem for simplicity, and address parts of the dynamic problem later on. Incidentally, the visibility problem can be reformulated into a problem of determining light and dark regions given a point light source. We occasionally consider this point of view for clarification.

Under this point of view, a more precise set of assumptions we make in the visibility problem includes a space composed of a homogeneous medium and objects with nonreflecting and nondiffracting surfaces. Furthermore, we disregard interference, assuming that the distances between objects are large compared to the wavelength of light. Under these conditions, light rays travel in straight lines and are obliterated upon contact with the surface of an object. Thus a point is called visible with respect to a vantage point, the observer, if the line segment between the point and the vantage point does not intersect any of the obstructing objects or their surfaces in space.

The need for visibility information

Even under these simplifying assumptions, the visibility problem arises as a crucial part of numerous applications in different scientific fields, including rendering, visualization [16], etching [1], the modeling of melting ice [7], surveillance, navigation, and inverse problems, to name a few. In the case of computer graphics and rendering, for example, determination of the visible portions of object surfaces allows for those portions alone to be rendered, thus saving a lot of costly computation. Additionally, there are recent variational formulations for surface reconstructions that require the solution of the visibility problem [17].

While explicit surfaces, for example triangulated surfaces, are used in a majority of computer graphics and vision applications, implicitly represented surfaces are gaining more attention. The advantages can be seen in the automatic resolution of surfaces as well as the incorporation of geometric information and the handling of various surface topologies afforded, for example, by a level set framework (see, e.g., [6, 9, 15, 16, 17, 26]). Currently there are numerous algorithms for solving

the visibility problem using explicit surface representations. For example, the work of [12] and [14] uses linearity to process triangulated surfaces. A detailed review of related work on the visibility problem, especially concerning explicit surfaces, can be found in [13]. Furthermore, there are a variety of visibility algorithms from computational geometry (see, e.g., [2, 3]). Visibility algorithms for implicit surfaces mostly consist of sending rays out from the point source and testing for intersections with the surfaces of the objects using information arising from the implicit formulation. Our proposed method for ray tracing is different. In essence, we send out rays in an implicit manner so as to propagate the causality relation of visibility. We describe this approach in more detail below.

Level set approach

We propose to solve the static and dynamic visibility problems in an implicit manner using a level set formulation. The level set method was first proposed by Osher and Sethian [20] as a PDE (Partial Differential Equation) based numerical device to capture moving interfaces. Over the years, a level set calculus has been developed that allows for the application of the level set method to a multitude of problems and situations. See, for example, the review paper [19] for an overview on the basics as well as recent advances in level set methods. We only recapitulate here that implicit PDE approaches such as the level set approach retain the important self interpolating property when propagating interfaces, thus obtaining good resolution of the interfaces. Furthermore, Boolean operations on sets, including finding curves of intersections and the trimming commonly required in CAD (Computer Aided Design) are easy to implement in a level set framework.

In our case of visibility, a real valued two or three dimensional function ϕ, called the level set function, is introduced. The zero level set of this function represents the surfaces of the occluding objects. Furthermore, we require that the points where ϕ is negative represent the interior of the objects. Several algorithms have been developed in the literature to efficiently obtain this representation. A level set method for visibility will use this function ϕ whenever the objects are considered.

One idea in determining whether a point is visible to a given observer is to compare the geodesic and Euclidean distances between the observer and that point. See [24] for an example of this approach. The geodesic distance between two points is the distance in the space in the presence of obstacles, namely the objects. Let $\mathbf{x}$ represent the point of interest and $\mathbf{x_o}$ represent the observer point. The geodesic distance can thus be calculated by solving the Eikonal equation

$$H(\phi)|\nabla u| = 1,$$

where H is the one dimensional Heaviside function, with condition $u(\mathbf{x_o}) = 0$. Thus the point $\mathbf{x}$ is occluded if and only if

$$u(\mathbf{x}) > |\mathbf{x} - \mathbf{x_o}|.$$

However, this algorithm is at best $O(N \log N)$, where N is the number of grid points (see, e.g., [25]). This may not be optimal, making it too slow for applications requiring real time computations. Furthermore, numerical implementation of the Heaviside function may cause problems for accuracy.

Our level set approach

We present here a few level set based algorithms for determining various types of visibility information in any dimension, though three dimensions is probably the one of interest. We first introduce a multiresolution algorithm for solving the visibility problem for a given fixed vantage point. This algorithm constructs the occlusion boundary, the interface separating visible from invisible. At each resolution level, we solve a radially defined causality relation on a given grid in one pass, obtaining not only a conservative estimate of the visible and invisible regions but a locally second order approximation of the occlusion boundary. In addition, our algorithm is independent of both the convexity of the occluders and the grid geometry, and its parallelization is straightforward.

The level set framework is especially important as it handles occluder fusion, where the occlusion boundary merges during the construction process. Furthermore, the implicit representation, though not as effective on occluders which are open surfaces, can still handle this case by considering them as very thin hypersurfaces.

Dynamic visibility

In the second part of the paper, we extend our study to the dynamic visibility problem. In this case, we consider a moving vantage point. Obviously the static visibility problem can be applied at each time to solve this problem, and our algorithm can be used to solve it efficiently enough. However, this static approach does not give us other useful information about the dynamics; for instace, how fast a point in space will become visible or invisible. In many cases, the problem can be solved even faster if the visibility at a previous time is used effectively to produce visibility at future time.

Thus we study the dynamics of curves on the occluders that separate light and dark regions on the occluders. The curves in fact can be represented using a level set approach, following the work of [5, 8, 10]. We also study other types of curves,

called horizons, and their motions which form a superset of the curves separating light from dark but can be constructed quickly. We rigorously derive motion laws for all these types of curves and evolve them under the level set framework. This framework allows for topological changes which may occur in the curves and its self interpolating property automatically produces well resolved results. Finally, we derive an emergence-time estimate to predict an occluded object's emergence into view. Thus our work complements the book of Cipolla and Giblin [11] which discusses the reconstruction of shape from the perspective (orthogonal) projection of the horizons. This is exactly what we are trying to evolve.

Notations

Through out this paper, we use the following notation:

- The space in which we work will be $\mathbb{R}^d$, where $d = 2$ or 3.
- $\mathbf{x_o}$ denotes the position of the vantage point, or observer. We further assume that $\mathbf{x_o}$ never lies in the interior of the objects.
- Ω is a set of connected domains whose closure denotes the objects in question. Furthermore, let $\Gamma = \partial\Omega$.
- ϕ denotes the level set function representing the objects of interest. We may further assume that ϕ is the signed distance function to Γ. This particular level set function can be efficiently computed using fast algorithms such as the fast marching method of [25] or fast sweeping methods.
- We define the view direction vector pointing from $\mathbf{x_o}$ to $\mathbf{x}$ by $\nu(\mathbf{x_o}, \mathbf{x}) = (\mathbf{x} - \mathbf{x_o})/|\mathbf{x} - \mathbf{x_o}|$. When the context is clear, we will drop the arguments and write simply $\nu(\mathbf{x})$ or ν.
- Let $\mathbf{x_1}$ and $\mathbf{x_2}$ denote two points in space. We say $\mathbf{x_1} \preceq \mathbf{x_2}$ ($\mathbf{x_1}$ is "before" $\mathbf{x_2}$) if the conditions $\nu(\mathbf{x_o}, \mathbf{x_1}) = \nu(\mathbf{x_o}, \mathbf{x_2})$ and $|\mathbf{x_1} - \mathbf{x_o}| \leq |\mathbf{x_2} - \mathbf{x_o}|$ are satisfied. We also define the strict relation $\prec$ if the condition $|\mathbf{x_1} - \mathbf{x_o}| \leq |\mathbf{x_2} - \mathbf{x_o}|$ above is replaced by $|\mathbf{x_1} - \mathbf{x_o}| < |\mathbf{x_2} - \mathbf{x_o}|$.
- A point $\mathbf{y} \in \Gamma$ is called a horizon point if and only if $\nu(\mathbf{x_o}, \mathbf{y}) \cdot \mathbf{n}(\mathbf{y}) = 0$, where $\mathbf{n}(\mathbf{y})$ is the outer normal of Γ at $\mathbf{y}$. The horizon thus refers to the set of horizon points.
- A point $\mathbf{y} \in \Gamma$ is a cast horizon point if and only if there is a point $\mathbf{y}^*$ such that: 1) $\mathbf{y}^* \prec \mathbf{y}$ and 2) $\mathbf{y}^*$ is a horizon point. The cast horizon thus refers to the set of cast horizon points.

- The visible contour refers to the set of visible points of the horizons and cast horizons.

2 Implicit ray tracing

We now set up the foundation of our approach and derive properties of ray tracing of a single point source in an implicit framework. The motivation is as follows: we observe that the visibility status of points sharing the same radial direction centered at the vantage point satisfy a causality condition. This means if a point is occluded, then all other points farther away from the vantage point in the same radial direction are also occluded, i.e., if $\mathbf{x_1}$ is occluded and $\mathbf{x_1} \preceq \mathbf{x_2}$, then $\mathbf{x_2}$ is also occluded.

This fact can be described more rigorously as follows. Define

$$\rho(\mathbf{p}) = \begin{cases} \min_{\mathbf{x}\in\mathbb{R}^d}\{|\mathbf{x}-\mathbf{x_o}| : \nu(\mathbf{x_o},\mathbf{x}) = \mathbf{p}, \phi(\mathbf{x}) \le 0\} & \text{if exists} \\ \infty & \text{otherwise,} \end{cases} \tag{1}$$

giving the distance between $\mathbf{x_o}$ and the closest point on Γ in the direction of $\mathbf{p}$ from $\mathbf{x_o}$. Thus a given point $\mathbf{x}$ is invisible if $\rho(\nu(\mathbf{x},\mathbf{x_o})) \le |\mathbf{x}-\mathbf{x_o}|$. Refer to Figure 1 for an example and clarification. Therefore, we can define the visibility indicator

$$\Xi(\mathbf{x},\mathbf{x_o}) := \rho(\nu(\mathbf{x},\mathbf{x_o})) - |\mathbf{x}-\mathbf{x_o}|,$$

so that $\{\Xi \ge 0\}$ is the set of visible regions in $\mathbb{R}^d$ and $\{\Xi < 0\}$ is the set of occluded regions.

From another view point, the problem becomes: compute

$$\psi(\mathbf{x}) := \min_{\xi\in L(\mathbf{x_o},\mathbf{x})} \phi(\xi),$$

where $L(\mathbf{x_o},\mathbf{x})$ is the line segment connecting $\mathbf{x_o}$ and $\mathbf{x}$. Thus if $\psi(\mathbf{x})$ is negative, then $\mathbf{x}$ is occluded. In fact, our implicit ray tracing algorithm is is an approximation of this formula.

Implicit formulation

Our implicit framework encodes visibility information in a Lipschitz continuous function ψ so that a point $\mathbf{y}$ is visible if $\psi(\mathbf{y}) \ge 0$ and invisible if $\psi(\mathbf{y}) < 0$. Thus we can compute the value of $\psi(\mathbf{x})$ by

$$\psi(\mathbf{x}) = \min(\psi(\mathbf{x}'), \phi(\mathbf{x})) \tag{2}$$

where $\mathbf{x}'$ is some point "right before" $\mathbf{x}$ in the ray direction. We can therefore start from the vantage point $\mathbf{x_o}$ and update the grid points following the ray directions outwards. A simple algorithm for this reads:

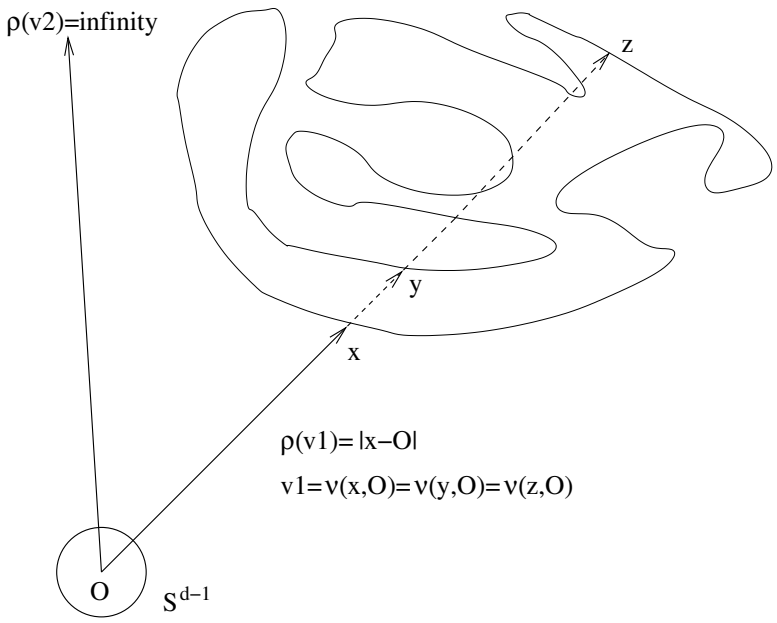


Figure 1: Illustration of ρ

1. Set $\psi(\mathbf{x_o}) = \phi(\mathbf{x_o})$.
2. Do a star-shaped[1] updating sequence on the grid.
3. For each grid point $\mathbf{x}$, choose $\mathbf{x}'$ depending on the grid geometry.
4. Compute the value of $\psi(\mathbf{x})$ via (2).

Each grid node is visited in a specified order that maintains the causality. As long as the updated grid nodes form a star-shaped region centered at the vantage point, causality is maintained. See the Appendix for an example of such an updating method. Due to the minimization and the linear interpolation (see Section 5.1) used to find $\mathbf{x}'$, the algorithm is l_∞-stable. More precisely, we have

$$\phi_{\text{int}}(\mathbf{x}') \leq \max\{|\phi(x_l)| : x_l \text{ are the points used in the interpolation}\},$$

where ϕ_{int} is the interpolant we constructed. Please see Section 5.1. Figure 2 shows what ψ should look like in a one space dimension setting.

[1] See 5.2

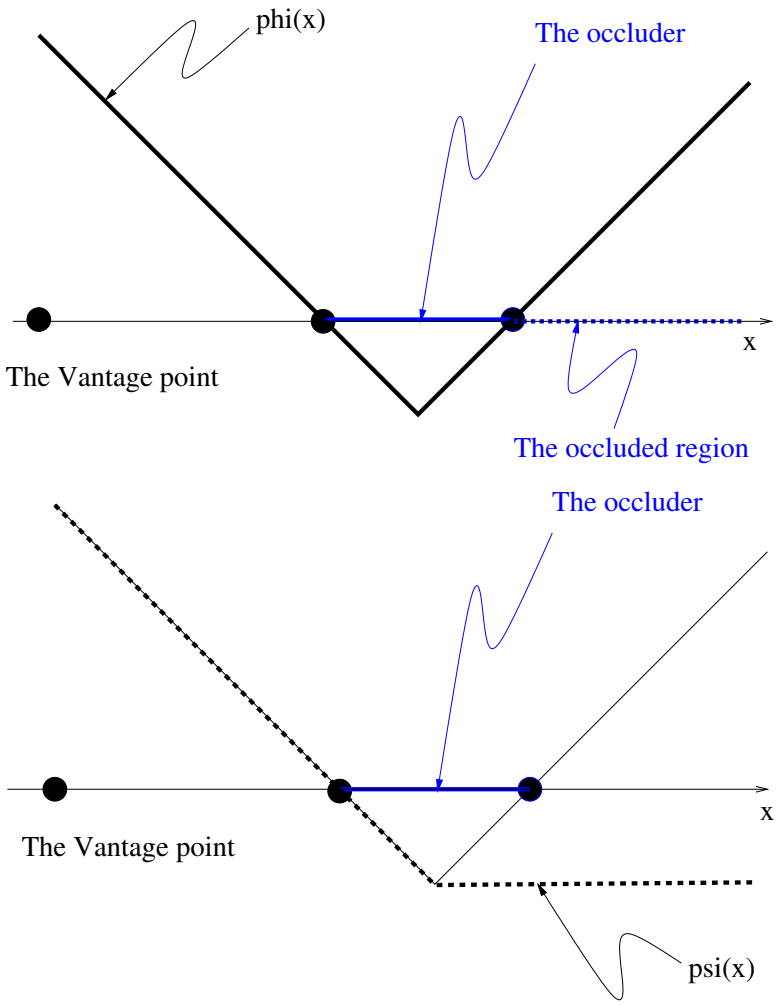


Figure 2: A demonstration of the motivation of our implicit ray tracing algorithm in one dimension.

Volumetric visibility processing

If we want to determine volumetric visibility information, i.e., we want to find regions that are occluded from a given rectangular region (view cell), the above algorithm can be modified for this purpose. For simplicity, assume that the view cell degenerates to a line with end points $\mathbf{x}_1$ and $\mathbf{x}_2$. At each point $\mathbf{x}$, we compute $\mathbf{x}'_1$ and $\mathbf{x}'_2$ as in step 2 with respect to $\mathbf{x}_1$ and $\mathbf{x}_2$. The update formula then becomes: $\psi(\mathbf{x}) = \min(\phi(\mathbf{x}), \max(\psi(\mathbf{x}'_1), \psi(\mathbf{x}'_2)))$.

2.1 Multi-resolution calculation

Finding inside/outside

Any multi-resolution approach of the visibility problem requires the skipping of large regions which we know a priori are either visible or invisible. This hinges upon the ability to determine whether any given voxel is completely "inside" or "outside" of the objects. This can be done conservatively with the help of the Lipschitz constant of the embedding level set function.

Let C be the Lipschitz constant of ϕ. Let $\mathbf{x}_c$ be the center point and $\mathbf{x}_i$ the vertices of the given voxel V. If

$$\phi(\mathbf{x}_i) + C|\mathbf{x}_c - \mathbf{x}_i| < 0 \ \ \forall\, i, \tag{3}$$

then we know $\phi|_V < 0$ ($V \subset \{\phi < 0\}$). Conversely, if

$$\phi(\mathbf{x}_i) + C|\mathbf{x}_c - \mathbf{x}_i| > 0 \ \ \forall\, i, \tag{4}$$

then we know $\phi|_V > 0$. Since our embedding function is the signed distance function, the Lipschitz constant $C = 1$.

Correspondingly, we also have:

$$\phi|_V < 0 \ \text{ if } \ \phi(\mathbf{x}_c) + C|\mathbf{x}_i - \mathbf{x}_c| < 0 \ \ \forall\, i,$$

and

$$\phi|_V > 0 \ \text{ if } \ \phi(\mathbf{x}_c) + C|\mathbf{x}_i - \mathbf{x}_c| > 0 \ \ \forall\, i.$$

A voxel V is occluded if it lies completely inside the an object or if it is "behind" an occluded voxel $\tilde{V}$.This idea can be implemented by a careful reinterpretation of formula (2). A similar condition for determining completely visible voxels can also be easily derived.

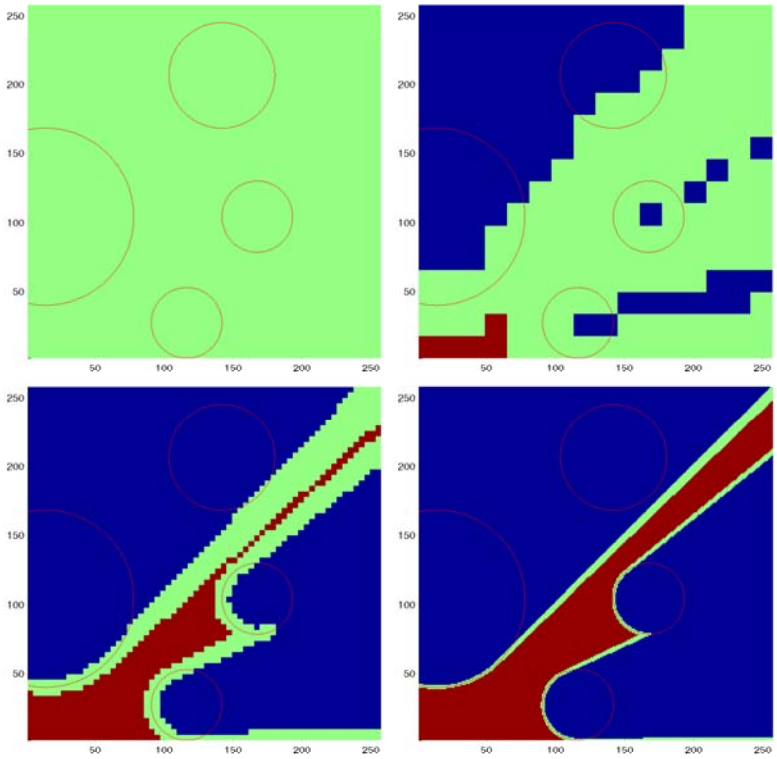


Figure 3: This is a schematic diagram for the multi-resolution algorithm. Occluded voxels are depicted in blue and visible ones in red. The regions are target for next level refinement.to be refine. The red curves represent the boundaries of the occluders, and the vantage point is positioned at $(1, 1)$. The sizes of the voxels are: 64×64, 16×16,4×4, and 1×1.

2.2 A Multi-resolution algorithm

Let h give the resolution of our grid such that smaller h means high resolution (i.e. finer grid). In practice, we can simply set h to be the mesh size. Under resolution h, we use (i^h, j^h, k^h) to denote the grid indices and V^h for a voxel in the grid. We will drop the superscripts of these indices when the context is clear.

Algorithm: (Multi-resolution visibility sweeping)

Let $\{h_l : h_l > h_{l+1}, l = 0, \cdots, m\}$ be the set of resolutions of interest. For each resolution level h, decending from h_0 to h_m,

1. Compute ψ on the (i^h, j^h, k^h) which do not lie on a voxel marked either visible or occluded.

2. For each voxel V^h which does not lie on a voxel marked either visible or occluded, mark V^h to be visible or occluded according to formulas (3) and (4).

We remark that the Lipschitz constant of the linearly interpolated ψ is can be taken from that of ϕ. Therefore, Step 2 above is well defined. Please see Figure 3 for a demonstration of this algorithm.

As for complexity, the multiresolution algorithm for constructing visiblity information should be an $O(N^{d-1} \log N)$ algorithm in terms of speed. Here $N = 1/h$ where h is the smallest spatial stepsize used in the multiresolution framework and d is the dimension of the space. The N^{d-1} part of the complexity comes from the fact that a codimension one hypersurface in d dimensional space is being generated under fast sweeping and the $\log N$ part comes from multiresolution. The memory allocation of our algorithm is also $O(N^{d-1} \log N)$, with the $\log N$ part once again due to multiresolution. In practice, our algorithms have proven to be very fast, obtaining detailed visiblity information in almost real-time. For example, it takes less than a second to perform this algorithm running on only one resolution (meaning no multi-resolution), on a grid with 100^3 cells, on a moderate PC.

2.3 Multi-scale considerations

If we consider the visibility problem in applications related to human vision, such as 3D virtual environment rendering, it is natural to put a scale parameter into the size of the objects related to the distance of the object from the vantage point. We want to ignore certain **isolated and small** objects that are **far away** from the vantage point using this information. It is important to notice that a collection of closely positioned small objects can form a visible ensemble, seen for example in clouds and trees.

Using the level set representation of the virtual environment in conjunction with PDEs, we are able to deal with this issue easily without explicitly considering each object separately. The idea is to *dilate* the interface first so that small objects can merge to form ensembles of larger size. We then *shrink* the interfaces (one possibility is to perform curvature driven motion) such that remaining small objects will disappear. This approach follows the regularization effect of viscosity solution theory for Hamilton-Jacobi Equations. It is basic mathematical morphology, and can be done easily, see e.g. [4][23].

3 Dynamic visibility

We now consider the case in which the vantage point is moving. Naturally visibility information changes according to the position of the vantage point. We are interested in how visibility changes and when hidden objects become visible. This amounts to studying how the boundaries between visible and invisible regions move with respect to the vantage point motion. We first remark that these boundaries are hypersurfaces in regions outside of the occluders, and that the Gaussian curvature on such surfaces is 0.

For a single convex object, the horizon determines the visibility information on the surface. Therefore, tracking the motion of the horizon for all time gives us incremental information on the change of the visible portion of the object.

We formulate the visibility problem so that the points which are on the boundaries of the visible regions on the surfaces can easily be identified. The dynamics of these points are derived so that one can track the visible regions according to the motion of the vantage point $\mathbf{x_o}$.

The points forming the boundaries of visible regions on given surfaces can be placed into two categories:

- points that are part of the horizon;
- points that border *shadows cast by some surface* (cast horizon).

Thus, two types of motions need to be investigated, namely, that of the horizon and of the cast horizon. The motion of the horizon is characterized by the orthogonality constraint and it, in turn, becomes a part of the constraints of the cast horizon motion.

In a level set formulation, we want to create a level set function whose zero level set captures the points described above. We need a description relating each point on the cast horizon to a point on the horizon of the surface casting the shadow.

Furthermore, our description should be global, that is, quantities should vary "continuously" with respect to points not on the surface. This requirement is essential for the success of our level set formulation.

Assuming that there is no singularity in the velocity field of the horizon or cast horizon motion, and there are no other considerations, the level set approach of tracking the visible contours is optimal. With the fast sweeping algorithms and local storage strategies, the complexity of the level set approach to track the horizon and cast horizon curves is formally $\mathcal{O}(N)$ in operation counts and in storage. Here, the number N is the number of points used to resolve the curves. In actual applications, there are other aspects that affects the overall complexity of this approach. We will address this point in a later subsection.

3.1 Finding the horizon and the cast horizon implicitly

Horizons and cast horizons are objects of codimension 2. We may therefore categorize these objects by the intersection of the zeros of two level set functions. Furthermore, for numerical reasons, we want the two zero level sets to be more or less orthogonal to each other near their intersection.

Finding the horizon

We extend the orthogonality condition that defines the horizon and arrive at

$$\hbar(\mathbf{x}, t) = (\mathbf{x} - \mathbf{x}_0) \cdot \nabla\phi(\mathbf{x}). \tag{5}$$

$\hbar$ determines the visibility of any convex object embedded in ϕ :

$$\{\hbar(\mathbf{x}) \leq 0\} \bigcap \{\phi = 0\} \iff \text{visible.}$$

In general cases, where there are multiple objects (convex and nonconvex), $\hbar$ does not give exact visibility information anymore. It just provides local visibility information just as local extrema may not be absolute extrema. Thus clearly, for objects hiding completely behind other objects, $\hbar$ will still be non-negative on the parts of the surface facing the source. Instead, $\hbar$ gives a conservative "estimate" of the shadow:

$$\{\hbar(\mathbf{x}) > 0\} \bigcap \{\phi = 0\} \implies \text{invisible.}$$

Thus the visible horizon is a subset of $\{\phi = 0\} \bigcap \{\hbar = 0\} \bigcap \{\psi \geq 0\}$, where ψ is the visibility function coming from our static algorithm. Figure 4 gives an example of horizons found this way.

Finding the cast horizon

How do we find the cast horizon? The idea is to overshoot the shadow boundaries generated by the visible horizon when it hits another part of Γ, creating a level set piece $\tilde{\psi}$ in that neighborhood, and then propagate ψ as usual. $\{\tilde{\psi} = 0\}$ will cut through Γ on the cast horizon, therefore providing an implicit representation of it. We can even make $\{\tilde{\psi} = 0\}$ perpendicular to Γ locally around the intersection by iterating on the following PDE used in [18]:

$$\tilde{\psi}_\tau + \text{sgn } (\phi)\nabla\tilde{\psi} \cdot \frac{\nabla\phi}{|\nabla\phi|} = 0.$$

A more direct approach is to define $\tilde{\psi}$ on a grid point to be the "upwind" value of ϕ. This will introduce an overshoot of the size of a mesh size. Alternatively, we notice that the occlusion generated by the set $\{\hbar \geq 0\}\bigcap\{\phi \leq 0\}$ is the same[2] as $\{\phi \leq 0\}$. Therefore, we can define $\tilde{\psi}$ from the result of our algorithm under the configuration $\{\hbar \geq 0\}\bigcap\{\phi \leq 0\}$. Figure 4 shows the operations described above in a simple two circle setting.

With these characterizations and the visibility result, we can easily identify the visible contours. See Figures 5, 6, and Figures 7, 8[3] for examples. In these figures, the visible portions of the horizons and the cast horizons are depicted as cyan and yellow curves respectively. A green circle is drawn to reveal the location of the vantage point in each setting. The boundaries between visible and invisible regions are represented by blue surfaces. We observe that the blue surfaces cut through the objects exactly at the visible contours.

3.2 The dynamics of the horizon

Let $\mathbf{x}_o(t)$ be the position of the vantage point and $\mathbf{x}(t)$ be a corresponding point on the horizon at time t. We first consider a single convex occluder Ω embedded by the signed distance function ϕ. Let $n(x)$ denote the outer normal of $\partial\Omega$ at $\mathbf{x}$. This translates into the following constraints on $\mathbf{x}(t)$:

$$\begin{cases} \phi(\mathbf{x}(t)) = 0 \\ (\mathbf{x} - \mathbf{x_o}) \cdot \nabla\phi(\mathbf{x}) = 0 \end{cases} \tag{6}$$

In two dimensions, we can invert the above constraints and derive that

$$\dot{\mathbf{x}} = \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \frac{1}{\kappa}\frac{\dot{\mathbf{x}}_\mathbf{o} \cdot n(\mathbf{x})}{|\mathbf{x} - \mathbf{x}_o|}\nu(\mathbf{x}). \tag{7}$$

[2] modulo a small subset of $\{\phi \leq 0\}$, which we know is invisible by definition.

[3] The terrain data is obtained from `ftp://ftp.research.microsoft.com/users/hhoppe/data/gcanyon/.`

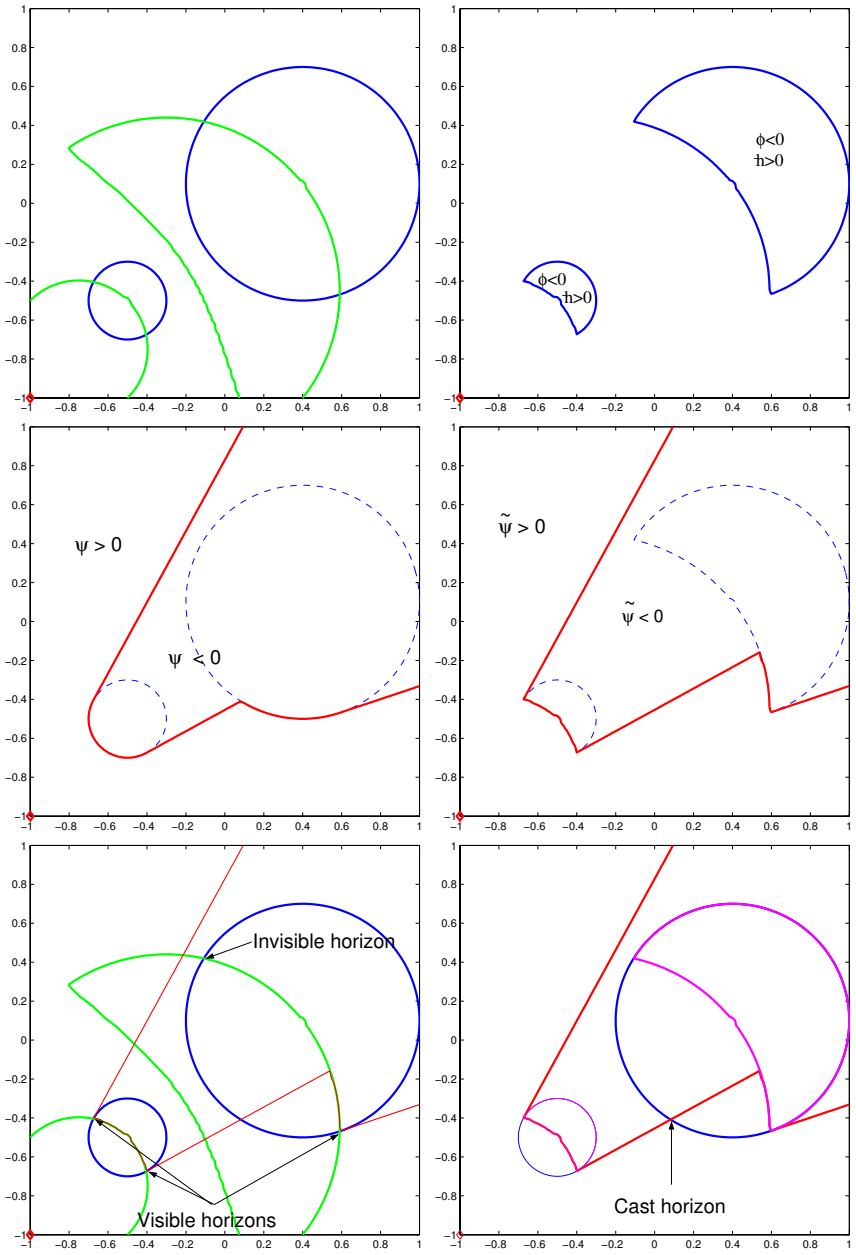


Figure 4: Finding the visible horizons and their casts. The occluders are the two circles depicted by the blue curves, and the vantage point is located at $(-1, -1)$. The green curves are the zero level set of $\hbar$. Visible horizons and their casts are characterized by the intersections of different level set functions as described in the text.

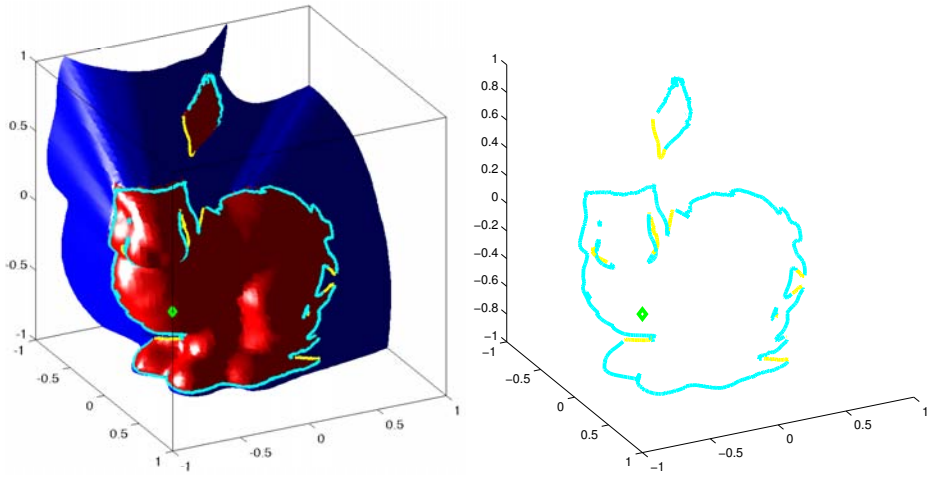


Figure 5: Visible contour (portions of horizon and cast horizon that are visible)

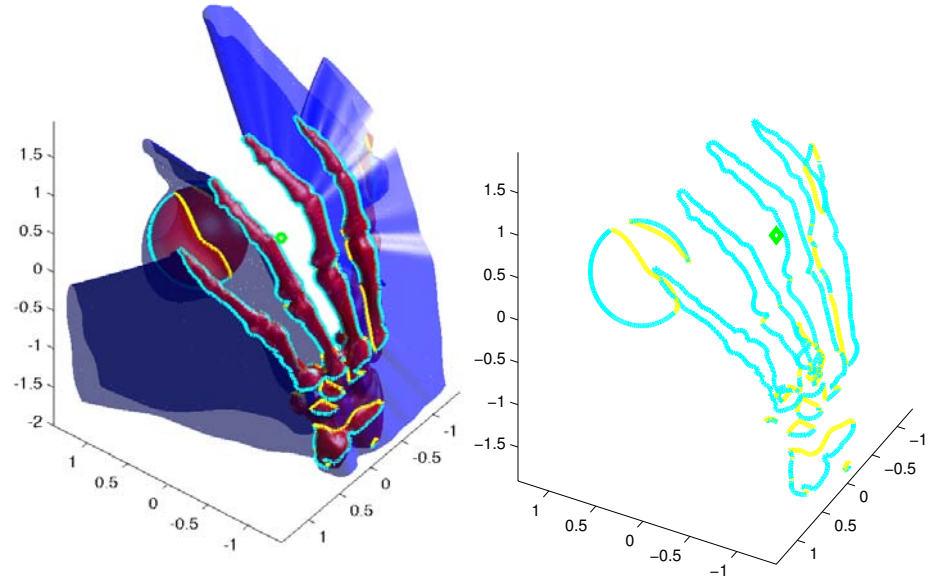


Figure 6: Visible contour (portions of horizon and cast horizon that are visible)

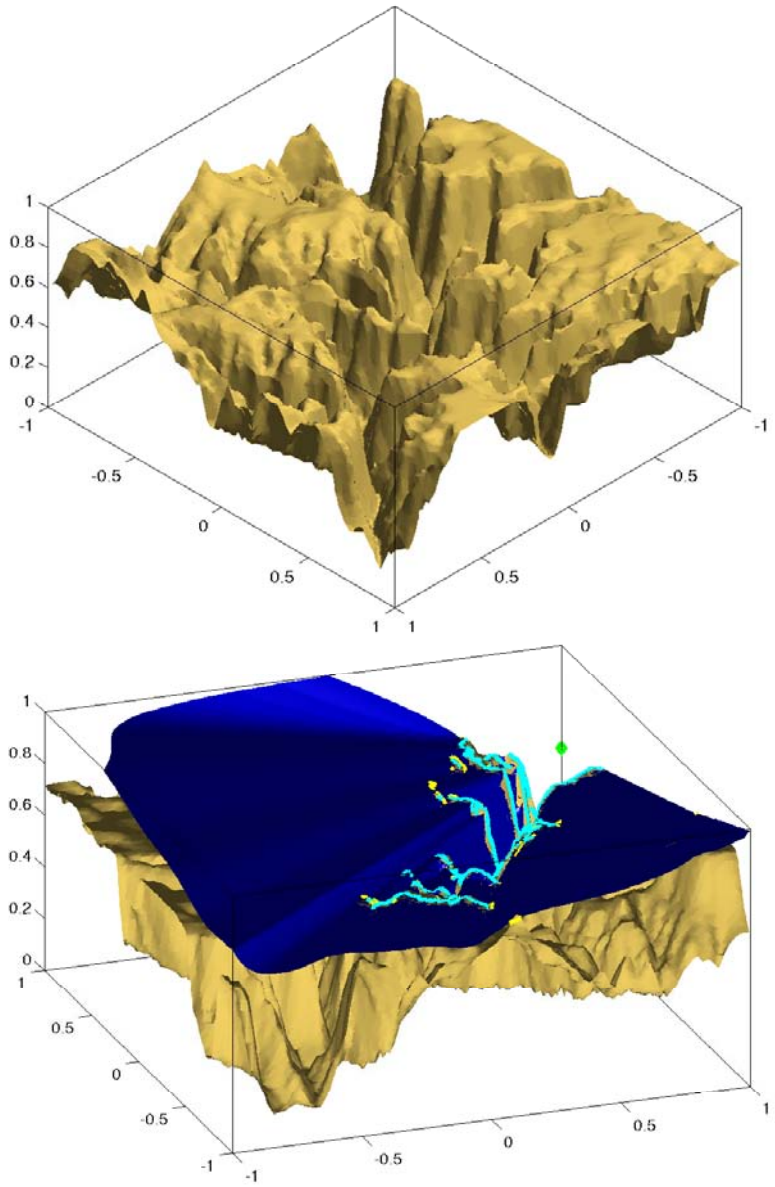


Figure 7: Horizons and cast horizons obtained from the elevation data of Grand Canyon.

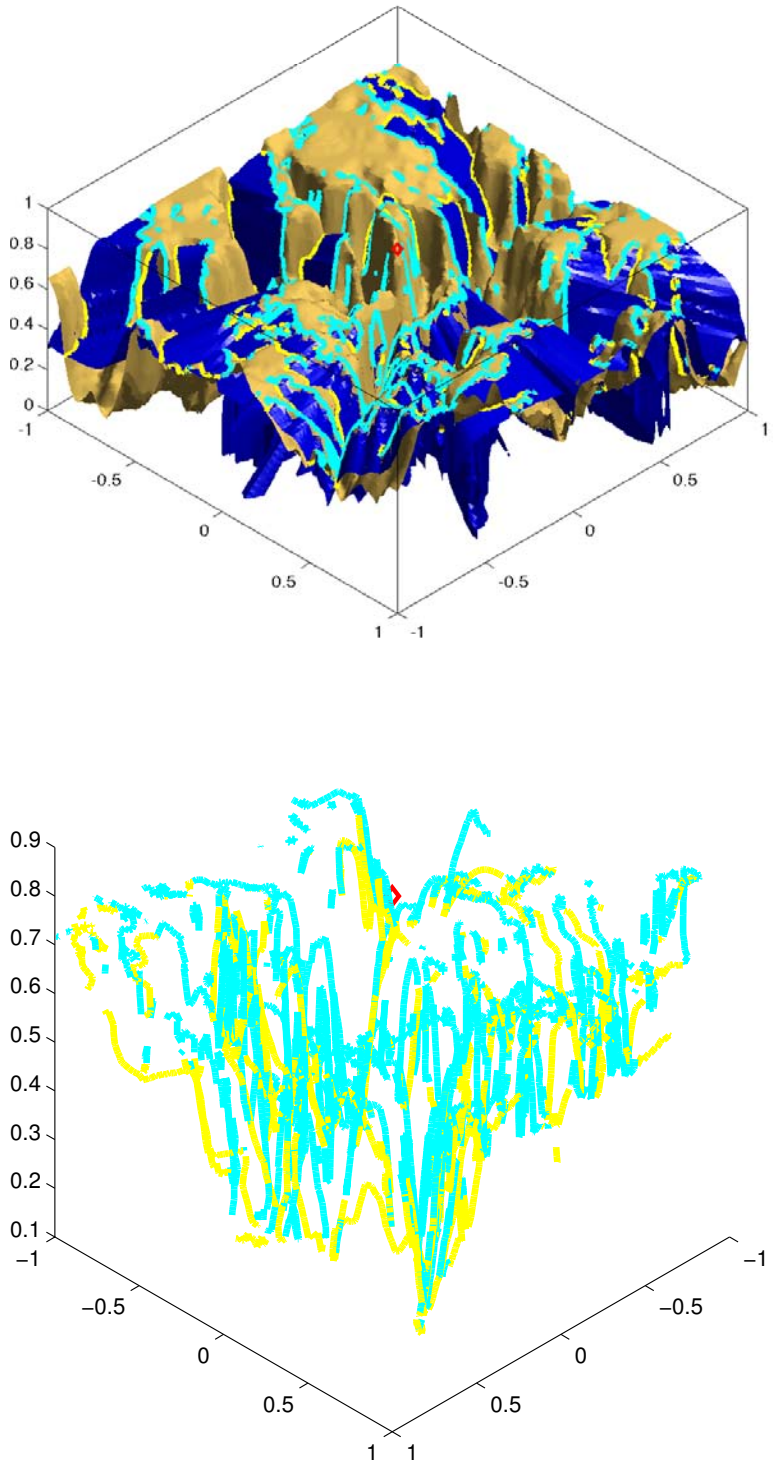


Figure 8: Horizons and cast horizons obtained from the elevation data of Grand Canyon.

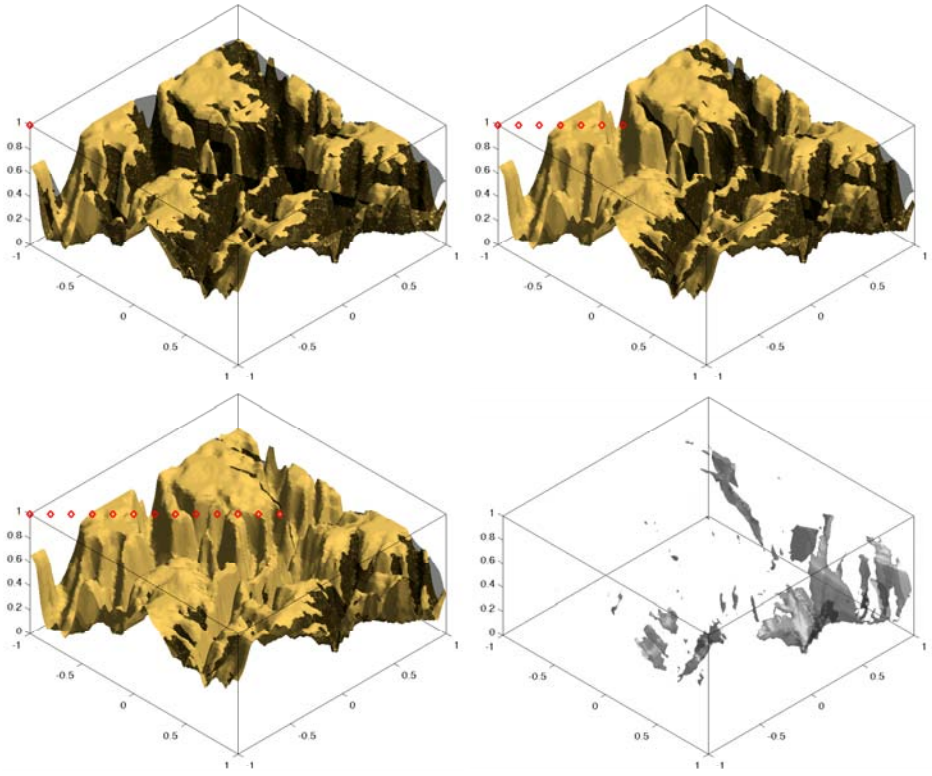


Figure 9: By taking the intersection of the occlusion during a trajectory of the observer, we can find the cumulative occlusion easily and efficiently. The following pictures show a progression of the cumulative occlusion subject to an observer ("spy plane") moving across a region of Grand Canyon.

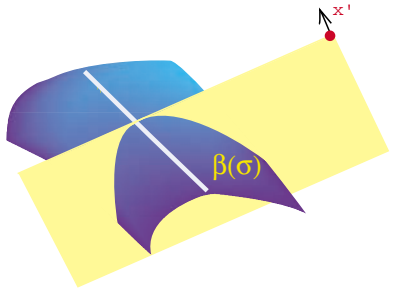


Figure 10:

Here, κ is the curvature of the occluding surface at $\mathbf{x}$. In three space dimensions, the horizon becomes a closed curve $\Gamma(s) = \mathbf{x}(s,t)$, where s is the arc length of $\Gamma(s)$. Let P be the plane tangent to $\dot{x}_o$, passing through $\Gamma(s)$ and x_o. Let $\beta(\sigma)$ be the curve on the intersection of P and $\partial\Omega$. Then, locally at t and x, we have a two dimensional visibility problem on the plane P, in which $\beta(\sigma)$ defines the boundary of the objects. Following this reasoning, κ should naturally be taken from $\beta(\sigma)$. See Figure 10.

Alternatively and more naturally under our level set formulation, we rederive the above motion law as

$$\dot{x} = \frac{II(x-x_0)}{|II(x-x_0)|^2}\left(\dot{x}_0 \cdot \frac{\nabla\phi}{|\nabla\phi|}\right), \tag{8}$$

where II is the *second fundamental form*, which can conveniently be extended to the other level sets and takes the form:

$$II = \frac{1}{|\nabla\phi|} P_{\nabla\phi} \nabla^2\phi P_{\nabla\phi}.$$

Here, $P_{\nabla\phi}$ is the orthogonal projection matrix projecting vectors to the plane with normal vector parallel to $\nabla\phi$.

For a detailed derivation and implementation, please see sections 5.4, 5.3 and 5.7. Figure 11 shows a result of horizon motion on a nonconvex body.[4]

[4]For more exmaples on the horizon motion, please see `http://www.math.ucla.edu/ ytsai/math_page`

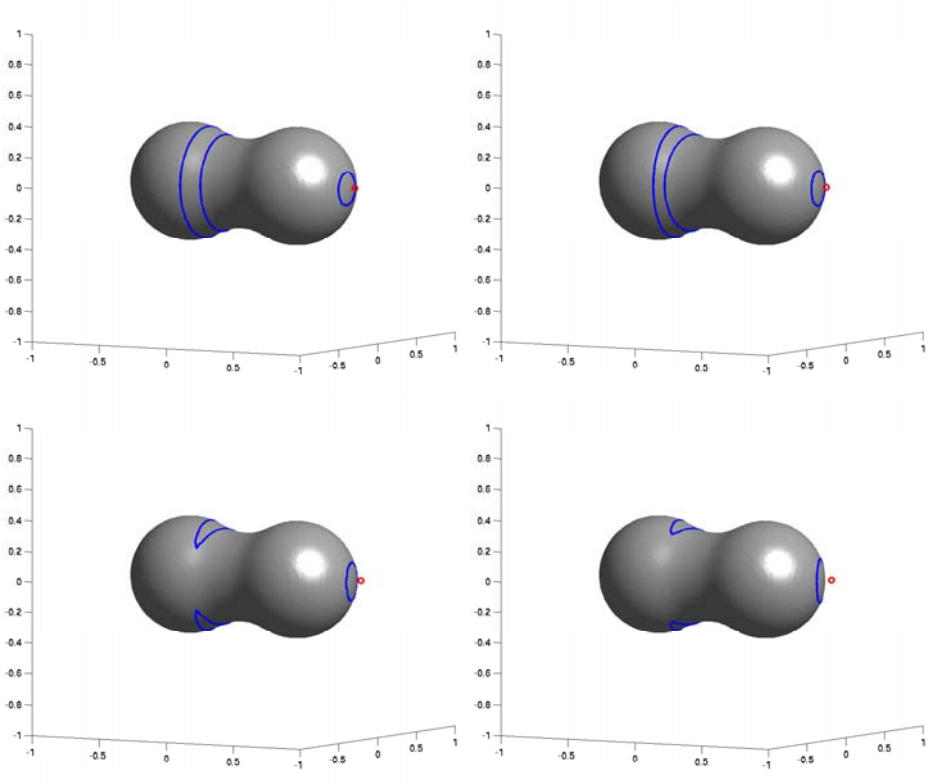


Figure 11: A example of moving horizon around a nonconvex occluder. Observe that the horizon curves break and change topology.

3.3 The dynamics of the cast horizon

Assume that $\mathbf{x}$ is a cast horizon point and $\mathbf{x}^*(\mathbf{x})$ is its generator. In two dimensions, the motion of $\mathbf{x}$ is determined by the following constraints:

$$\begin{cases} \phi(\mathbf{x}) = 0, \\ \frac{\mathbf{x}-\mathbf{x}_o}{|\mathbf{x}-\mathbf{x}_o|} = \frac{\mathbf{x}^*-\mathbf{x}_o}{|\mathbf{x}^*-\mathbf{x}_o|}. \end{cases} \tag{9}$$

Inverting, we find that the motion of the cast horizon can be written as follows:

$$\dot{\mathbf{x}} = \frac{1}{\nu \cdot \mathbf{n}(\mathbf{x})}\left(\frac{|\mathbf{r}|}{|\mathbf{r}^*|}\dot{\mathbf{r}}^* \cdot (\nu^*)^\perp + \dot{\mathbf{x}}_\mathbf{o} \cdot \nu^\perp\right)\mathbf{n}^\perp(\mathbf{x}), \tag{10}$$

where

$$\mathbf{n}^\perp(\mathbf{x}) := \begin{pmatrix} \phi_{x_2}(\mathbf{x}) \\ -\phi_{x_1}(\mathbf{x}) \end{pmatrix} / |\nabla\phi(\mathbf{x})|,$$

and similarly for $\nu^\perp$. See Section 5.5 for a detailed derivation. See Figure 12 for a computational result using this formula. We notice that these constraints also tell us how the shadow boundaries should move.

In three dimensions, we can reduce the instantaneous motion to a two dimensional problem on the "right" section of the surface following the reasoning given in the previous subsection.

Motions of the shadow boundaries

How does the shadow move in space? We can constrain a point on the shadow boundary to move only normal to the viewing direction (ergo, the shadow boundary):

$$\begin{cases} \mathbf{x}' \cdot \nu = 0, \\ \frac{\mathbf{x}-\mathbf{x}_o}{|\mathbf{x}-\mathbf{x}_o|} = \frac{\mathbf{x}^*-\mathbf{x}_o}{|\mathbf{x}^*-\mathbf{x}_o|}. \end{cases} \tag{11}$$

Motions of horizons and cast horizons of dynamic surfaces

We remark that we we are able to derive the motion laws of the visible contours even when the occluders are changing shapes. In this case, the embedding level set function ϕ is a function of space and time, $\phi(\mathbf{x}, t)$ and differentiating formulas (6) and (9) with respect to t will bring $\phi(\mathbf{x}, t)$ into the equations.

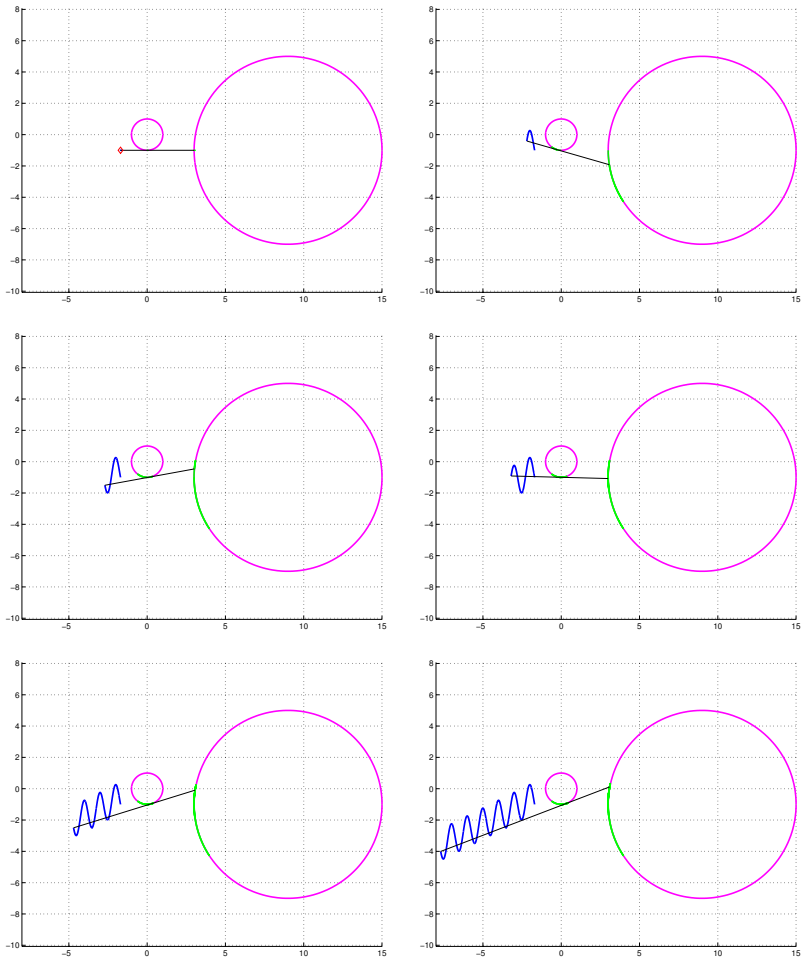


Figure 12: A result of tracking the horizon and cast horizon motion using the formulas derived in this paper. The blue curve represents the trajectory of the vantage point and the green curves represent the paths of the horizon and cast horizon. The black line links the current position of the vantage point and the cast horizon; it shows that the colinearity of the vantage point, the horizon and its cast is preserved.

3.4 Analysis of the motions

PDEs for moving the level set functions

Once we have the motion laws, we can extend the velocity to the domain near the surfaces and obtain the corresponding velocity field $\mathbf{v}(\mathbf{x})$. We then evolve the level set function(s) u in question by

$$u_t + \mathbf{v} \cdot \nabla u = 0.$$

Furthermore, the velocity fields for horizon and cast horion motions do not depend on the function u. In horizon motion, the velocity is a function of postion, time, $\dot{\mathbf{x}}_\mathbf{o}$, and the derivatives of ϕ, i.e. $\mathbf{v} = \mathbf{v}(t, \mathbf{x}, \mathbf{x_o}, \nabla\phi, D^2\phi)$. Furthermore, the level set function to be evolved is $\hbar$. In cast horizon motion, we have $\tilde{\mathbf{v}} = \tilde{\mathbf{v}}(t, \mathbf{x}, \mathbf{x_o}, \nabla\phi, D^2\phi, \hbar)$, and the level set function to be evolved is $\tilde{\psi}$. Therefore, we are evolving the following two level set equations:

$$\hbar_t + \mathbf{v}(\mathbf{t}, \mathbf{x}, \mathbf{x_o}, \nabla\phi, D^2\phi) \cdot \nabla\hbar = 0,$$

$$\tilde{\psi}_t + \tilde{\mathbf{v}}(\mathbf{t}, \mathbf{x}, \mathbf{x_o}, \nabla\phi, D^2\phi, \hbar) \cdot \nabla\tilde{\psi} = 0.$$

These are simple convection equations whose viscoity solutions are well studied, provided that the velocity fields are bounded. We only have to be careful near singularities.

Singularities in the velocity fields

Formula (7) reveals a few interesting facts. First, we notice that the speed of the horizon motion is inversely proportional to the normal curvature in the viewing direction and to the distance between the horizon and the vantage point. If the vantage point is moving in the tangent direction ν, the horizon will not move (since $\dot{\mathbf{x}}_o \cdot \mathbf{n} = 0$). The speed of the horizon motion becomes singular if the curvature of the surface at the horizon location becomes zero. On strictly convex objects, this will never happen. If we restrict our analysis to a single connected smooth non-convex object, we will see easily that at the instance in which a horizon point moves into the location where $\kappa = 0$, a neightborhood of this location becomes completely visible. This signifies the disappearance of the horizon point. If the course of the vantage point is reversed, we get the genesis of a new horizon point.

Formula (10) tells us that the motion of a cast horizon point becomes singular when it is a horizon point ($\nu \cdot \mathbf{n} = 0$). On a single non-convex smooth surface, this happens precisely when a horizon point and its cast across the concavity collide into each other at the location where $\kappa = 0$. In the setting where there are multiple

strictly convex objects, this also describes the changing of the cast horizon into a visible horizon point which is previously invisible. Therefore, the singularities of the horizons and cast horizons describe a part of their genesis. A complete genesis of the visible contours includes another part, in which a hidden object suddenly becomes visible. We shall discuss this point in a later subsection.

3.5 Relating horizon and its shadow

To move the cast horizon, following the notation used in the previous section, we need to find $\mathbf{x}^*(\mathbf{x})$ for each point $\mathbf{x}$ on the cast horizon.

3.5.1 Explicit formula

$\mathbf{x}$ and $\mathbf{x}^*(\mathbf{x})$ are related by

$$\mathbf{x}^*(\mathbf{x}) := \mathbf{x} - r(\mathbf{x})\nu(\mathbf{x});$$

$r(\mathbf{x})$ can be computed by

$$r(\mathbf{x}) = |\mathbf{x} - \mathbf{x}_o| - \rho(\nu(\mathbf{x})),$$

where ν and ρ are defined as previously.

3.5.2 Implicit formulation

We follow the spirit of formula (2) introduced in Section 2 and propagate the link between the horizon and its cast shadow implicitly. Define $\vartheta : \mathbb{R}^d \mapsto \mathbb{R}^d$ to be

$$\vartheta(\mathbf{x}) := \begin{cases} \mathbf{x} & \text{if } \tilde{\psi}(\mathbf{x}) = \phi(\mathbf{x}) \\ \mathbf{x}' & \text{if } \tilde{\psi}(\mathbf{x}) = \tilde{\psi}(\mathbf{x}'). \end{cases}$$

$\tilde{\psi}$ is the function defined in Section 3.1. When we move the points $\mathbf{x}$ near the cast horizon, we also move the points $\vartheta(\mathbf{x})$, which are points near the horizon. By continuity around the cast horizon, we will have the right motion of the cast horizon.

3.6 Reinitialization and emergence-time estimate

It can be easily seen from figure 13 that a completely hidden object may suddenly become visible at a later time during the journey of the vantage point. At the time of emergence, we need to reinitialize our algorithm, i.e., we need to find the apparent contours on the newly emerged surfaces to get the correct visibility information.

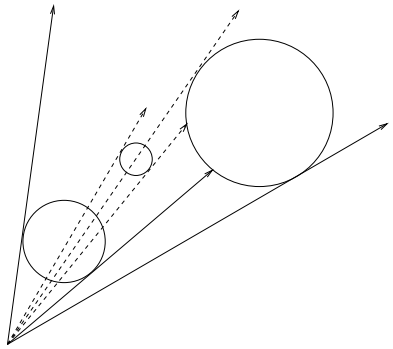


Figure 13: Model scenario I

An explicit reinitialization criterion

Assuming that we are merely tracking the visibility boundaries on the objects. How do we know when to initialize? We can formalize the reasoning as follows. We define the map $\mathcal{G}^{-1} : S^{d-1} \mapsto \{\mathbf{x}_i\}_{i=0}^{N(\theta)}$ such that

$$\mathcal{G}^{-1}(\theta) := \{\mathbf{x}_i \in \mathbb{R}^d : \nabla\phi(\mathbf{x}_i)/|\nabla\phi(\mathbf{x}_i)| = \theta \text{ and } \phi(\mathbf{x}_i) = 0\}.$$

This map is the inverse of the Gauss map in the case that $\{\phi = 0\}$ is a strictly convex hypersurface. Let $\mathcal{S}$ be the set containing x and x^*. We reinitialize whenever there exists an $x \in S$ such that $\exists\, y \in \mathcal{G}(\nu(\mathbf{y}))$ with $\mathbf{x}^*(\mathbf{x}) \prec y \prec \mathbf{x}$.

This provides an explicit criterion for reinitialization. However, we can do better with our implicit formulation. This amounts to knowing 1) how the shadow moves 2) how far a hidden surface is from the shadow boundaries.

Emergence-time estimate

Given current vantage point position and its motion, we want to estimate the emergence time for an object that is occluded. We begin by assuming the the curvatures of the surfaces locally around the regions of interest are constant. The diagram in Figure 14 shows a model configuration: The small circle is initially occluded by the larger circle on the left. We want to estimate the time interval δt between this instance t_0 and the time $t_1 = t_0 + \delta t$ when the small circle first emerges into the scene.

Following the discussion above, consider a point $\mathbf{y}$ on the shadow boundaries away from the horizon such that $\nabla\phi(\mathbf{y}) \perp \nu(\mathbf{y})$ and $\nabla\phi(\mathbf{y}) \cdot \nabla\phi(\mathbf{y}(\mathbf{x})) > 0$. This is the point closest to some hidden part of the objects.

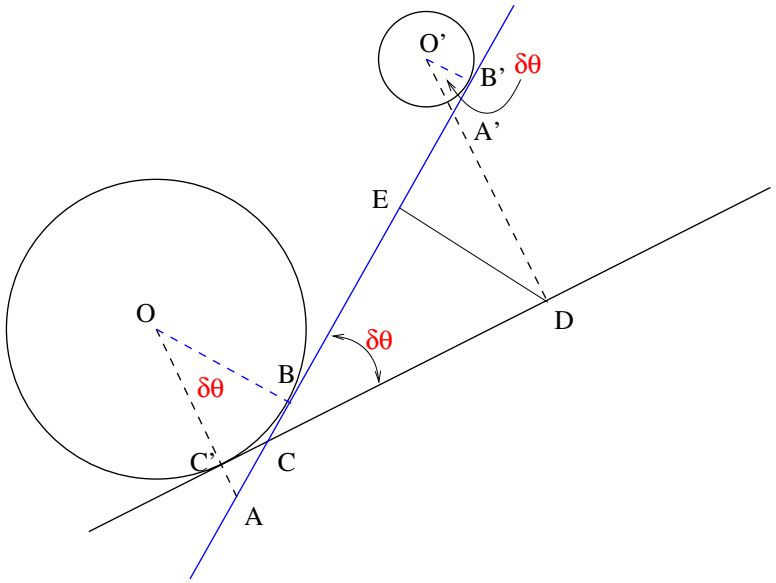


Figure 14: A diagram for emergence time estimate.

For consistency of notation, we will use D in place of $\mathbf{y}$. Let d be the distance between D and the circle centered at O'. Let ρ and ρ' be the radius of the circle centered at O and O' respectively. Let r denote the distance between D and C'. By elementary Euclidean geometry, we have the following identities:

$$CC' = \rho \tan \frac{\delta\theta}{2}, \;\; CD = r - CC' = r - \rho \tan \frac{\delta\theta}{2}, \;\; \Longrightarrow DE = CD \sin \delta\theta,$$

$$O'A' = \frac{1}{\cos \delta\theta} \rho'.$$

Therefore, we can find $\delta\theta$ from the last two equalities. Since we know how fast the horizon is moving, we can then determine δt.

Let $\dot{\mathbf{x}} = |\dot{\mathbf{x}}|_\infty$,

$$\frac{DE}{\delta t} = |\dot{\mathbf{x}}|_\infty, \;\; \delta t = \frac{DE}{|\dot{\mathbf{x}}|_\infty}.$$

Further considerations

We have mentioned in the beginning of this paper that the approach of moving the visible contour may not be more efficient than simply performing implicit ray tracing in general "large world" configurations. Here we construct such a case to

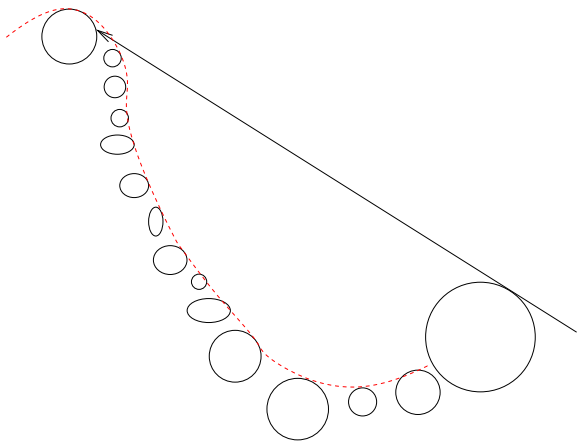


Figure 15: bad case for moving curves

validate our arguments. Consider a nonconvex part of an object as shown by the dashed red curve in Figure 15. Suppose the red curve is broken down into dense small disconnected components. In the case where the red curve is the nonconvex part of a connected component and with the viewing direction being depicted in Figure 15, the cast horizon will move continuously on the nonconvex part of the object without need for reinitialization. However, in the second case, we have to reinitialize very often because the cast horizon will "jump" from component to component.

4 Conclusion and future directions

In this article, we introduced a fast implicit ray tracing algorithm independent of grid geometry and easily parallelizable. This is then extended to a multi-resolution algorithm for near optimal efficiency. Furthermore, we showed that the implicit framework captures accurately the shadow boundaries, which include the horizon and cast horizon curves. We studied how these objects move when the source point is moving. Explicit formulas which reveal the relations between the motions and the local/global geometry of the given configuration are derived and are tightly coupled with our level set framework for implementation. Also, questions such as "how soon will this hidden object appear" can be answered as a result of our algorithm.

There is a rich pool of applications related to the visibility problem described in this paper. Currently we are working on problems related to navigation, visibility with occluders changing shapes in time, in non-uniform media. Our solutions

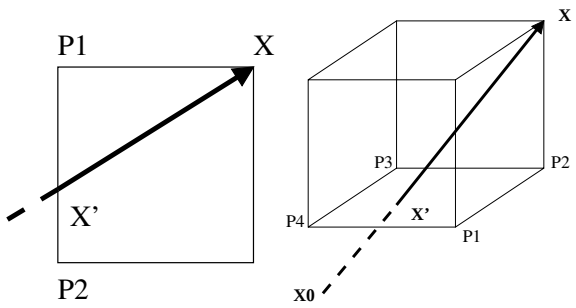


Figure 16: A demonstration of 2D and 3D interpolation

will combine approaches both from the PDE formulation and the algorithms in computational geometry.

Acknowledgement

The author YT wishes to thank the organizers of IPAM Geometrically Based Motion Workshop and IPAM staff.

5 Appendix

5.1 Interpolation schemes

Since the majority of visibility applications benefit from the simplicity of Cartesian grids, we need to adapt the algorithm in order to take advantage of this. As described in the algorithm, at each grid point $x = (i, j, k)$, we need to determine an upwind neighbor x' and find the value of $\psi(x')$. In most cases, x' does not lie on the grid. Therefore, we need to interpolate the values of ψ from the grid points closest to x'. For simplicity and speed considerations, we choose to perform linear interpolation in 2D and bilinear interpolation in 3D. In Figure 16, we use $\psi(P_1)$ and $\psi(P_2)$ for linear interpolation in the 2D case and use $\psi(P_i)$, $i = 1, 2, 3, 4$, for bilinear interpolation of ψ.

We note that a fast marching or fast sweeping strategy for determining distance from the source point and passing values can be used in place of this interpolation.

Let ψ_{int} be the interpolant near x', we know that $\psi_{\text{int}}(x') = \psi(x') + \mathcal{O}(h^2)$. Thus, the discrete visibility equation (2) is in effect

$$\psi(x) = \min(\psi_{\text{int}}(x'), \phi(x)).$$

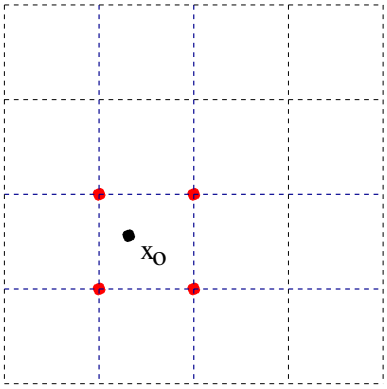


Figure 17: The red points denote the cell vertices.

5.2 Examples of star-shaped updating sequence (sweeping)

There are many different ways of implementing a star-shaped updating sequence. One approach is to use the algorithm based on the heap sort strategy [25] to find grid nodes for update based on their distance to the vantage point. However, due the complexity involved with heap sort, this algorithm is not optimal.

Alternatively, we use a sweeping approach in our simulation. For example, let us consider a Cartesian grid in 2D and assume that the vantage point lies on a grid node; we can then consider separately the visibility problem in each of the four quadrants centered at the vantage point. For simplicity, let us assume that the vantage point is at the origin and the grid is represented by the lattice $[-n_x, n_x] \times [-n_y, n_y] \subset \mathbb{Z}^2$. A compact way of writing this sweeping sequence in C/C++ is:

```
for(s1=-1;s1<=1;s1+=2)
for(s2=-1;s2<=1;s2+=2)
for(i=0;(s1<0?i>=-nx:i<=nx);i+=s1)
for(j=0;(s2<0?j>=-ny:j<=ny);j+=s2)
 update ψ_{i,j}.
```

In the case where $\mathbf{x_o}$ does not lie on a grid node, we describe an easy modification to the updating sequence above. Let $\mathbf{x_o} \in I_o := [x_{i_0}, x_{i_0+1}) \times [y_{j_0}, y_{j_0+1})$. Update the values of ψ on the vertices of I_o. Then update the grid nodes in the strips $\{(x_i, y_j) : i = i_0, i_0 + 1 \text{ and } j = -n_y \text{ to } n_y\}$ and $\{(x_i, y_j) : i = -n_x, n_x \text{ and } j = j_0, j_{0+1}\}$. Finally, update the remaining four quadrants independently. See Figure 17 for a depiction of this approach.

5.3 Finding the curvature of a specified direction

As we argued in Section 3.2, the three dimensional problem of determining the motion of the horizon can be reduced to an instantaneous two dimensional problem. In order to move the horizon in this manner, we need to evaluate the curvature of the surface in the specified direction. Here we present a way to do that.

Let τ be the tangent vector being specified. We want to find the curvature on $\partial\Omega$ in this direction. First let $p(\mathbf{x};\tau)$ be the plane passing through $\mathbf{x}$, spanned by $\mathbf{n}(x)$ and τ, and let P be the level set function that embeds this plane. Then

$$\tilde{\tau} = \frac{\nabla\phi \times \nabla P}{|\nabla\phi \times \nabla P|},$$

where $\tilde{\tau}(\mathbf{x}) = \tau$, and the curvature is

$$k_\tau \mathbf{n} = \vec{\nabla}\tilde{\tau} \cdot \tilde{\tau}.$$

5.4 Derivation of the dynamics of horizon

We follow the constraints (6):

$$\begin{cases} \phi(\mathbf{x}) = 0 \\ (\mathbf{x} - \mathbf{x}_0) \cdot \nabla\phi(\mathbf{x}) = 0 \end{cases}$$

and differentiate with respect to t, we have

$$\nabla\phi(\mathbf{x}) \cdot \dot{\mathbf{x}} = 0 \tag{12}$$

$$(\mathbf{x} - \mathbf{x}_o) \cdot D^2\phi(\mathbf{x})\dot{\mathbf{x}} = \dot{\mathbf{x}}_o \cdot \nabla\phi(\mathbf{x}) \tag{13}$$

In 2 space dimensions, these two relations uniquely determine the motion of x with given initial conditions. Writing $(\mathbf{x} - \mathbf{x}_o) = |\mathbf{x} - \mathbf{x}_o|\, n^{\perp}(\mathbf{x}) = |\mathbf{x} - \mathbf{x}_o|\,(-\phi_y, \phi_x)/|\nabla\phi|$, we have

$$\begin{aligned} &\frac{|\mathbf{x} - \mathbf{x}_o|}{|\nabla\phi|} \begin{pmatrix} -\phi_y \\ \phi_x \end{pmatrix} \cdot \begin{pmatrix} \phi_{xx} & \phi_{xy} \\ \phi_{yx} & \phi_{yy} \end{pmatrix} \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} \\ &= \frac{|\mathbf{x} - \mathbf{x}_0|}{|\nabla\phi|} \begin{pmatrix} -\phi_y\phi_{xx} + \phi_x\phi_{yx} \\ -\phi_y\phi_{xy} + \phi_x\phi_{yy} \end{pmatrix} \cdot \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix}, \end{aligned}$$

and

$$\begin{aligned} \frac{|\mathbf{x} - \mathbf{x}_o|}{|\nabla\phi|} \begin{pmatrix} \phi_x & \phi_y \\ -\phi_y\phi_{xx} + \phi_x\phi_{yx} & -\phi_y\phi_{xy} + \phi_x\phi_{yy} \end{pmatrix} \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} \\ = \begin{pmatrix} 0 \\ \dot{\mathbf{x}}_o \cdot \nabla\phi(\mathbf{x}) \end{pmatrix} \end{aligned}$$

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \frac{|\nabla\phi|}{|\mathbf{x}-\mathbf{x}_o|}\frac{1}{D}\begin{pmatrix} -\phi_y\phi_{xy}+\phi_x\phi_{yy} & -\phi_y \\ \phi_y\phi_{xx}-\phi_x\phi_{yx} & \phi_x \end{pmatrix}\begin{pmatrix} 0 \\ \dot{\mathbf{x}}_o\cdot\nabla\phi(\mathbf{x}) \end{pmatrix}$$
$$= \frac{|\nabla\phi|}{|\mathbf{x}-\mathbf{x}_o|}\frac{\dot{\mathbf{x}}_o\cdot\nabla\phi(\mathbf{x})}{D}\begin{pmatrix} -\phi_y \\ \phi_x \end{pmatrix},$$

where

$$\begin{aligned} D &= \det\begin{pmatrix} \phi_x & \phi_y \\ -\phi_y\phi_{xx}+\phi_x\phi_{yx} & -\phi_y\phi_{xy}+\phi_x\phi_{yy} \end{pmatrix} \\ &= -\phi_x\phi_y\phi_{xy}+\phi_x^2\phi_{yy}+\phi_y^2\phi_{xx}-\phi_x\phi_y\phi_{xy} \\ &= \phi_x^2\phi_{yy}+\phi_y^2\phi_{xx}-2\phi_x\phi_y\phi_{xy}. \end{aligned}$$

Since the curvature of $\partial\Omega$ at x is

$$\begin{aligned} \kappa &= \nabla\cdot\frac{\nabla\phi}{|\nabla\phi|} \\ &= \frac{1}{|\nabla\phi|^3}(\phi_x^2\phi_{yy}+\phi_y^2\phi_{xx}-2\phi_x\phi_y\phi_{xy}), \end{aligned}$$

the motion of x is

$$\dot{\mathbf{x}} = \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \frac{1}{\kappa}\frac{\dot{x}_o\cdot n(\mathbf{x})}{|\mathbf{x}-\mathbf{x}_o|}n^{\perp}(\mathbf{x}). \tag{14}$$

We define $n^{\perp}(\mathbf{x}) = (\mathbf{x}-\mathbf{x}_o)/|\mathbf{x}-\mathbf{x}_o|$.

Alternatively, we can write $\dot{x}$ in a slightly different form:

$$\dot{x} = \frac{P_{\nabla\phi}\nabla^2\phi(x-x_0)}{|P_{\nabla\phi}\nabla^2\phi(x-x_0)|^2}(\dot{x}_0\cdot\nabla\phi),$$

where P_v is the orthogonal projection matrix projecting vectors to the plane with normal vector v. Let us check this expression for the velocity of the curve. Note $\nabla\phi\cdot\dot{x}=0$ since $v\cdot P_v w = 0$ for all vectors v and w. Also, $\nabla^2\phi(x-x_0)\cdot\dot{x} = \nabla\phi\cdot\dot{x}_0$ is satisfied since $P_v w\cdot w = |P_v w|^2$ for all vectors v and w. Thus this velocity is valid and is the first form in our alternate derivation.

Geometric interpretation

If x indeed represents the position of the curve, then $x-x_0$ is tangent to the object surface at x and so $x-x_0 = P_{\nabla\phi}(x-x_0)$. Making this replacement above gives our second form for the velocity,

$$\dot{x} = \frac{P_{\nabla\phi}\nabla^2\phi P_{\nabla\phi}(x-x_0)}{|P_{\nabla\phi}\nabla^2\phi P_{\nabla\phi}(x-x_0)|^2}(\dot{x}_0\cdot\nabla\phi).$$

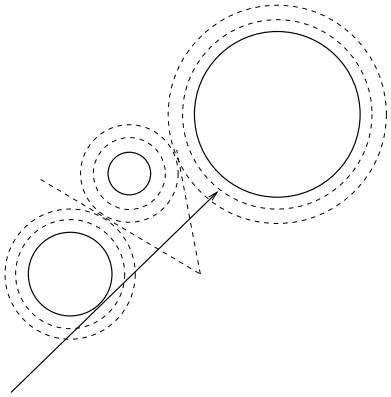


Figure 18: Model cast horizon scenario

This form is particularly nice because we know the second fundamental form in a level set framework is transformed to

$$II = \frac{1}{|\nabla\phi|} P_{\nabla\phi} \nabla^2 \phi P_{\nabla\phi}.$$

Thus we can rewrite the velocity in its final form,

$$\dot{x} = \frac{II(x - x_0)}{|II(x - x_0)|^2} \left(\dot{x}_0 \cdot \frac{\nabla\phi}{|\nabla\phi|} \right).$$

IIv evaluated at a point represents the change in the normals of the object surface in the direction of v at that point.

5.5 Derivation of the dynamics of the cast horizon

We assume that the level sets of ϕ near $\mathbf{x}$ are smooth curves and are not tangent to ν. In two space dimension, we have two equations that determine the dynamics of $\mathbf{x}$:

$$\begin{cases} \phi(\mathbf{x}) = \text{constant} \\ \nu = \tilde{\nu} \end{cases} \tag{15}$$

Let $\mathbf{r}$ and $\tilde{\mathbf{r}}$ denote $(\mathbf{x} - \mathbf{x_o})$ and $(\tilde{\mathbf{x}} - \mathbf{x_o})$ respectively. Differentiating these equations, we arrive at:

$$\nabla\phi(\mathbf{x}) \cdot \mathbf{x}' = 0,$$

$$\frac{\mathbf{r}'}{|\mathbf{r}|} - \frac{\mathbf{r}}{|\mathbf{r}|^2} \nu \cdot \mathbf{r}' = \frac{\tilde{\mathbf{r}}'}{|\tilde{\mathbf{r}}|} - \frac{\tilde{\mathbf{r}}}{|\tilde{\mathbf{r}}|^2} \tilde{\nu} \cdot \tilde{\mathbf{r}}'.$$

Notice that the term

$$\nu \cdot \mathbf{r}' \frac{\mathbf{r}}{|\mathbf{r}|} = (\nu \cdot \mathbf{r}')\nu$$

is $\mathbf{r}'$ projected onto the unit vector ν. Therefore, the left hand side denotes the projection of $\mathbf{r}'$ onto the unit vector $\nu^\perp$:

$$\frac{1}{|\mathbf{r}|}(\mathbf{r}' - \nu \cdot \mathbf{r}'\nu) = \frac{\mathbf{r}' \cdot \nu^\perp}{|\mathbf{r}|} = \frac{1}{|\mathbf{r}|} P_\nu \mathbf{r}'.$$

Similarly, with the right hand side, we have the equation:

$$\frac{1}{|\mathbf{r}|} P_\nu \mathbf{r}' = \frac{1}{|\tilde{\mathbf{r}}|} P_{\tilde{\nu}} \tilde{\mathbf{r}}'.$$

Keeping in mind that we want to solve for $\mathbf{x}'$, we move every other term to the right hand side and arrive at

$$P_\nu \mathbf{x}' = \frac{|\mathbf{r}|}{|\tilde{\mathbf{r}}|} P_{\tilde{\nu}} \tilde{\mathbf{r}}' + P_\nu \mathbf{x}'_{\mathbf{o}},$$

$$\nabla \phi(\mathbf{x}) \cdot \mathbf{x}' = 0.$$

In two dimensions, $P_\nu w = (w \cdot \nu^\perp)\nu^\perp$**,** and $P_\nu w \cdot \nu^\perp = w \cdot \nu^\perp$, therefore, we have

$$\begin{pmatrix} \nu_2 & -\nu_1 \\ \phi_{x_1} & \phi_{x_2} \end{pmatrix} \begin{pmatrix} x'_1 \\ x'_2 \end{pmatrix} = \begin{pmatrix} \frac{|\mathbf{r}|}{|\tilde{\mathbf{r}}|} \tilde{\mathbf{r}}' \cdot v^\perp + \mathbf{x}'_{\mathbf{o}} \cdot \nu^\perp \\ 0 \end{pmatrix},$$

and consequently,

$$\begin{aligned} \begin{pmatrix} x'_1 \\ x'_2 \end{pmatrix} &= \frac{1}{\nu \cdot \nabla \phi(\mathbf{x})} \cdot \\ &\quad \begin{pmatrix} \phi_{x_2} & \nu_1 \\ -\phi_{x_1} & \nu_2 \end{pmatrix} \begin{pmatrix} \frac{|\mathbf{r}|}{|\tilde{\mathbf{r}}|} \tilde{\mathbf{r}}' \cdot \nu^\perp + \mathbf{x}'_{\mathbf{o}} \cdot \nu^\perp \\ 0 \end{pmatrix} \\ &= \frac{1}{\nu \cdot \nabla \phi(\mathbf{x})} \\ &\quad (\frac{|\mathbf{r}|}{|\tilde{\mathbf{r}}|} \tilde{\mathbf{r}}' \cdot \nu^\perp + \mathbf{x}'_{\mathbf{o}} \cdot \nu^\perp) \begin{pmatrix} \phi_{x_2} \\ -\phi_{x_1} \end{pmatrix} \\ &= \frac{\mathbf{n}^\perp(\mathbf{x})}{\nu \cdot \mathbf{n}(\mathbf{x})} (\frac{|\mathbf{r}|}{|\tilde{\mathbf{r}}|} \tilde{\mathbf{r}}' + \mathbf{x}'_{\mathbf{o}}) \cdot \nu^\perp \\ &= \frac{(\frac{|\mathbf{r}|}{|\tilde{\mathbf{r}}|} \tilde{\mathbf{r}}' + \mathbf{x}'_{\mathbf{o}}) \cdot \nu^\perp}{\nu \cdot \mathbf{n}(\mathbf{x})} \mathbf{n}^\perp(\mathbf{x}) \end{aligned}$$

where

$$\nabla^\perp \phi(\mathbf{x}) := \begin{pmatrix} \phi_{x_2} \\ -\phi_{x_1} \end{pmatrix}, \text{ and } n^\perp := \frac{\nabla^\perp \phi(\mathbf{x})}{|\nabla^\perp \phi(\mathbf{x})|}.$$

5.6 An algorithm to project $\{\psi = 0\}$ to S^{n-1}

We discretize S^{n-1} over a Cartesian grid: $\rho : [0, 2\pi) \times [-\pi, \pi) \mapsto x_o + S^{n-1}/r$, with r large enough so that $x_o + S^{n-1}/r$ lies completely outside of ψ. $\Delta\theta_1$, $\Delta\theta_2$ are set to be the smallest angle possible on the grid that discretizes ψ. Therefore, we can set $\Delta\theta_1 = \tan^{-1}(\min(\Delta x, \Delta y)/L)$, where L is the diagonal length of the grid.

For each $\rho_{l,m}$, we then shoot a ray, starting from $p_0 = \rho_{l.m}$ outward according to the angle determined by (l, m), and use the bisection method to find p_1, the intersection of the ray and $\{\psi = 0\}$.

Note that the complexity is $N^2 \log N$ for a grid of size N^3.

5.7 Numerics

We computed the quantities describe in this paper using standard level set technologies. Please refer to [19, 20, 21, 22, 10] for details.

5.8 A list of level set functions used in this paper

We provide a comprehensive list of the level set functions we construct in this paper:

ϕ embeds the objects

$\hbar(\mathbf{x}) := (\mathbf{x} - \mathbf{x_o}) \cdot \nabla\phi(\mathbf{x})$ characterizes the horizon

$\tilde{\phi} := \max(\phi, -\hbar)$ $\{\tilde{\phi} \leq 0\} = \{\phi \leq 0\} \setminus \{\hbar < 0\}$, defines the same visibility as ϕ

ψ the visibility map resulting from the implicit ray tracing on ϕ

$\tilde{\psi}$ the visibility map resulting from the implicit ray tracing on $\tilde{\phi}$, characterizes the cast horizon

ϑ: $\mathbb{R}^d \mapsto \mathbb{R}^d$ links horizon to its cast implicitly

References

[1] D. Adalsteinsson and J.A. Sethian. An overview of level set methods for etching, deposition, and lithography development. *IEEE Transactions on Semiconductor Devices*, 10(1), February 1997.

[2] Pankaj K. Agarwal and Micha Sharir. Ray shooting amidst convex polygons in 2D. *J. Algorithms*, 21(3):508–519, 1996.

[3] Pankaj K. Agarwal and Micha Sharir. Ray shooting amidst convex polyhedra and polyhedral terrains in three dimensions. *SIAM J. Comput.*, 25(1):100–116, 1996.

[4] Luis Alvarez, Frédéric Guichard, Pierre-Louis Lions, and Jean-Michel Morel. Axioms and fundamental equations of image processing. *Arch. Rational Mech. Anal.*, 123(3):199–257, 1993.

[5] M. Bertalmio, G. Sapiro, and G. Randall. Dynamic visibility in an implicit framework. *IEEE Trans. Medical Imaging*, 18:448–451, 1999.

[6] Marcelo Bertalmio, Li-Tien Cheng, Stanley Osher, and Guillermo Sapiro. Variational problems and partial differential equations on implicit surfaces. *J. Comput. Phys.*, 174(2):759–780, 2001.

[7] M. D. Betterton. Theory of structure formation in snowfields motivated by penitentes, suncups, and dirt cones. *Physical Review E*, 63, 2001.

[8] Paul Burchard, Li-Tien Cheng, Barry Merriman, and Stanley Osher. Motion of curves in three spatial dimensions using a level set approach. *J. Comput. Phys.*, 170:720–741, 2001.

[9] J. C. Carr, R. K. Beatson, J.B. Cherrie T. J. Mitchell, B. C. McCallum W. R. Fright, and T. R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *SIGGRAPH*, 2001.

[10] Li-Tien Cheng, Paul Burchard, Barry Merriman, and Stanley Osher. Motion of curves constrained on surfaces using a level set approach. *J. Comput. Phys.*, 175:604–644, 2002.

[11] Roberto Cipolla and Peter Giblin. *Visual motion of curves and surfaces*. Cambridge University Press, 2000.

[12] Satyan Coorg and Seth Teller. Real-time occlusion culling for models with large occluders. In *ACM Symposium on Interactive 3D Graphics*, 1997.

[13] Fredo Durand. 3d visibility: Analysis study and applications. *PhD Thesis, MIT*, 1999.

[14] Fredo Durand, George Drettakis, Joëlle Thollot, and Claude Puech. Conservative visibility preprocessing using extended projections. In *SIGGRAPH*, 2000.

[15] Memoli Facundo and Guillermo Sapiro. Fast computation of distance functions and geodesics on implicit surfaces. To appear, Journal of Computational Physics (2001).

[16] Aaron Hertzmann and Denis Zorin. Illustrating smooth surfaces. In *SIGGRAPH*, 2001.

[17] Hailin Jin, Anthony Yezzi, and Stefano Soatto. Stereoscopic shading: Integating multi-frame shape cues in a variational framework. *In Preparation*.

[18] Stanley Osher, Li-Tien Cheng, Myungjoo Kang, Hyeseon Shim, and Yen-Hsi Tsai. Geometric optics in a phase space based level set and eulerian framework. 2001. Submitted to Journal of Computational Physics.

[19] Stanley Osher and Ronald P. Fedkiw. Level set methods: an overview and some recent results. *J. Comput. Phys.*, 169(2):463–502, 2001.

[20] Stanley Osher and James A. Sethian. Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations. *J. Comput. Phys.*, 79(1):12–49, 1988.

[21] Stanley Osher and Chi-Wang Shu. High-order essentially nonoscillatory schemes for Hamilton-Jacobi equations. *SIAM J. Numer. Anal.*, 28(4):907–922, 1991.

[22] Danping Peng, Barry Merriman, Stanley Osher, Hongkai Zhao, and Myungjoo Kang. A PDE-based fast local level set method. *J. Comput. Phys.*, 155(2):410–438, 1999.

[23] G. Sapiro, R. Kimmel, D. Shaked, B. B. Kimia, and A. M. Bruckstein. Implementing continuous scale morphology via curve evolution. *Pattern Recognition*, 26(9):1363–1372, 1993.

[24] J. A. Sethian. *Level set methods and fast marching methods*. Cambridge University Press, Cambridge, second edition, 1999. Evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science.

[25] John Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Transactions on Automatic Control*, 40(9):1528–1538, 1995.

[26] Hong-Kai Zhao, Stanley Osher, Barry Merriman, and Myungjoo Kang. Implicit and non-parametric shape reconstruction from unorganized points using variational level set method. *Computer Vision and Image Understanding*, 80:295–319, 2000.

Fast Surface Reconstruction Using the Level Set Method

Hong-Kai Zhao* Stanley Osher† Ronald Fedkiw‡

Abstract

In this paper we describe new formulations and develop fast algorithms for implicit surface reconstruction based on variational and partial differential equation (PDE) methods. In particular we use the level set method and fast sweeping and tagging methods to reconstruct surfaces from scattered data set. The data set might consist of points, curves and/or surface patches. A weighted minimal surface-like model is constructed and its variational level set formulation is implemented with optimal efficiency. The reconstructed surface is smoother than piecewise linear and has a natural scaling in the regularization that allows varying flexibility according to the local sampling density. As is usual with the level set method we can handle complicated topologies and deformations, as well as noisy or highly non-unifrom data sets easily. The method is based on a simple rectangular grid, although adaptive and triangular grids are also possible. Some consequences, such as hole filling capability, are demonstrated, as well as a rigorous proof of the viability and convergence of our new fast tagging algorithm.

Keywords: implicit surface, partial differential equations, variational formulation, convection, minimal surface, hole filling

1 Introduction

Surface reconstruction from unorganized data set is very challenging in three and higher dimensions. The problem is ill-posed, i.e, there is no unique solution. Furthermore the ordering or connectivity of data set and the topology of the real surface can be very complicated in three and higher dimensions. A desirable reconstruction procedure should be able to deal with complicated topology and geometry as well as noise and non-uniformity of the data to construct a surface that is a good approximation of the data set and has some smoothness (regularity). Moreover, the reconstructed surface should have a representation and data structure that not only good for static rendering but also good for deformation, animation and other dynamic operation on surfaces. None of the present approaches possess all of these properties. In general there are two kinds of surface representations, explicit or implicit. Explicit surfaces prescribe the precise location of a surface while implicit surfaces represent a surface as a particular isocontour of a scalar function. Popular explicit representations include parametric surfaces and triangulated surfaces. For examples, for parametric surfaces such as NURBS [22, 23], the reconstructed surface is smooth and the data set can be non-uniform. However this requires one to parametrize the data set in a nice way such that the reconstructed surface is a graph in the parameter space. The parametrization and patching can be very difficult for surface reconstruction from an arbitrary data set in three and higher dimensions. Also noise in the data set is difficult to deal with. Another popular approach in computer graphics is to reconstruct a triangulated surfaces using Delaunay triangulations and Voronoi diagrams. The reconstructed surface is typically a subset of the faces of the Delaunay triangulations. A lot of work has been done along these lines [3, 4, 5, 8, 12, 13] and efficient algorithms are available to compute Delaunay triangulations and Voronoi diagrams. Although this approach is more versatile in that it can deal with more general data sets, the constructed surface is only piecewise linear and it is difficult to handle non-uniform and noisy data. Furthermore the tracking of large deformations and topological changes is usually quite difficult using explicit surfaces.

Recently, implicit surfaces or volumetric representations have attracted a lot of attention. There are two main approaches for creating and analyzing implicit surfaces. The traditional approach [7, 18, 27, 29] uses a combination of smooth basis functions, such as blobs, to find a scalar function such that all data points are close to an isocontour of that scalar function. This isocontour represents the constructed implicit surface. Although the implicit surface is usually smooth, the construction is global, i.e. all the basis functions are coupled together and a single data point change can result in globally different coefficients. This makes human interaction, incremental updates and deformation difficult. The second approach uses the data set to define a signed distance function on rectangular grids and denotes the zero isocontour of the signed distance function as the reconstructed implicit surface [6, 9, 16]. The construction of the signed distance function uses a discrete approach and needs an estimation of local tangent planes or normals for the orientation, i.e. a distinction needs to be made between inside and outside. Similar ideas have been applied to shape reconstruction from range data and image fusion [11, 15] where partial connections are available on each piece of data and some "zippering" is needed to patch things together. The advantages of implicit surfaces include topological flexibility, a simple data structure, depth/volumetric information and memory storage efficiency. Using the signed distance representation, many surface operations such as Boolean operations, ray tracing and offset become quite simple. Moreover an extremely efficient marching cubes algorithm [17] is available to turn an implicit surface into a triangulated surface.

We approach this fundamental problem on the continuous level by constructing continuous models using differential geometry and partial differential equations. We also develop efficient and robust numerical algorithms for our continuous formulations. Moreover we combine the level set method and implicit surfaces to provide a general framework for surface modeling, analysis, deformation and many other applications. In our previous work [31] we proposed a new "weighted" minimal surface model based on variational formulations and PDE methods. Only the unsigned distance function to the data set was used in our formulation. Our reconstructed surface is smoother than piecewise linear. In addition, in our formulation there is a regularization that is adaptive to the local sampling density which can keep sharp features if the a local sampling condition is satisfied. The formulation handles noisy as well as non-unform data and works in any number of dimensions. We use the level set method as the numerical technique to deform the implicit surface continuously following the gradient descent of the

*Department of Mathematics, University of California, Irvine, CA 92697-3875, Research supported by NSF DMS 9706566. zhao@math.uci.edu

†Department of Mathematics, University of California, Los Angeles, CA 90095-1555, Research supported by ONR N00014-97-1-0027 and NSF DMS 9706827. sjo@math.ucla.edu

‡Stanford University, Gates Computer Science Bldg., Stanford, CA 94305-9020, Research supported by ONR N00014-97-1-0027. fedkiw@cs.stanford.edu

energy functional for the final reconstruction. Instead of tracking a parametrized explicit surface we solve an PDE on a simple rectangular grid and handle topological changes easily. In this paper we develop a simple physically motivated convection model and a fast tagging algorithm to construct a good initial approximation for our minimal surface reconstruction. This will speed up our previous reconstruction by an order of magnitude. We also introduce a smoothing algorithm similar to [28] as a post process to smooth implicit surfaces or reconstructed implicit surfaces from noisy data.

In the next section we briefly review the variational formulation for the weighted minimal surface model in introduced in [31]. A physically motivated simple convection model is developed in section 3. In section 4 we introduce the level set method for our problems and a simple denoising/smoothing formulation for implicit surfaces . We explain the details of the numerical implementation and fast algorithms in section 5 and show results in section 6.

2 A Weighted Minimal Surface Model

Let $\mathcal{S}$ denote a general data set which can include data points, curves or pieces of surfaces. Define $d(\boldsymbol{x}) = dist(\boldsymbol{x}, \mathcal{S})$ to be the distance function to $\mathcal{S}$. (We shall use bold faced characters to denote vectors.) In [31] the following surface energy is defined for the variational formulation:

$$E(\boldsymbol{\Gamma}) = \left[\int_{\boldsymbol{\Gamma}} d^p(\boldsymbol{x}) ds\right]^{\frac{1}{p}}. \quad 1 \le p \le \infty, \tag{1}$$

where $\boldsymbol{\Gamma}$ is an arbitrary surface and ds is the surface area. The energy functional is independent of parametrization and is invariant under rotation and translation. When $p = \infty$, $E(\boldsymbol{\Gamma})$ is the value of the distance of the point $\boldsymbol{x}$ on $\boldsymbol{\Gamma}$ furthest from $\mathcal{S}$. For $p < \infty$, The surface energy $E(\boldsymbol{\Gamma})$ is equivalent to $\int_{\boldsymbol{\Gamma}} d^p(\boldsymbol{x}) ds$, the surface area weighted by some power of the distance function. We take the local minimizer of our energy functional, which mimics a weighted minimal surface or an elastic membrane attached to the data set, to be the reconstructed surface.

As derived in [31] the gradient flow of the energy functional (1) is

$$\frac{d\boldsymbol{\Gamma}}{dt} = -\left[\int_{\boldsymbol{\Gamma}} d^p(\boldsymbol{x}) ds\right]^{\frac{1}{p}-1} d^{p-1}(\boldsymbol{x}) \left[\nabla d(\boldsymbol{x}) \cdot \boldsymbol{n} + \frac{1}{p} d(\boldsymbol{x}) \kappa\right] \boldsymbol{n}, \tag{2}$$

and the minimizer or steady state solution of the gradient flow satisfies the Euler-Lagrange equation

$$d^{p-1}(\boldsymbol{x}) \left[\nabla d(\boldsymbol{x}) \cdot \boldsymbol{n} + \frac{1}{p} d(\boldsymbol{x}) \kappa\right] = 0, \tag{3}$$

where $\boldsymbol{n}$ is the unit outward normal and κ is the mean curvature. We see a balance between the attraction $\nabla d(\boldsymbol{x}) \cdot \boldsymbol{n}$ and the surface tension $d(\boldsymbol{x})\kappa$ in the equations above. Moreover the nonlinear regularization due to surface tension has a desirable scaling $d(\boldsymbol{x})$. Thus the reconstructed surface is more flexible in the region where sampling density is high and is more rigid in the region where the sampling density is low. In the steady state equation(3) above, since $\nabla d \cdot \boldsymbol{n} \le 1$, a local sampling density condition similar to the one proposed in [4], which says sampling densities should be proportional to the local curvature of the feature. To construct the minimal surface we used a continuous deformation in [31]. We start with an initial surface that encloses all data and follow the gradient flow (2). The parameter p affects the flexibility of the membrane to some extent. When $p = 1$, the gradient flow (2) is scale invariant i.e., dimensionless. In practice we find that $p = 1$ or 2 (similar to a least squares formulation) are good choices. Some more details can be found in [31].

In two dimensions, it was shown in [31] that a polygon which connects adjacent points by straight lines is a local minimum. This result shows a connection between the variational formulation and previous approaches. On the other hand this result is not surprising since a minimal surface passing through two points is a straight line in two dimensions. However in three dimensions the situation becomes much more interesting. The reconstructed minimal surface has no edges and is smoother than a polyhedron.

3 The Convection Model

The evolution equation (2) involves the mean curvature of the surface and is a nonlinear parabolic equation. A time implicit scheme is not currently available. A stable time explicit scheme requires a restrictive time step size, $\Delta t = O(h^2)$, where h is the spatial grid cell size. Thus it is very desirable to have an efficient algorithm to find a good approximation before we start the gradient flow for the minimal surface. We propose the following physically motivated convection model for this purpose.

The convection of a flexible surface $\boldsymbol{\Gamma}$ in a velocity field $\boldsymbol{v}(\boldsymbol{x})$ is described by the differential equation

$$\frac{d\boldsymbol{\Gamma}(t)}{dt} = \boldsymbol{v}(\boldsymbol{\Gamma}(t)).$$

If the velocity field is created by a potential field $\mathcal{F}$, then $\boldsymbol{v} = -\nabla \mathcal{F}$. In our convection model the potential field is the distance function $d(\boldsymbol{x})$ to the data set $\mathcal{S}$. This leads to the convection equation

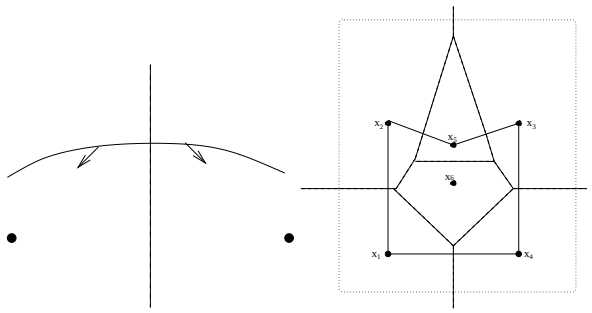
$$\frac{d\boldsymbol{\Gamma}(t)}{dt} = -\nabla d(\boldsymbol{x}). \tag{4}$$

For example, if the data set contains a single point $\boldsymbol{x}_0$, the potential field is $d(\boldsymbol{x})$=$|\boldsymbol{x}-\boldsymbol{x}_0|$ and the velocity field is $\boldsymbol{v}(\boldsymbol{x}) = -\nabla d(\boldsymbol{x}) = -\frac{\boldsymbol{x}-\boldsymbol{x}_0}{|\boldsymbol{x}-\boldsymbol{x}_0|}$, a unit vector pointing towards $\boldsymbol{x}_0$. Any particle in this potential field will be attracted toward $\boldsymbol{x}_0$ along a straight line with unit speed. For a general data set $\mathcal{S}$, a particle will be attracted to its closest point in $\mathcal{S}$ unless the particle is located an equal distance from two or more data points. The set of equal distance points has measure zero. Similarly, points on a curve or a surface, except those equal distance points, are attracted by their closest points in the data set (see Fig. 1(a)). The ambiguity at those equal distance points is resolved by adding a small surface tension force which automatically exists as numerical viscosity in our finite difference schemes. Those equal distance points on the curve or surface are dragged by their neighbors and the whole curve or surface is attracted to the data set until it reaches a local equilibrium (see Fig.1(b)), which is a polygon or polyhedron whose vertices belong to the data set as the viscosity tends to zero (see Fig.1(b)).

Here are some properties of this simple convection model: (1) the normal velocity of the curve or the surface is less than or equal to 1, (2) each point of the curve or surface is attracted by its closest point in the data set.

Figure 1(b) is an illustration of the convection of a curve. The initial curve (the dotted rectangle) feels the attraction of $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4$ and closes in. Then it begins to feel $\boldsymbol{x}_5$. The final shape is a pentagon that goes through $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4$ and $\boldsymbol{x}_5$ while $\boldsymbol{x}_6$ is screened out.

Since the convection equation is a first order linear differential equation, we can solve it using a time step $\Delta t = O(h)$ leading to significant computational savings over typical parabolic $\Delta t = O(h^2)$ time step restrictions. The convection model by itself very often results in a good surface reconstruction. In section 5 we will construct a very fast tagging algorithm that finds a crude approximation of the local equilibrium solution for our convection model.



(a) (b)

(a) the attraction of a piece of curve by two points, (b) dotted line is the initial curve, solid line is the final curve, dashed line is the Voronoi diagram.

Figure 1:

4 The Level Set Formulation

In general we do not have any á priori knowledge about the topology of the shape to be reconstructed. Topological changes may occur during the continuous deformation process. This makes explicit tracking, which requires consistent parametrization, almost impossible to implement. Here we introduce the level set method as a powerful numerical technique for the deformation of implicit surfaces. Although implicit surfaces have been used in computer graphics for quite a while, they were mostly used for static modeling and rendering and were based on discrete formulations [7]. The *level set method* is based on a continuous formulation using PDEs and allows one to deform an implicit surface, which is usually the zero isocontour of a scalar (level set) function, according to various laws of motion depending on geometry, external forces, or a desired energy minimization. In numerical computations, instead of explicitly tracking a moving surface we implicitly capture it by solving a PDE for the level set function on rectangular grids. The data structure is extremely simple and topological changes are handled easily. The level set formulation works in any number of dimensions and the computation can easily be restricted to a narrow band near the zero level set, see e.g. [1, 21]. We can locate or render the moving surface easily by interpolating the zero isosurface of the level set function. The level set method was originally introduced by Osher and Sethian in [20] to capture moving interfaces and has been used quite successfully in moving interface and free boundary problems as well as in image processing, image segmentation and elsewhere. See [19] for a comprehensive review.

Two key steps for the level set method are:

- Embed the surface: we initially represent a co-dimension one surface $\mathbf{\Gamma}$ as the zero isocontour of a scalar (level set) function $\phi(\boldsymbol{x})$, i.e. $\mathbf{\Gamma} = \{\boldsymbol{x} : \phi(\boldsymbol{x}) = 0\}$. $\phi(\boldsymbol{x})$ is negative inside $\mathbf{\Gamma}$ and positive outside $\mathbf{\Gamma}$. Geometric properties of the surface $\mathbf{\Gamma}$, such as the normal, surface area, volume, mean and Gaussian curvature can be easily computed using ϕ. For example, the outward unit normal $\boldsymbol{n}$ is simply $\frac{\nabla\phi}{|\nabla\phi|}$ and the mean curvature κ is $\nabla \cdot \frac{\nabla\phi}{|\nabla\phi|}$.

- Embed the motion: we derive the time evolution PDE for the level set function such that the zero level set has the same motion law as the moving surface, i.e. the moving surface coincides with the zero level set for all time. Since $\mathbf{\Gamma}(t) = \{\boldsymbol{x} : \phi(\boldsymbol{x}, t) = 0\}$,

$$\frac{d\phi(\mathbf{\Gamma}(t), t)}{dt} = \phi_t + \frac{d\mathbf{\Gamma}(t)}{dt} \cdot \nabla\phi = 0, \tag{5}$$

where we replace $\frac{d\mathbf{\Gamma}(t)}{dt}$ with a velocity field $\boldsymbol{v}(\boldsymbol{x})$ defined for all $\boldsymbol{x}$ and equal to $\frac{d\mathbf{\Gamma}(t)}{dt}$ for $\boldsymbol{x}$ on $\mathbf{\Gamma} = \{\boldsymbol{x} : \phi(\boldsymbol{x}, t) = 0\}$.

To develop the time evolution PDE for the level set function, one needs to extend the velocity at the zero level set, which is given by the motion law of the original surface, to other level sets in a natural way. For geometric motions, i.e. where the motion law (velocity) depends only on the geometry of the moving surface, the most natural way to define $\boldsymbol{v}$ is to apply the same motion law for all level sets of the level set function, which will result in a morphological PDE [2]. For example, the gradient flow (2) is a geometric motion. Using the fact (see, e.g., [25, 31])

$$\int_{\Gamma} d^p(\boldsymbol{x}) ds = \int d^p(\boldsymbol{x}) \delta(\phi(\boldsymbol{x})) |\nabla\phi(\boldsymbol{x})| d\boldsymbol{x},$$

where $\phi(\boldsymbol{x}, t)$ is the level set function whose zero level set is $\mathbf{\Gamma}(t)$ and $\delta(x)$ is the one dimensional delta function, and extending the motion (normal velocity) to all level sets we have the level set formulation for the gradient flow (2)

$$\frac{\partial\phi}{\partial t} = \frac{1}{p}|\nabla\phi| \left[\int d^p(\boldsymbol{x})\delta(\phi)|\nabla\phi| d\boldsymbol{x}\right]^{\frac{1}{p}-1} \nabla\cdot\left[d^p(\boldsymbol{x})\frac{\nabla\phi}{|\nabla\phi|}\right], \tag{6}$$

For the convection model (4), since the velocity field $-\nabla d(\boldsymbol{x})$ is defined everywhere, we can naturally extend the convection to all level sets of $\phi(\boldsymbol{x}, t)$ to obtain

$$\frac{\partial\phi}{\partial t} = \nabla d(\boldsymbol{x}) \cdot \nabla\phi. \tag{7}$$

Although all level set functions are equally good theoretically, in practice the signed distance function is preferred for numerical computations. However even if we start with a signed distance function the level set function will generally not remain a signed distance function. As an example, in the convection model all level sets are attracted to the data set simultaneously and they become more and more packed together. We need a procedure to force them apart while keeping the zero level set intact. We use a numerical procedure called reinitialization, see e.g. [21, 25], to redistance the level set function locally without interfering with the motion of the zero level set. The reinitialization process will also provide us with a signed distance function for rendering the implicit surface after the deformation procedure stops.

If the data set contains noise, we derive a post-smoothing process similar to that of [28] for our reconstructed implicit surfaces using the variational level set formulation. Let ϕ_0 denote the initial level set function whose zero level set is the surface we would like to denoise or smooth. We define the denoised or smoothed implicit surface as the zero level set of ϕ that minimizes the following functional

$$\frac{1}{2}\int (H(\phi) - H(\phi_0))^2 d\boldsymbol{x} + \epsilon \int \delta(\phi)|\nabla\phi| d\boldsymbol{x}, \tag{8}$$

where $H(x)$ is the one dimensional Heaviside function. The first term in the above energy functional is a fidelity term that measures the symmetric volume difference between two closed surfaces. The second integral in the above functional is the surface area of the zero level set of ϕ, which is a regularization term that minimizes the surface area of the denoised or smoothed surface. The constant ϵ is a parameter that controls the balance between the fidelity and the regularization. We again find the minimizer by following the gradient flow of (8), whose level set formulation is:

$$\phi_t = |\nabla\phi|[\epsilon\kappa - (H(\phi) - H(\phi_0))]$$

To some extent this variational formulation is also related to Total Variation (TV) denoising for images proposed in [24]. In fact it is exactly TV denoising applied to $H(\phi)$, since the total variation of a function can be represented as the integration of the parameter length of all level sets of the function by co-area formula [14].

5 Numerical Implementation

There are three key numerical ingredients in our implicit surface reconstruction. First, we need a fast algorithm to compute the distance function to an arbitrary data set on rectangular grids. Second, we need to find a good initial surface for our gradient flow. Third, we have to solve time dependent PDEs for the level set function.

5.1 Computing the distance function

The distance function $d(\boldsymbol{x})$ to an arbitrary data set $\mathcal{S}$ solves the following Eikonal equation:

$$|\nabla d(\boldsymbol{x})| = 1, \qquad d(\boldsymbol{x}) = 0,\ \boldsymbol{x} \in \mathcal{S}. \tag{9}$$

From the PDE point of view, the characteristics of this Eikonal equation are straight lines which radiate from the data set. This reveals the causality property for the solution of the PDE, i.e., the information propagates along straight lines from the data set, and the solution at a grid point should be determined only by its neighboring grid points that have smaller distance values. We use an algorithm [10, 31] that combines upwind differencing with Gauss-Seidel iterations of different sweeping order to solve (9) on rectangular grids. From numerical experiments it seems that the total number of iterations is independent of mesh size, i.e. the complexity is $O(M+N)$ for N grid points and M data points.

Suppose we have a set of data points and a rectangular grid. We use an initialization procedure, of complexity $O(M+N)$ to assign initial values for N grid points and M data points. Those gird points that belong to the data set are assigned zero. Those grid points that are neighbors (i.e., vertices of grid cells that contain data points,) are assigned the exact distance values. These grid points are our boundary points and their distance values will not change in later computations. We assign a large positive number to all other grid points. These values will be updated in later computations. We can deal with more general data set as long as the distance values on grids neighboring to the set are provided initially. In one dimension, the following upwind differencing is used to discretize the Eikonal equation (9) at ith grid point that are not boundary points,

$$[(d_i - d_{i-1})^+]^2 + [(d_i - d_{i+1})^+]^2 = h^2, \ \ i = 1, 2, \ldots, I \tag{10}$$

where h is the grid size, I is the total number of grids and $(x)^+ = \begin{cases} x & x > 0 \\ 0 & x \le 0 \end{cases}$. We use two different sweeps of Gauss-Seidel iterations successively, i.e., for $i = 1 : I$ and $i = I : 1$, to solve this system of equations. At the ith grid, using the current values of d_{i-1} and d_i, there exists at least one solution for equation (10) $[min(d_{i-1}, d_{i+1}) + \frac{h}{\sqrt{2}}, max(d_{i-1}, d_{i+1}) + \frac{h}{\sqrt{2}}]$, which only depends on neighbors with smaller values. We take d_i to be the smaller one if there are two solutions. It can be shown that these two sweeps will get the exact solution of the discrete system (10), which is of first order $O(h)$ accuracy to the real distance function. In two dimensions, a slightly more complicated system,

$$\begin{aligned}&[(d_{i,j} - d_{i-1,j})^+]^2 + [(d_{i,j} - d_{i+1,j})^+]^2 \\ &+[(d_{i,j} - d_{i,j-1})^+]^2 + [(d_{i,j} - d_{i,j+1})^+]^2 = h^2,\end{aligned}$$

$i = 1, \ldots, I,\ j = 1, \ldots, J$, has to be solved using sweeps of Gauss-Seidel iterations of four different orders,

$$\begin{aligned}&(1)\ i = 1 : I, j = 1 : J \quad (2)\ i = 1 : I, j = J : 1 \\ &(3)\ i = I : 1, j = 1 : J \quad (4)\ i = I : 1, j = J : 1\end{aligned}$$

In most numerical computations, a total of five or six sweeps is enough in two dimensions. Similarly a three dimensional extension is straight forward. This distance algorithm is versatile, efficient and will be used in later stages of the surface reconstruction.

5.2 Finding a good initial guess

We can use an arbitrary initial surface that contains the data set such as a rectangular bounding box, since we do not have to assume any á priori knowledge for the topology of the reconstructed surface. However, a good initial surface is important for the efficiency of our PDE based method. On a rectangular grid, we view an implicit surface as an interface that separates the exterior grid points from the interior grid points. In other words, volumetric rendering requires identifying all exterior (interior) grid points correctly. Based on this idea, we propose a novel, extremely efficient tagging algorithm that can identify as many correct exterior grid points as possible and hence provide a good initial implicit surface. As always, we start from any initial exterior region that is a subset of the true exterior region. Here is the description of our fast tagging algorithm and the proof of its viability. For simplicity of exposition only, we shall consider a uniform grid

$$(x_i, y_j) = \boldsymbol{x}_{ij} = (i\Delta, j\Delta), \ \ i, j = 0, \pm 1, \pm 2, -$$

in 2 dimensions, where Δ is the grid size. The results work in any number of dimensions and for more general grid structure.

Let $d_{ij} = d(\boldsymbol{x}_{ij})$ be the unsigned distance of $\boldsymbol{x}_{ij}$ to the data set $\mathcal{S}$ (i.e. to the closest point on $\mathcal{S}$). We say $\boldsymbol{x}_{ij} < \boldsymbol{x}_{kl}$, or $\boldsymbol{x}_{ij}$ is closer than $\boldsymbol{x}_{kl}$ or $\boldsymbol{x}_{ij}$ is smaller than $\boldsymbol{x}_{kl}$ if $d_{ij} < d_{kl}$.

We define $\mathcal{S}^\Delta$ to be the set of grid nodes $\boldsymbol{x}_{ij}$ for which $d_{ij} < \Delta$. (Note: if every data point lies on a grid node, then $\mathcal{S}^\Delta = \mathcal{S}$).

We wish to obtain a set Ω for which $\mathcal{S}^\Delta \subset \Omega$ and for which the boundary $\partial\Omega$ serves as a very good initial guess for our final reconstructed surface. From our convection model, we need to rapidly find a crude approximate solution to the steady state equation

$$\nabla d(x) \cdot \boldsymbol{n} = 0,$$

where $\boldsymbol{n}$ is the unit outward normal of the boundary $\partial\Omega$. This equation can be written (in 2D) as

$$d_x \phi_x + d_y \phi_y = 0, \tag{11}$$

where ϕ is the level set function whose zero level set is $\partial\Omega$ that surrounds $\mathcal{S}^\Delta$. We wish to march quickly in a manner reminiscent of the fast algorithm of [26], but for a very different problem – this is not the eikonal equation, and steady states generally depend on the initial guess. There are some similarities in that a heap sort algorithm is used as is the Cartesian structure of the grid.

For a point $\boldsymbol{x}_{ij}$ we define its neighbors as the four points $\boldsymbol{x}_{i\pm1,j}, \boldsymbol{x}_{i,j\pm1}$. A boundary point of a set Ω is defined to be the set of $\boldsymbol{x}_{ij}$ in Ω for which at least one of its four neighbors are in the exterior of Ω. The boundary of Ω is denoted by $\partial\Omega$.

Given Ω_0 for which $\mathcal{S}^\Delta$ we shall march quickly towards $\Omega \subset \Omega_0$ whose boundary $\partial\Omega$ will act as our initial level surface for the convection and convection-diffusion algorithms defined below. This is our fast tagging algorithm.

We begin by considering $\partial\Omega_0$ which we order in a nondecreasing sequence via a heap sort algorithm. We denote $\partial\Omega_0$ as a temporary boundary set at stage 0. We shall inclusively create Ω_n and a temporary boundary set $\partial\Omega_{n,t} \subset \partial\Omega_n$ so that, after a finite number of steps the largest point in $\partial\Omega_{n,t}$ is also in $\mathcal{S}^\Delta$, at which point the algorithm terminates and Ω_n is the final Ω. At each marching step, we either tag the largest (furthest) temporary boundary point into the final boundary or turn the largest (furthest) temporary boundary

point into an exterior point. This fast tagging algorithm is of complexity $O(N \log N)$; the log N term appears because of the sorting step.

The tagging algorithm is as follows: Consider the largest $\boldsymbol{x}_{ij}^{(n)} \in \partial\Omega_{n,t}$.

(a) If there is at least one interior neighbor of $\boldsymbol{x}_{ij}^{(n)}$ which is not closer to $\mathcal{S}$, put $\boldsymbol{x}_{ij}^{(n)}$ into the tagged boundary set and define $\partial\Omega_{n+1,t} = \partial\Omega_{n,t} - \{\boldsymbol{x}_{ij}^{(n)}\}, \Omega_{n+1} = \Omega_n$.

(b) If all interior neighbors of $\boldsymbol{x}_{ij}^{(n)}$ are closer to $\mathcal{S}$, put $\boldsymbol{x}_{ij}^{(n)}$ into the exterior and include its interior neighbors into the new temporary boundary, i.e., define $\Omega_{n+1} = \Omega_n - \{\boldsymbol{x}_{ij}^{n}\}$ and $\partial\Omega_{n+1,t} = \partial\Omega_{n+1} - \{\partial\Omega_n - \partial\Omega_{n,t}\}$.

Repeat this process until either (a) the temporary boundry set becomes empty, or (b) maximum distance of the untagged temporary boundary points, (the set $\partial\Omega_{n,t}$) to $\mathcal{S}$ is less than Δ.

We now prove the algorithm is viable and converges. If condition (a) is satisfied at stage n, then since $\partial\Omega_{n+1,t} \subset \partial\Omega_{n,t}$ $\boldsymbol{x}_{ij}^{(n+1)} \leq \boldsymbol{x}_{ij}^{(n)}$. If condition (b) is satisfied then $\partial\Omega_{n+1,t}$ will include new points that are neighbors of $\boldsymbol{x}_{ij}^{(n)}$ and are closer to $\mathcal{S}$ than $\boldsymbol{x}_{ij}^{(n)}$. Thus $\boldsymbol{x}_{ij}^{(n+1)} \leq \boldsymbol{x}_{ij}^{(n)}$. This ends the proof that the algorithm is viable and converges.

Remark1: Our tagging algorithm produces a very crude approximation to the steady state solution of the convection equation, which we rewrite: $\nabla d \cdot \nabla \phi = 0$. We solve this crudely on a grid for a function $\phi_i j$ which has value either +1 or -1. We initialize so that $\phi_i j = 1$ in the exterior of $\Omega_0, \phi_i j = -1$ inside Ω_0. At every grid point $\boldsymbol{x}_i j$ to be updated we march in an "upwind" direction, which means the new $\phi_i j$ depends only on values at the four neighbors which are further from $\mathcal{S}$ than $\boldsymbol{x}_i j$. Thus we order the temporary boundary points and update the largest untagged point via a crude process

$$\phi_{ij} = P[\phi^n_{i-1,j}, \phi^n_{i+1,j}, \phi^n_{i,j-1}, \phi^n_{i,j}]$$

where P denotes a procedure that picks one of the values from its arguments that corresponds to a more remote point as follows: if there are any more remote interior points, it picks the one which is furthest from $\mathcal{S}$. Else it picks the furthest exterior point. This is equivalent to using a convex combination of either interior or exterior points to approximate ∇d and enforce ϕ to be constant along ∇d It is easy to see that this approximates the level set equation (11) and is exactly our tagging algorithm.

Remark2: At every stage of our tagging algorithm, all points $x^n_{\nu\mu}$ which are interior points of Ω_n and which are more remote than the point being tagged, x^n_{ij}, will remain in Ω_N for all $N > n$, and hence in the final set Ω, since the maximum distance on the untagged temporary boundary is decreasing. Thus we generally obtain a nontrivial limit set Ω.

Figure 2 illustrates how our fast tagging algorithm works. Starting from an arbitrary exterior region that is a subset of the final exterior region, the furthest point on the temporary boundary is tangent to a distance contour and does not have an interior point that is farther away. The furthest point will be tagged as an exterior point and the boundary will move inward at that point. Now another point on the temporary boundary becomes the furthest point and hence the whole temporary boundary moves inward. After a while the temporary boundary is close to a distance contour and moves closer and closer to the data set following the distance contours until the distance contours begin to break into spheres (circles in the 2D figure) around data points. We now see that the temporary boundary point at the breaking point of the distance contour, which is equally distant from distinct data points, will have neighboring interior points that have a larger distance. So this temporary boundary point will be tagged as a final boundary point by our procedure and the temporary boundary will stop moving inward at this breaking point. The temporary boundary starts deviating from the distance contours and continues moving closer to the data set until all temporary boundary points either have been tagged as final boundary points or are close to the data points. The final boundary is approximately a a polyhedron (polygon in 2D) with vertices belonging to the data set.

This general tagging algorithm can incorporate human interaction easily by putting any new exterior point(s) or region(s) into our tagged exterior region at any stage in our tagging algorithm. After the tagging algorithm is finished we again use the fast distance algorithm to compute a signed distance to the tagged final boundary.

The tagging method above requires an initial guess for the exterior region. This can either be the bounding box of our computational rectangular domain or an outer contour of the distance function, $d(\boldsymbol{x}) = \epsilon$. An outer contour of the distance function can be found by starting with any exterior point, such as the corners of of our rectangular domain, and expanding the exterior region by repeatedly tagging those grid points which are connected to the starting exterior point and have a distance larger than ϵ as exterior points. When the tagging algorithm is finished the boundary of the exterior region is approximately the outer contour of $d(\boldsymbol{x}) = \epsilon$ or roughly an ϵ offset of the real shape. When using this $d(\boldsymbol{x}) = \epsilon$ contour, first proposed in [31], one needs to exercise caution in choosing ϵ. For example, if ϵ is too small, we will have isolated spheres surrounding data points. If the sampling density of the data set is fine enough to resolve the real surface, then we can find an appropriate ϵ and get a very good initial surface with $O(N+M)$ operations. When combined with the above fast tagging algorithm, we can find a good initial approximation very efficiently. For non-uniform data points the intersection of a bounding box and a distance contour with moderate ϵ, which is a simple Boolean operation for implicit surfaces, often gives a good initial surface.

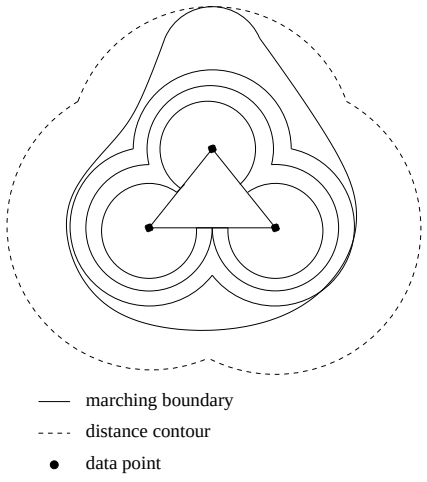


Figure 2:

5.3 Solving the partial differential equation.

After we find the distance function $d(\boldsymbol{x})$ and a good initial implicit surface using the above algorithms, we can start the continuous deformation following either the gradient flow (2) or the convection (4) using the corresponding level set formulation (6) or (7). Our numerical implementations are based on standard algorithms for the level set method. The one dimensional Delta function $\delta(x)$ and Heaviside function $H(x)$ are approximated numerically if needed.

Details can be found in, for example, [21, 30, 31]. The convection model is simple and fast but the reconstructed surface is close to a piecewise linear approximation. In contrast, the energy minimizing gradient flow, which contains a weighted curvature regularization effect, is more complicated and computationally expensive but reconstructs a smooth weighted minimal surface. These two continuous deformations can be combined, and in particular, the gradient flow can be used as a smoothing process for implicit surfaces. In most of our applications, about one hundred time steps in total are enough for our continuous deformation. Since we use a reinitialization procedure regularly during the deformation, we finish with a signed distance function for the reconstructed implicit surface.

5.4 Multiresolution

There are two scales in our surface reconstruction. One is the resolution of the data set. The other is the resolution of the grid. The computational cost generally depends mainly on the grid size. To achieve the best results those two resolutions should be comparable. However our grid resolution can be independent of the sampling density. For example, we can use a low resolution grid when there is noise and redundancy in the data set or when memory and speed are important. From our numerical results figure 9(c) our reconstruction is quite smooth even on a very low resolution grid. We can also use a multiresolution algorithm, i.e., reconstruct the surface first on coarser grids and interpolate the result to a finer resolution grid for further refinement in an hierarchical way.

5.5 Efficient storage

To store or render an implicit surface, we only need to record the values and locations (indices) of those grid points that are next to the surface, i.e., those grid points that have a different sign from at least one of their neighbors. These grid points form a thin grid shell surrounding the implicit surface. No connectivity or other information needs to be stored. We reduce the filesize by at least an order of magnitude by using this method. Moreover we can easily reconstruct the signed distance function in $O(N)$ operations for the implicit surface using the following procedure. (1) Use the fast distance finding algorithm to find the distance function using the absolute value of the stored grid shell as an initial condition. (2) Use a tagging algorithm, similar to the one used above to find exterior points outside a distance contour, to identify all exterior points and interior points separated by the stored grid shell and turn the computed distance into the signed distance. For example, if we store the signed distance funtion for our reconstructed Happy Buddha from almost half a million points on a $146 \times 350 \times 146$ grid in binary form, the file size is about 30MB. If we use the above efficient way of storage the file size is reduced to 2.5MB without using any compression procedure and we can reconstruct the signed distance function in 1 minute using the above algorithm .

6 Results

In this section we present a few numerical examples that illustrate the efficiency and quality of our surface construction. In particular we show (1) how the level set method handles surface deformation and topological change easily, (2) how quickly our tagging algorithm constructs a good initial guess, (3) how smooth the reconstructed surfaces are by using either the convection model or the minimal surface model, (4) how our algorithm works with non-uniform, noisy or damaged data, and (5) how multiresolution works in our formulation. All calculations were done with a Pentium III, 600Mhz processor. Data points for the drill, the dragon and the Buddha were obtained from www-graphics.stanford.edu/data/3Dscanrep and data points for the hand skeleton and turbine blade were obtained from www.cc.gatech.edu/projects/large_models. Only locations of the data points are used in our reconstructions.

The first group of examples show surface reconstruction from synthesized data. Figure 4 show surface reconstruction, a torus, from damaged data, which is like hole filling. Figure 5 shows the reconstruction of a sphere from a box using eight longitudinal circles and eight latitudinal circles. For this example we do not have any discrete data points. We only provide the unsigned distance function. This can also be viewed as an extreme case of non-uniform data. Figure 5(a) shows those circles and figure 5(b) shows reconstruction using the convection model. Figure 5(c) shows the final minimal surface reconstruction following the gradient flow on top of figure 5(b).

The second group of examples are from real data. Timings, number of data points and grid size are shown in table 3. CPU time is measured in minutes. CPU (initial) is the time for the initial reconstruction using the distance contour and the fast tagging algorithm. CPU (total) is the total time used for the reconstruction. Since our PDE based algorithms are iterative procedures, different convergence criterion will give different convergence times. For data sets that are fairly uniform, such as the drill, the dragon, the Buddha and the hand skeleton, we start with an outer distance contour and use the fast tagging algorithm to get an initial reconstruction. The initial reconstruction is extremely fast, as we can see from table 3. After the initial reconstruction, we first use the convection model and then use the gradient flow to finish the final reconstruction. In our reconstruction, the grid resolution is much lower than the data samples and yet we get final results that are comparable to the reconstructions shown at those websites above.

Figure 6 shows the reconstruction for a rat brain from MRI slices. The data set is very non-uniform and noisy. We start with the intersection of a bounding box and an outer distance contour with relatively large $\epsilon = 12h$, which is shown in figure 6(b). The next example, figure 7 is our reconstruction of a 1.6mm drill bit from 1961 scanned data points. It is a quite challenging example for most methods for surface reconstruction from unorganized data as is shown in [11]. Figure 8 shows the reconstruction of a hand skeleton. Figure 9 shows the reconstruction of the Happy Buddha. Figure 9(a) shows the initial reconstruction using the fast tagging algorithm only. We start with an outer distance contour, $d = 3h$, initially and it takes only 3 minutes for half a million points on a $146 \times 350 \times 146$ grid. Figure 10 is the reconstruction of the dragon. Figure 9(b) is the final reconstruction. Figure 9(c) is the reconstruction on a much under resolved coarse grid $63 \times 150 \times 64$ using the same amount of data points. It only takes 7 minutes and the result is quite good. For the example of the dragon, we show the initial reconstruction in figure 10(a), reconstruction using the convection model only in figure 10(b) and the final weighted minimal surface reconstruction in figure 10(c). Figure 10(d) shows the reconstruction using a much lower resolution data set on the same grid and the result is quite comparable to figure 10(c). The final example shows the reconstruction of a turbine blade on a $178 \times 299 \times 139$ grid for almost a million data points.

7 Conclusions

We present a variational and PDE based formulation for surface reconstruction from unorganized data. Our formulation only depends on the (unsigned) distance function to the data and the final reconstruction is smoother than piecewise linear. We use the level set method as a numerical tool to deform and construct implicit surfaces on fixed rectangular grids. We use fast sweeping algorithms for computing the distance function and fast tagging algorithms for

Model	Data points	Grid size	CPU (initial)	CPU (total)
Rat brain	1506	80x77x79	.12	3
Drill	1961	24x250x32	0.1	2
Buddha	543652	146x350x146	3	68
Buddha	543652	63x150x64	.3	7
Dragon	437645	300x212x136	4	77
Dragon	100250	300x212x136	3	66
Hand	327323	200x141x71	.5	10
Turbine blade	882954	178x299x139	2.5	60

Figure 3: timing table

initial construction. Our method works for complicated topology and non uniform or noisy data.

References

[1] D. Adalsteinsson and J.A. Sethian. A fast level set method for propagating interfaces. *J. Comp. Phys.*, 118(2):269–277, 1995.

[2] L. Alvarez, F. Guichard, P.-L. Lions, and J.-M. Morel. Axioms and fundamental equations of image processing. *Arch. Rat. Mechanics*, 123:199–257, 1993.

[3] N. Amenta and M. Bern. Surface reconstruction by Voronoi filtering. *14th ACM Symposium on Computational Geometry*, 1998.

[4] N. Amenta, M. Bern, and D. Eppstein. The crust and the β-skeleton: combinatorial curve reconstruction. *Graphical Models and Image Processing*, 60/2(2):125–135, 1998.

[5] N. Amenta, M. Bern, and M. Kamvysselis. A new Voronoi-based surface reconstruction algorithm. *Proc. SIGGRAPH'98*, pages 415–421, 1998.

[6] C. Bajaj, F. Bernardini, and G. Xu. Automatic reconstruction of surfaces and scalar fields from 3d scans. *SIGGRAPH'95 Proceedings*, pages 193–198, July 1995.

[7] J. Bloomenthal, C. Bajaj, J. Blinn, M.-P Cani-Gascuel, A. Rockwood, B. Wyvill, A. Rockwood, B. Wyvill, and G. Wyvill. *Introduction to implicit surfaces*. Morgan Kaufman, Inc., San Francisco, 1997.

[8] J.D. Boissonnat. Geometric structures for three dimensional shape reconstruction. *ACM Trans. Graphics 3*, pages 266–286, 1984.

[9] J.D. Boissonnat and F. Cazals. Smooth shape reconstruction via natural neighbor interpolation of distance functions. *ACM Symposium on Computational Geometry*, 2000.

[10] M. Boué and P. Dupuis. Markov chain approximations for deterministic control problems with affine dynamics and quadratic cost in the control. *SIAM J. Numer. Anal.*, 36(3):667–695, 1999.

[11] B. Curless and M. Levoy. A volumetric method for building complex models from range images. *SIGGRAPH'96 Proceedings*, pages 303–312, 1996.

[12] H. Edelsbrunner. Shape reconstruction with Delaunay complex. In *Proc. of LATIN'98: Theoretical Informatics*, volume 1380 of *Lecture Notes in Computer Science*, pages 119–132. Springer-Verlag, 1998.

[13] H. Edelsbrunner and E. P. Mücke. Three dimensional α shapes. *ACM Trans. Graphics 13*, pages 43–72, 1994.

[14] L.C. Evans and R.F. Gariepy. Measure theory and fine properties of functions. *Studies in Advanced Mathematics, CRC Press Inc.*, 1992.

[15] A. Hilton, A.J. Stoddart, J. Illingworth, and T. Windeatt. Implicit surface - based geometric fusion. *Comput. Vision and Image Understanding*, 69:273–291, 1998.

[16] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. *SIGGRAPH'92 Proceedings*, pages 71–78, 1992.

[17] W.E. Lorenson and H.E. Cline. Marching cubes: A high resolution 3d surface recomstruction algorithm. *Computer Graphics*, 21:163–169, 1987.

[18] S. Muraki. Volumetric shape description of range data using "blobby model". In *Computer Graphics (Proc. SIGGRAPH)*, volume 25, pages 227–235, July 1991.

[19] S. Osher and R. Fedkiw. Level set methods: an overview and some recent results. *to appear in J. Comp. Phys.*, 2000.

[20] S. Osher and J. Sethian. Fronts propagating with curvature dependent speed, algorithms based on a Hamilton-Jacobi formulation. *J. Comp. Phys.*, 79:12–49, 1988.

[21] D. Peng, B. Merriman, S. Osher, H.K. Zhao, and M. Kang. A PDE based fast local level set method. *J. Comp. Phys.*, 155:410–438, 1999.

[22] L. Piegl and W. Tiller. *The NURBS book*. Berlin, Germany: Springer-Verlag, 2nd edition edition, 1996.

[23] D.F. Rogers. *An Introduction to NURBS*. Morgan Kaufmann, 2000.

[24] L. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D*, 60:259–268, 1992.

[25] M. Sussman, P. Smereka, and S. Osher. A level set approach for computing solutions to incompressible two-phase flows. *J. Comp. Phys.*, 119:146–159, 1994.

[26] J.N. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Transactions on Automatic Control*, 40((9)):1528–1538, 1995.

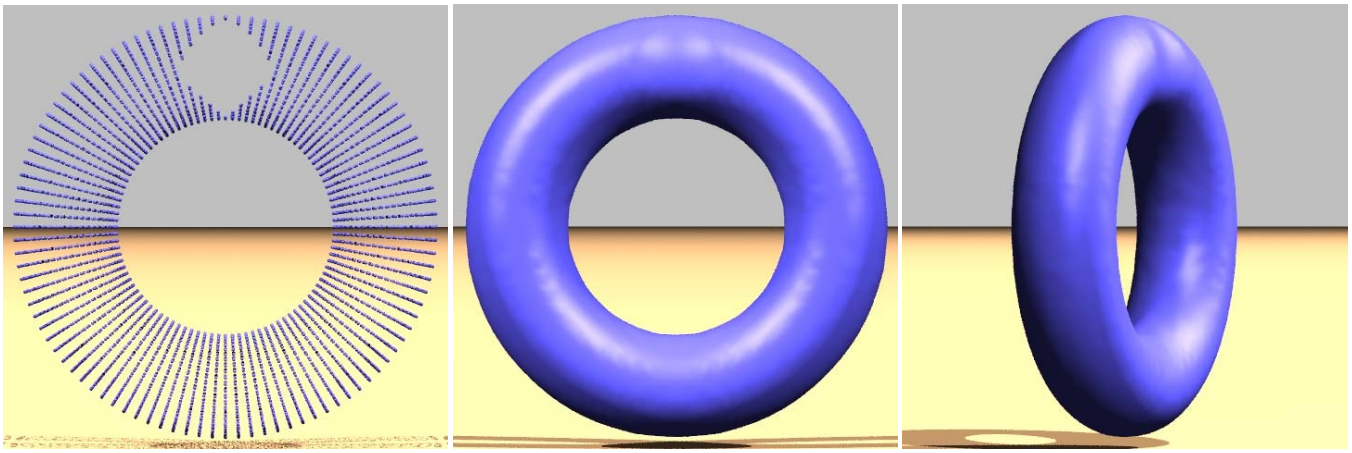
[27] G. Turk and J. ÒBrien. Shape transformation using variational implicit functions. *SIGGRAPH99*, pages 335–342, August 1999.

[28] R. Whitaker. A level set approach to 3D reconstruction from range data. *International journal of Computer Vision*, 1997.

[29] A.P. Witkin and P.S. Heckbert. Using particles to sample and control implicit surfaces. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH94)*, pages 269–278, July 1994.

[30] H.K. Zhao, T.F. Chan, B. Merriman, and S.Osher. A variational level set approach to multiphase motion. *J. Comp. Phys.*, 127:179–195, 1996.

[31] H.K. Zhao, S. Osher, B. Merriman, and M. Kang. Implicit and non-parametric shape reconstruction from unorganized points using variational level set method. *Computer Vision and Image Understanding*, 80(3):295–319, 2000.

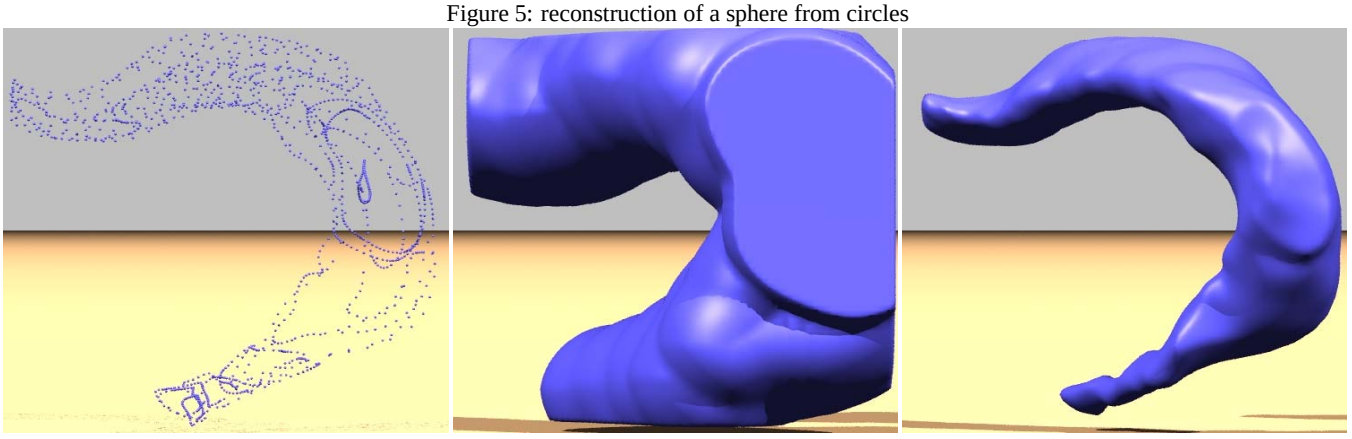


(a) data points (b) front view of the final reconstruction (c) side view of the final reconstruction

Figure 4: hole filling of a torus

(a) initial data (b) reconstruction using convection (c) final reconstruction

Figure 5: reconstruction of a sphere from circles



(a) data points (b) starting surface (c) final reconstruction

Figure 6: reconstruction of a rat brain

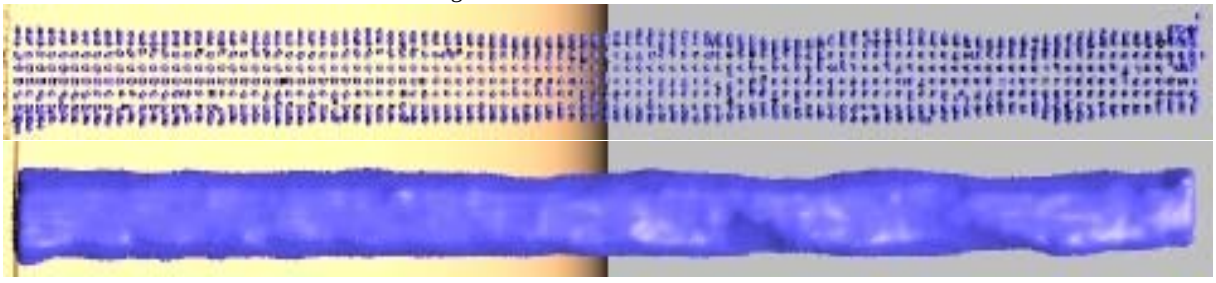
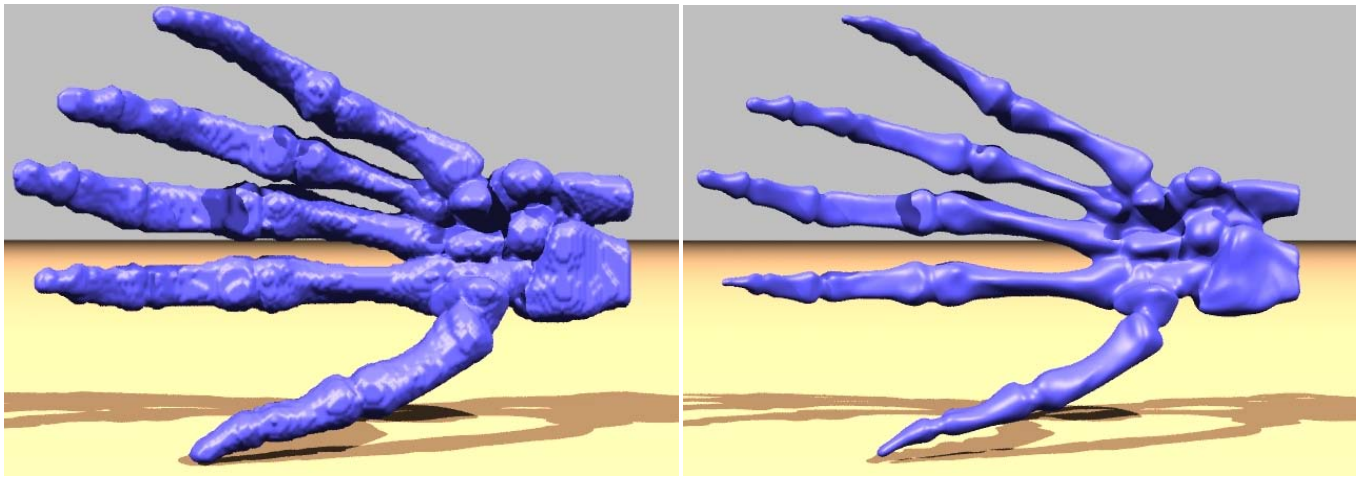


Figure 7: reconstruction of a drill

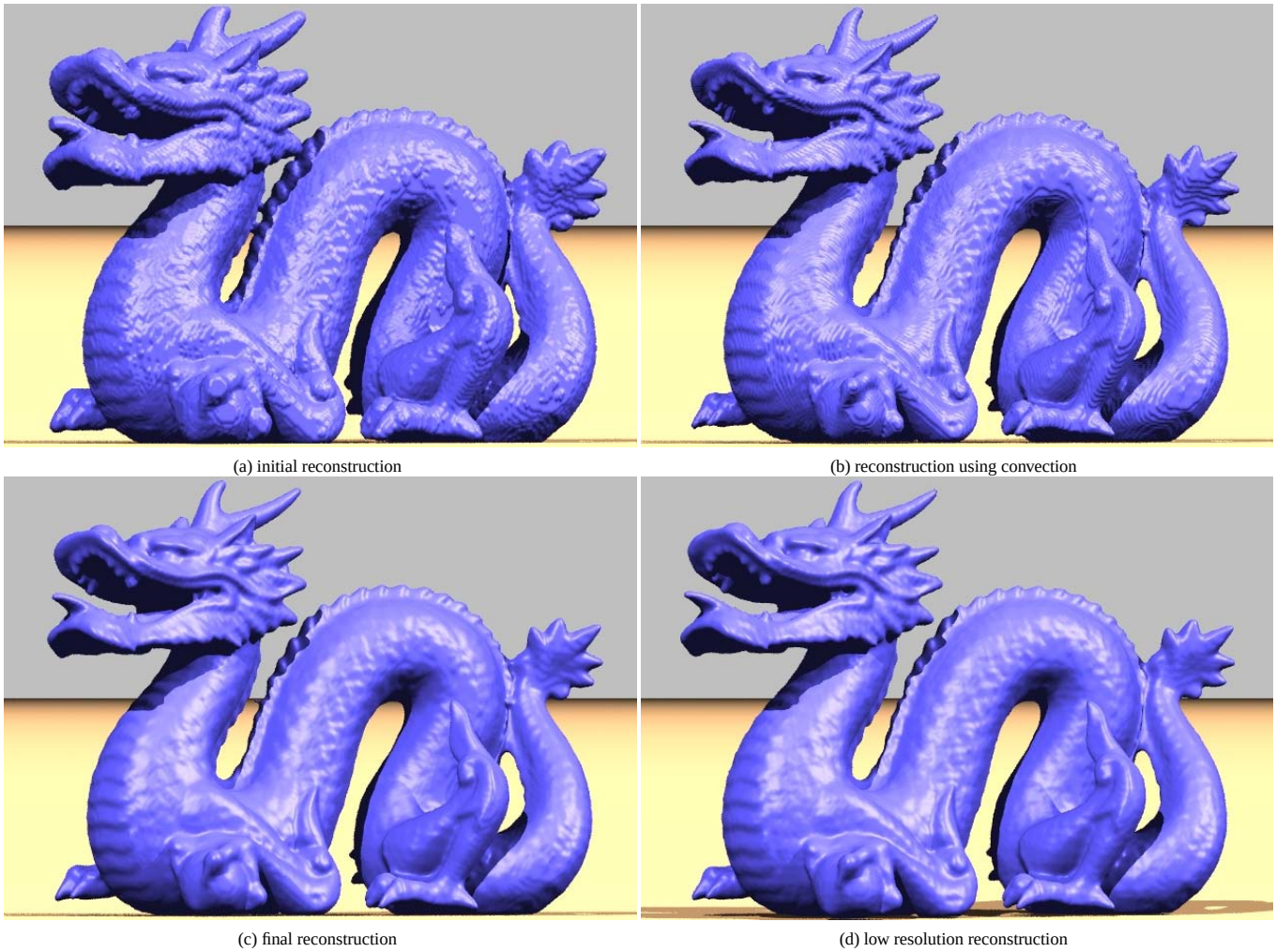


(a) initial reconstruction (b) final reconstruction

Figure 8: reconstruction of a hand skeleton

(a) initial reconstruction (b) final reconstruction (c) reconstruction on a coarse grid

Figure 9: reconstruction of the Happy Buddha



(a) initial reconstruction

(b) reconstruction using convection

(c) final reconstruction

(d) low resolution reconstruction

Figure 10: reconstruction of the dragon

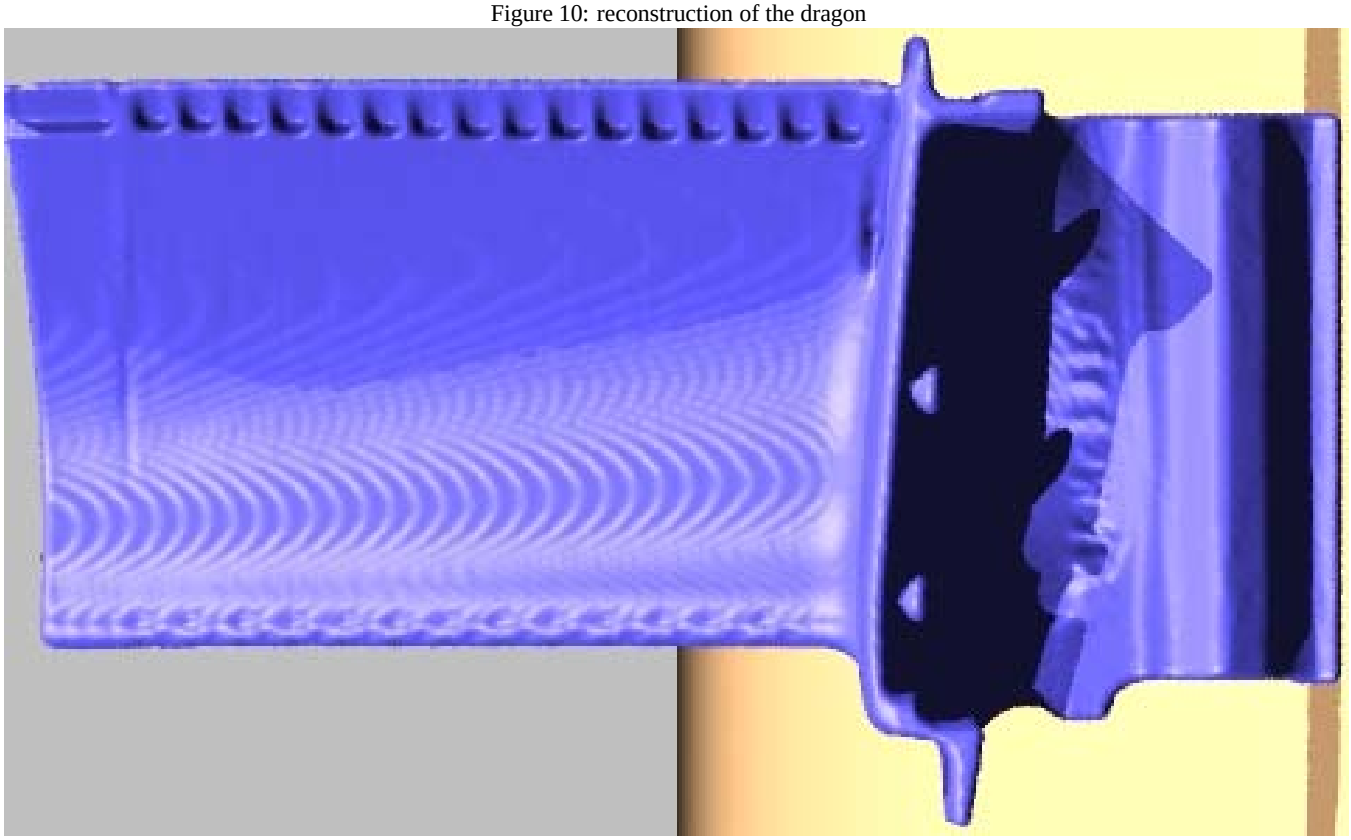


Figure 11: reconstruction of a turbine blade

Isosurfaces and Level-Set Surface Models [a]

Ross T. Whitaker

Technical Report UUCS-02-010

[a] Versions of these notes and the accompanying talk appeared in tutorials at IEEE Visualization 2000 and 2001, and ACM SIGGRAPH 2001 and 2002.

School of Computing
University of Utah
Salt Lake City, UT 84112 USA

April 3, 2002

Abstract

This paper is a set of notes that present the basic geometry of isosurfaces and the basic methods for using level sets to model deformable surfaces. It begins with a short introduction to isosurface geometry, including curvature. It continues with a short explanation of the level-set partial differential equations. It also presents some practical details for how to solve these equations using up-wind scheme and sparse calculation methods. This paper presents a series of examples of how level-set surface models are used to solve problems in graphics and vision. Finally, it presents some examples of implementations using *VISPack*, an object oriented, C++ library for doing volume processing and level-set surface modeling.

1 Introduction

1.1 Motivation

These notes address mechanisms for analyzing and processing volumes in a way that deals specifically with *isosurfaces*. The underlying philosophy is to use isosurfaces as a modeling technology that can serve as an alternative to parameterized models for a variety of important applications in visualization and computer graphics. This paper presents the mathematics and numerical techniques for describing the geometry of isosurfaces and manipulating their shapes in prescribed ways. We start with a basic introduction into the notation and fundamental concepts and then presents the geometry of isosurfaces. We describe the method of level sets, i.e., moving isosurfaces, and present the mathematical and numerical methods they entail. This paper concludes with some application examples and describes *VISPACK*, a *C++*, object-oriented library the performs volume processing and level-set modeling.

1.2 Isosurfaces

1.2.1 Modeling Surfaces With Volumes

When considering surface models for graphics and visualization, one is faced with a staggering variety of options including meshes, spline-based patches, constructive solid geometry, implicit blobs, and particle systems. These options can be divided into two basic classes — explicit (parameterized) models and implicit models. With an implicit model, one specifies the model as a *level set* of a scalar function,

$$\begin{array}{rccc} \phi: & U & \mapsto & \mathbb{R} \\ & x, y, z & & k \end{array}, \tag{1}$$

where $U \subset \mathbb{R}^3$ is the domain of the volume (and the *range* of the surface model). Thus, a surface $\mathcal{S}$ is

$$\mathcal{S} = \{\boldsymbol{x} | \phi(\boldsymbol{x}) = k\} . \tag{2}$$

The choice of k is arbitrary, and ϕ is sometimes called the *embedding*. Notice that surfaces defined in this way divide U into a clear inside and outside—such surfaces are always closed wherever they do not intersect the boundary of the domain.

Choosing this implicit strategy begs the question of how to represent ϕ. Historically, implicit models are represented using linear combinations of *basis* functions. These basis or

potential functions usually have several degrees of freedom including 3D position, size, and orientation. By combining these functions, one can create complex objects. Typical models might contain several hundred to several thousands of such primitives. This is the strategy behind the "blobby" models proposed by Blinn [1].

While such an implicit modeling strategy offers a variety of new modeling tools, it has some limitations. In particular, the global nature of the potential functions limits ones ability to model *local* surface deformations. Consider a point $\boldsymbol{x} \in \mathcal{S}$ where $\mathcal{S}$ is the level surface associated with a model $\phi = \sum_i \alpha_i$, and α_i is one of the individual potential functions that comprise that model. Suppose one wishes to move the surface at the point $\boldsymbol{x}$ in a way that maintains continuity with the surrounding neighborhood. With multiple, global basis functions one must decide which basis function or combination of basis functions to alter and at the same time control the effects on other parts of the surface. The problem is generally ill posed — there are many ways to adjust the basis functions so that $\boldsymbol{x}$ will move in the desired direction and yet it may be impossible to eliminate the effects of those movements on other disjoint parts of the surface. These problems can be overcome, however they usually entail heuristics that tie the behavior of the surface deformation to the choice of representation [2].

An alternative to using a small number of *global* basis functions is to use a relatively large number of *local* basis functions. This is the principle behind using a volume as an implicit model. A volume is a discrete sampling of the embedding ϕ. It is also an implicit model with a very large number of basis functions, as shown in Figure 1. The total number of basis functions is fixed, as are their positions (grid points) and extent. One can change only the magnitude of each basis function, i.e., each basis function has only one degree of freedom. A typical volume of size $128 \times 128 \times 128$ contains over a million such basis functions. The shape of each basis function is open to interpretation — it depends on how one interpolates the values between the grid points. A trilinear interpolation, for instance, implies a basis function that is a piece-wise cubic polynomial with a value of one at the grid point and zero at neighboring grid points. Another advantage of using volumes as implicit models, is that for the purposes of analysis we can treat the volume as a continuous function whose values can be *set* at each point according to the application. Once the continuous analysis is complete we can map the algorithm into the discrete domain using standard methods of numerical analysis. The sections that follow discuss how to compute the geometry of surfaces that are represented as volumes and how to manipulate the shapes of those surfaces by changing the gray-scale values in the volume.

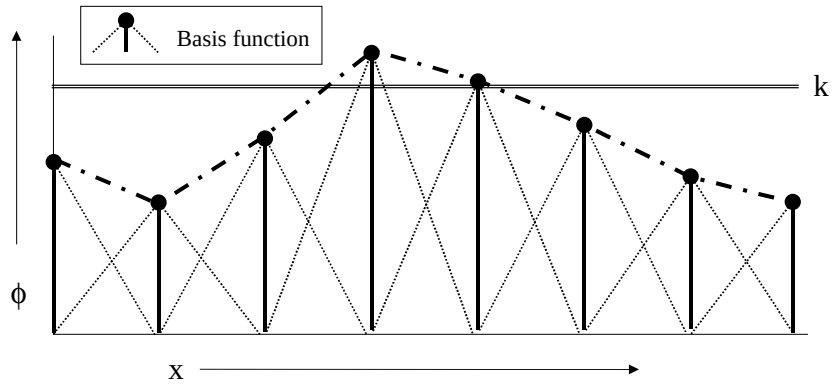


Figure 1: A volume can be considered as an implicit model with a large number of local basis functions.

1.2.2 Isosurface Extraction and Visualization

This paper addresses the question of how to use volumes as surface models. Depending on the application, however, a 3D grid of data (i.e. a volume) may not be a suitable model representation. For instance, if the goal is make measurements of an object or visualize its shape, an explicit model might be necessary. In such cases it is beneficial to convert between volumes and other modeling technologies.

For instance, the literature proposes several methods for scan converting polygonal meshes or solid models [3, 4]. Likewise a variety of methods exist for extracting parametric models of isosurfaces from volumes. The most prevalent method is to locate isosurface crossings along grid lines in a volume (between voxels along the 3 cardinal directions) and then to link these points together to form triangles and meshes. This is the strategy of "marching cubes" [5] and other related approaches. However, extracting a parametric surface is not essential for visualization, and a variety of direct methods [6, 7] are now computationally feasible and arguably superior in quality. These notes do not address the issue of extracting or rendering isosurfaces, but rather studies the geometry of isosurfaces and how to manipulate them directly by changing the grey-scale values in the underlying volume. Thus, we propose volumes as a mechanism for studying and deforming surfaces, regardless of the ultimate form of the output. Their are many ways of rendering or visualizing them and and these techniques are beyond the scope of this discussion.

2 Surface Normals

The surface normal of an isosurface is given by the normalized gradient vector. Typically, we identify a surface normal with a point in the volume domain D. That is

$$\boldsymbol{n}(\boldsymbol{x}) = \frac{\nabla\phi(\boldsymbol{x})}{|\nabla\phi(\boldsymbol{x})|} \quad \text{where} \quad \boldsymbol{x} \in D. \tag{3}$$

The convention regarding the direction of this vector is arbitrary; the negative of the normalized gradient magnitude is also normal to the isosurface. The gradient vector points toward that side of the isosurface which has greater values (i.e. brighter). When rendering, the convention is to use *outward pointing* normals, and the sign of the gradient must be adjusted accordingly. However, for most applications any consistent choice of normal vector will suffice. On a discrete grid, one must also decide how to approximate the gradient vector (i.e., first partial derivatives). In many cases central differences will suffice. However, in the presence of noise, especially when volume rendering, it is sometimes helpful to compute first derivatives using some smoothing filter (e.g., convolution with a Gaussian). When

using the normal vector to solve certain kinds of partial differential equations, it is sometimes necessary to approximate the gradient vector with discrete, one-sided differences, as discussed in successive sections.

Note that a single volume contains families nested isosurfaces, arranged like the layers of an onion. We specific the normal to an isosurface as a function of the position within the volume. That is, $\boldsymbol{n}(\boldsymbol{x})$ is the normal of the (single) isosurface that passes through the point $\boldsymbol{x}$. The k value associated with that isosurface is $\phi(\boldsymbol{x})$.

3 Second-Order Structure

In differential geometric terms, the second-order structure of a surface is characterized by a quadratic patch that shares first- and second-order contact with the surface at a point (i.e., tangent plane and osculating circles). The *principal directions* of the surface are those associated with the quadratic approximation, and the *principal curvatures*, k_1, k_2, are the curvatures in those directions.

The second-structure of the isosurface can be computed from the first- and second-order structure of the embedding, ϕ. All of the isosurface shape information is contained field of normals given by $\boldsymbol{n}(\boldsymbol{x})$. The 3×3 matrix of derivatives of this vector,

$$N = -\left[\boldsymbol{n}_x \;\; \boldsymbol{n}_y \;\; \boldsymbol{n}_z.\right] \tag{4}$$

The projection of this derivative onto the tangent plane of the isosurface gives the shape matrix, β. Let P denote normal projection operator, which is defined as

$$P = \boldsymbol{n} \otimes \boldsymbol{n} = \frac{1}{||\nabla\phi||^2} \begin{pmatrix} \phi_x^2 & \phi_x\phi_y & \phi_x\phi_z \\ \phi_y\phi_x & \phi_y^2 & \phi_y\phi_z \\ \phi_z\phi_x & \phi_z\phi_y & \phi_z^2 \end{pmatrix}. \tag{5}$$

The tangential projection operator is $I - P$, and thus the shape matrix is

$$\beta = NT = TH_\phi T, \tag{6}$$

where H_ϕ is the Hessian of ϕ. The shape matrix β has 3, real, eigenvalues which are

$$e_1 = k_1, e_2 = k_2, e_3 = 0. \tag{7}$$

The corresonding eigenvectors are the principle directions (in the tangent plane) and the normal, respectively.

The *mean curvature* is the mean of the two principal curvatures, which is one half of the trace of β, which is equal to the trace of N:

$$\begin{aligned} H &= \frac{k_1 + k_2}{2} = \frac{1}{2}\mathrm{Tr(N)} \\ &= \frac{\phi_x^2(\phi_{yy} + \phi_{zz}) + \phi_y^2(\phi_{xx} + \phi_{zz}) + \phi_z^2(\phi_{xx} + \phi_{yy}) - 2\phi_x\phi_y\phi_{xy} - 2\phi_x\phi_z\phi_{xz} - 2\phi_y\phi_z\phi_{yz}}{2(\phi_x^2 + \phi_y^2 + \phi_z^2)^{3/2}} \end{aligned} \tag{8}$$

The *Gaussian curvature* is the product of the principal curvatures:

$$\begin{aligned} K &= k_1 k_2 = e_1 e_2 + e_1 e_3 + e_2 e_3 = 2\mathrm{Tr(N)}^2 - \frac{1}{2}||\mathrm{N}|| \\ &= \frac{\begin{array}{c}\phi_z^2(\phi_{xx}\phi_{yy} - \phi_{xy}\phi_{xy}) + \phi_y^2(\phi_{xx}\phi_{zz} - \phi_{xz}\phi_{xz}) + \phi_x^2(\phi_{yy}\phi_{zz} - \phi_{yz}\phi_{yz}) \\ + 2(\phi_x\phi_y(\phi_{xz}\phi_{yz} - \phi_{xy}\phi_{zz}) + \phi_x\phi_z(\phi_{xy}\phi_{yz} - \phi_{xz}\phi_{yy}) + \phi_y\phi_z(\phi_{xy}\phi_{xz} - \phi_{yz}\phi_{xx}))\end{array}}{(\phi_x^2 + \phi_y^2 + \phi_z^2)^2}. \end{aligned} \tag{9}$$

The total curvature, also called the deviation from flatness, D, is the root sum of squares of the two principal curvatures, which is the Euclidean norm of the matrix β.

Notice, these measures exist at every point in U, and at each point they describe the geometry of the particular isosurface that passes through that point. All of these quantities can be computed on a discrete volume using finite differences, as described in successive sections.

4 Deformable Surfaces

This section begins with mathematics for describing surface deformations on parametric models. The result is an evolution equation for a surface. Each of the terms in this evolution equation can be re-expressed in a way that is independent of the parameterization. Finally, the evolution equation for a parametric surface gives rise to an evolution equation (differential equation) on a volume, which encodes the shape of that surface as a level set.

4.1 Surface Deformation

A regular surface $\mathcal{S} \subset \mathbb{R}^3$ is a collection of points in 3D that can be be represented *locally* as a continuous function. In geometric modeling a surface is typically represented as a two-parameter object in a three-dimensional space, i.e., a surface is local a mapping $\boldsymbol{S}$:

$$\boldsymbol{S} : \underset{r}{V} \times \underset{s}{V} \mapsto \underset{x,\, y,\, z,}{\mathbb{R}^3} \tag{10}$$

where $V \times V\mathbb{R}^2$, and the bold notation refers specifically to a parameterized surface (vector-valued function). A deformable surface exhibits some motion over time. Thus $\boldsymbol{S} = \boldsymbol{S}(r, s, t)$, where $t \in \mathbb{R}^+$. We assume second-order-continuous, orientable surfaces; therefore at every point on the surface (and in time) there is surface normal $\boldsymbol{N} = \boldsymbol{N}(r, s, t)$. We use $\mathcal{S}_t$ to refer to the entire set of points on the surface.

Local deformations of $\boldsymbol{S}$ can be described by an evolution equation, i.e., a differential equation on $\boldsymbol{S}$ that incorporates the position of the surface, local and global shape properties, and responses to other forcing functions. That is,

$$\frac{\partial \boldsymbol{S}}{\partial t} = \boldsymbol{G}\left(\boldsymbol{S}, \boldsymbol{S}_r, \boldsymbol{S}_s, \boldsymbol{S}_{rr}, \boldsymbol{S}_{rs}, \boldsymbol{S}_{ss}, \ldots\right), \tag{11}$$

where the subscripts represent partial derivatives with respect to those parameters. The evolution of $\boldsymbol{S}$ can be described by a sum of terms that depends on both the geometry of $\boldsymbol{S}$ and the influence of other functions or data.

There are a variety of differential expressions that can be combined for different applications. For instance, the model could move in response to some directional "forcing" function [8, 9], $\boldsymbol{F} : U \mapsto \mathbb{R}^3$, that is

$$\frac{\partial \boldsymbol{S}}{\partial t} = \boldsymbol{F}(\boldsymbol{S}). \tag{12}$$

Alternatively, the surface could expand and contract with a spatially-varying speed. For instance,

$$\frac{\partial \boldsymbol{S}}{\partial t} = G(\boldsymbol{S})\boldsymbol{N} \tag{13}$$

where $G : \mathbb{R}^3 \mapsto \mathbb{R}$ is a signed speed function. The evolution might also depend on the surface geometry itself. For instance,

$$\frac{\partial \boldsymbol{S}}{\partial t} = \boldsymbol{S}_{rr} + \boldsymbol{S}_{ss} \tag{14}$$

describes a surface that moves in way that is becomes more *smooth* with respect to its own parameterization. This motion can be combined with the motion of Equation 12 to produce a model that is pushed by a forcing function but maintains a certain smoothness in its shape and parameterization. There are myriad terms that depend on both the differential geometry of the surface and outside forces or functions to control the evolution of a surface.

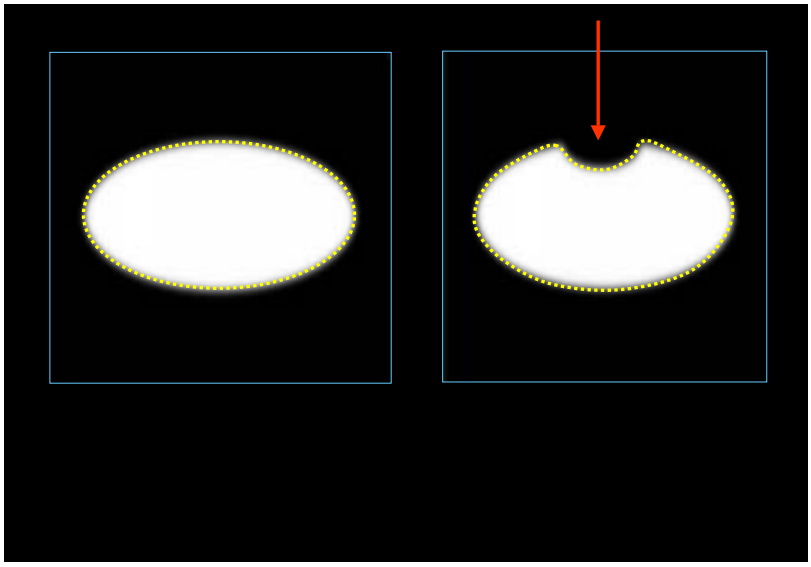


Figure 2: Level-set models represent curves and surfaces implicitly using greyscale images: a) an ellipse is represented as the level set of an image, b) to change the shape we modify the greyscale values of the image.

5 Deformation: The Level Set Approach

The method of level-sets, proposed by Osher and Sethian [10] and described extensively in [11], provides the mathematical and numerical mechanisms for computing surface deformations as time-varying iso-values of ϕ by solving a partial differential equation on the 3D grid. That is, the level-set formulation provides a set of numerical methods that describe how to manipulate the greyscale values in a volume, so that the isosurfaces of ϕ move in a prescribed manner (shown in Figure 2).

We denote the movement of a point on a surface as it deforms as $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$, and we assume that this motion can be expressed in terms of the position of $\boldsymbol{x} \in U$ and the geometry of the surface at that point. In this case, there are generally two options for representing such surface movements implicitly:

Static: A single, static $\phi(\boldsymbol{x})$ contains a family of level sets corresponding to surfaces as different times t. That is,

$$\phi(\boldsymbol{x}(t)) = k(t) \quad \Rightarrow \quad \nabla\phi(\boldsymbol{x}) \cdot \frac{\partial \boldsymbol{x}}{t} = \frac{\mathrm{d}k(t)}{\mathrm{d}t}. \tag{15}$$

To solve this static method requires constructing a ϕ that satisfies equation 15. This is a boundary value problem, which can be solved somewhat efficiently starting with a single surface using the fast marching method of Sethian [12]. This representation has some significant limitations, however, because (by definition) a surface cannot pass back over itself over time, i.e., motions must be strictly monotonic — inward or outward.

Dynamic: The approach is to use a one-parameter *family* of embeddings, i.e., $\phi(\boldsymbol{x}, t)$ changes over time, $\boldsymbol{x}$ remains on the k level set of ϕ as it moves, and k remains constant. The behavior of ϕ is obtained by setting the total derivative of $\phi(\boldsymbol{x}(t), t) = k$ to zero. Thus,

$$\phi(\boldsymbol{x}(t), t) = k \;\; \Rightarrow \;\; \frac{\partial \phi}{\partial t} = -\nabla \phi \cdot \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t}. \tag{16}$$

This approach can accommodate models that move forward and backward and cross back over their own paths (over time). However, to solve this requires solving the initial value problem (using finite forward differences) on $\phi(\boldsymbol{x}, t)$ — a potentially large computational burden. The remainder of this discussion focuses on the dynamic case, because of its superior flexibility.

All surface movements depend on position and geometry, and the level-set geometry is expressed in terms of the differential structure of ϕ. Therefore the dynamic formulation from equation 16 gives a general form of the partial differential equation on ϕ:

$$\frac{\partial \phi}{\partial t} = -\nabla \phi \cdot \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = -\nabla \phi \cdot \boldsymbol{F}(\boldsymbol{x}, \mathrm{D}\phi, \mathrm{D}^2\phi, \ldots), \tag{17}$$

where $D^n\phi$ is the set of order-n derivatives of ϕ evaluated at $\boldsymbol{x}$. Because this relationship applies to every level-set of ϕ, i.e. all values of k, this equation can be applied to all of U, and therefore the movements of *all* the level-set surfaces embedded in ϕ can be calculated from Equation 17.

The level-set representation has a number of practical and theoretical advantages over conventional surface models, especially in the context of deformation and segmentation. First, level-set models are topologically flexible, they can easily represent complicated surface shapes that can, in turn, form holes, split to form multiple objects, or merge with other objects to form a single structure. These models can incorporate many (millions) of degrees of freedom, and therefore they can accommodate complex shapes. Indeed, the shapes formed by the level sets of ϕ are restricted only by the resolution of the sampling. Thus, there is no need to reparameterize the model as it undergoes significant deformations.

Such level-set methods are well documented in the literature [10, 13] for applications such as computational physics [14], image processing [15, 16], computer vision [17, 18], medical image analysis [19, 18], and 3D reconstruction [20, 21]. For instance, in computational

physics level-set methods are a a powerful tool for modeling moving interfaces between different materials (see Osher and Fedkiw [14] for a nice overview of recent results). Examples are water-air and water-oil. In such cases, level-set methods can be used to compute deformations that minimize surface area while preserving volumes for materials that *split and merge* in arbitrary ways. The method can be extended to multiple, non-overlapping objects.

Level-set methods have also been shown to be effective in extracting surface structures from biological and medical data. For instance Malladi *et al.* [18] propose a method in which the level-sets form an expanding or contracting contour which tends to "cling" to interesting features in 2D angiograms. At the same time the contour is also influenced by its own curvature, and therefore remains smooth. Whitaker *et al.* [19, 22] have shown that level sets can be used to simulate conventional deformable surface models, and demonstrated this by extracting skin and tumors from *thick-sliced* (e.g. clinical) MR data, and by reconstructing a fetal face from 3D ultrasound. A variety of authors [23, 24, 16, 25] have presented variations on the method and presented results for 2D and 3D data. Sethian [11] gives several examples of level-set curves and surface for segmenting CT and MR data.

5.1 Deformation Modes

In the case of parametric surfaces, one can choose from a variety of different expressions to construct an evolution equation that is appropriate for a particular application. For each of those parametric expressions, there is a corresponding expression that can be formulated on ϕ, the volume in which the level-set models are embedded. In constructing evolutions on levels sets, there can be no reference to the underlying surface parameterization (terms depending on r and s in Equations 10 through 14). This has two important implications: 1) only those surface movements that are normal to the surface are represented—any other movement is equivalent to a reparameterization 2) all of the derivatives with respect to surface parameters r and s must be expressed in terms of invariant surface properties that can be derived without a parameterization.

Consider the term $\boldsymbol{S}_{rr} + \boldsymbol{S}_{ss}$ from equation 14. If r, s is an orthonormal parameterization, the effect of that term is based purely on surface shape, not on the parameterization, and the expression $\boldsymbol{S}_{rr} + \boldsymbol{S}_{ss}$ is twice the *mean curvature*, H, of the surface. The corresponding level-set formulation is given by Equation 8.

Table 1 shows a list of expressions used in the evolution of parameterized surfaces and their equivalents for level-set representations. Also given are the assumptions about the parameterization that give rise to the level-set expressions.

	Effect	Parametric Evolution	Level-Set Evolution	Parameter Assumptions
1	External force	$\boldsymbol{F}$	$\boldsymbol{F} \cdot \nabla\phi$	None
2	Expansion/ contraction	$G(\boldsymbol{x})\boldsymbol{N}$	$G(\boldsymbol{x})\|\nabla\phi(\boldsymbol{x},t)\|$	None
3	Mean curvature	$S_{rr} + S_{ss}$	$H\|\nabla\phi\|$	Orthonormal
4	Gauss curvature	$S_{rr} \times S_{ss}$	$K\|\nabla\phi\|$	Orthonormal
5	Second order	S_{rr} or S_{ss}	$\left(H \pm \sqrt{H^2 - K}\right)\|\nabla\phi\|$	Principal curvatures

Table 1: A list of evolution terms for parametric models has a corresponding expression on the embedding, ϕ, associated with the level-set models.

6 Numerical Methods

By taking the strategy of embedding surface models in volumes, we have converted equations that describe the movement of surface points to nonlinear, partial differential equations defined on a volume, which is generally a rectilinear grid. The expression $u^n_{i,j,k}$ refers to the nth time step at position i, j, k, which has an associated value in the 3D domain of the continuous volume $\phi(x_i, y_j, z_k)$. The goal is to solve the differential equation consisting of terms from Table 5.1 on the discrete grid $u^n_{i,j,k}$.

The discretization of these equations raises two important issues. First is the availability of accurate, stable numerical schemes for solving these equations. Second is the problem of computational complexity and the fact that we have converted a *surface* problem to a *volume* problem, increasing the dimensionality of the domain over which the evolution equations must be solved.

The level-set terms in Table 1 are combined, based on the needs of the application, to create a partial differential equation on $\phi(\boldsymbol{x}, t)$. The solutions to these equations are computed using finite differences. Along the time axis solutions are obtained using finite *forward* differences, beginning with an initial model (i.e., volume) and stepping sequentially through a series of discrete times steps (which are denoted as superscripts on u). Thus the update equation is:

$$u^{n+1}_{i,j,k} = u^n_{i,j,k} + \Delta t \Delta u^n_{i,j,k}, \tag{18}$$

The term $\Delta u^n_{i,j,k}$ is a discrete approximation to $\partial\phi/\partial t$, which consists of a weighted sum

of terms such as those in Table 5.1. Those terms must, in turn, be approximated using finite differences on the volume grid.

6.1 Up-wind Schemes

The terms in Table 1 fall into two basic categories: the first-order terms (items 1 and 2 in Table 1) and the second-order terms (items 3 through 5). The first-order terms describe a moving wave front with a space-varying velocity (expression 1) or speed (expression 2). Equations of this form cannot be solved with a simple finite forward difference scheme. Such schemes tend to overshoot, and they are unstable. To address this issue Osher and Sethian [26] have proposed an *up-wind* scheme. The up-wind method relies on a one-sided derivative that looks in the up-wind direction of the moving wave front, and thereby avoids the over-shooting associated with finite forward differences.

We denote the type of discrete difference using superscripts on a difference operator, i.e., $\delta^{(+)}$ for forward differences, $\delta^{(-)}$ for backward differences, and δ for central differences. For instance, differences in the x direction on a discrete grid, $u_{i,j,k}$, with domain X and uniform spacing h are defined as

$$\delta_x^{(+)} u_{i,j,k} \triangleq (u_{i+1,j,k} - u_{i,j,k})/h, \tag{19}$$

$$\delta_x^{(-)} u_{i,j,k} \triangleq (u_{i,j,k} - u_{i-1,j,k})/h, \quad \text{and} \tag{20}$$

$$\delta_x u_{i,j,k} \triangleq (u_{i+1,j,k} - u_{i-1,j,k})/(2h), \tag{21}$$

$$\tag{22}$$

where we have left off the time superscript for conciseness. Second-order terms are computed using the *tightest-fitting* central difference operators. For example,

$$\delta_{xx} u_{i,j,k} \triangleq \left(u_{i+1,j,k} + u_{i-1,j,k} - 2u_{i,j,k}\right)/h^2, \tag{23}$$

$$\delta_{zz} u_{i,j,k} \triangleq \left(u_{i,j,k+1} + u_{i,j,k-1} - 2u_{i,j,k}\right)/h^2, \quad \text{and} \tag{24}$$

$$\delta_{xy} u_{i,j,k} \triangleq \delta_x \delta_y u_{i,j,k} \tag{25}$$

The discrete approximation to the first-order terms of in Table 5.1 are computed using the up-wind proposed by Osher and Sethian [10]. This strategy avoids overshooting by approximating the gradient of ϕ using a one-sided differences in the direction that is up-wind of the moving level-set thereby ensuring that no *new* contours are created in the process of updating $u_{i,j,k}^n$ (as depicted in Figure 3). The scheme is separable along each axis (i.e., x, y, and z).

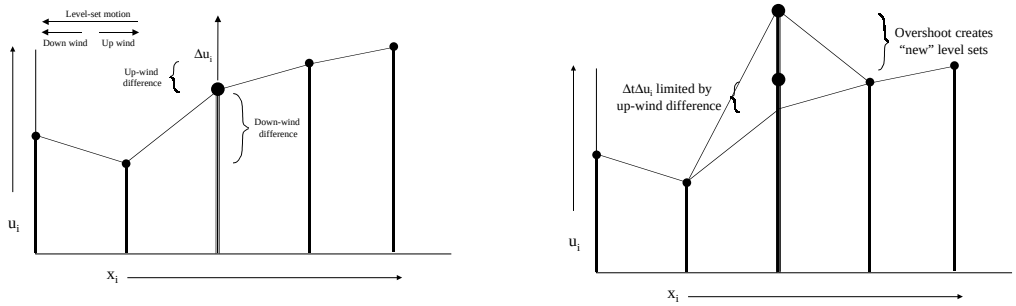


Figure 3: The up-wind numerical scheme uses one-sided derivatives to prevent overshooting and the creation of new level sets.

Consider Term 1 in Table 5.1. If we use superscripts to denote the vector components, i.e.,

$$\boldsymbol{F}(x,y,z) = (F^{(x)}(x,y,z), F^{(y)}(x,y,z), F^{(z)}(x,y,z)), \tag{26}$$

the up-wind calculation for a grid point $u^n_{i,j,k}$ is

$$\boldsymbol{F}(x_i,y_i,z_i)\cdot\nabla\phi(x_i,y_j,z_k,t) \approx \sum_{q\in\{x,y,z\}} F^{(q)}(x_i,y_i,z_i) \begin{cases} \delta_q^+ u^n_{i,j,k} & F^{(q)}(x_i,y_i,z_i) > 0 \\ \delta_q^i u^n_{i,j,k} & F^{(q)}(x_i,y_i,z_i) < 0 \end{cases} \tag{27}$$

The time steps are limited—the fastest moving wave front can move only one grid unit per iteration. That is

$$\Delta t_{\boldsymbol{F}} \leq \frac{1}{\sum_{q\in\{x,y,z\}} \sup_{i,j,k\in X}\{|\nabla F^{(q)}(x_i,y_j,z_k)|\}}. \tag{28}$$

For Term 2 in Table 5.1 the direction of the moving surface depends on the normal, and therefore the same up-wind strategy is applied in a slightly different form.

$$\begin{aligned} &G(x_i,y_j,z_k)|\nabla\phi(x_i,y_j,z_k,t)| \approx \\ &\sum_{q\in\{x,y,z\}} G(x_i,y_i,z_i) \begin{cases} \max^2(\delta_q^+ u^n_{i,j,k},0) + \min^2(\delta_q^- u^n_{i,j,k},0) & G(x_i,y_i,z_i) > 0 \\ \min^2(\delta_q^+ u^n_{i,j,k},0) + \max^2(\delta_q^- u^n_{i,j,k},0) & G(q)(x_i,y_i,z_i) < 0 \end{cases} \end{aligned} \tag{29}$$

The time steps are, again, limited by the fastest moving wave front:

$$\Delta t_G \leq \frac{1}{3\sup_{i,j,k\in X}\{|\nabla G(x_i,y_j,z_k)|\}} \tag{30}$$

To compute approximation the update to the second-order terms in Table 5.1 requires only central differences . Thus, the mean curvature is approximated as:

$$\begin{aligned} H^n_{i,j,k} &= \frac{1}{2}\left(\left(\delta_x u^n_{i,j,k}\right)^2 + \left(\delta_y u^n_{i,j,k}\right)^2 + \left(\delta_z u^n_{i,j,k}\right)^2\right)^{-1} \Big[\left(\left(\delta_y u^n_{i,j,k}\right)^2 + \left(\delta_z u^n_{i,j,k}\right)^2\right)\delta_{xx}u^n_{i,j,k} \\ &+ \left(\left(\delta_z u^n_{i,j,k}\right)^2 + \left(\delta_x u^n_{i,j,k}\right)^2\right)\delta_{yy}u^n_{i,j,k} + \left(\left(\delta_x u^n_{i,j,k}\right)^2 + \left(\delta_y u^n_{i,j,k}\right)^2\right)\delta_{zz}u^n_{i,j,k} \\ &- 2\delta_x u^n_{i,j,k}\delta_y u^n_{i,j,k}\delta_{xy}u^n_{i,j,k} - 2\delta_y u^n_{i,j,k}\delta_z u^n_{i,j,k}\delta_{yz}u^n_{i,j,k} - 2\delta_z u^n_{i,j,k}\delta_x u^n_{i,j,k}\delta_{zx}u^n_{i,j,k}\Big] \end{aligned} \tag{31}$$

Such curvature terms can be computing by using a combination of forward and backward differences as described in [27]. In some cases this is advantageous—but the details are beyond the scope of this paper.

The time steps are limited, for stability, to

$$\Delta t_H \leq \frac{1}{6}. \tag{32}$$

When combining terms, the maximum time steps for each terms is scaled by one over the weighting coefficient for that term.

6.2 Narrow-Band Methods

If one is interested in only *a single level set*, the formulation described previously is not efficient. This is because solutions are usually computed over the entire domain of ϕ. The solutions, $\phi(x, y, z, t)$ describe the evolution of an embedded family of contours. While this dense family of solutions might be advantageous for certain applications, there are other applications that require only a single surface model. In such applications the calculation of solutions over a dense field is an unnecessary computational burden, and the presence of contour families can be a nuisance because further processing might be required to extract the level set that is of interest.

Fortunately, the evolution of a single level set, $\phi(\boldsymbol{x}, t) = k$, is not affected by the choice of embedding. The evolution of the level sets is such that they evolve independently (to within the error introduced by the discrete grid). Furthermore, the evolution of ϕ is important only in the vicinity of that level set. Thus, one should perform calculations for the evolution of ϕ only in a neighborhood of the surface $\mathcal{S} = \{\boldsymbol{x} | \phi(\boldsymbol{x}) = k\}$. In the discrete setting, there is a particular subset of grid points whose values control a particular level set (see Figure 4). Of course, as the surface moves, that subset of grid points must change to account for its new position.

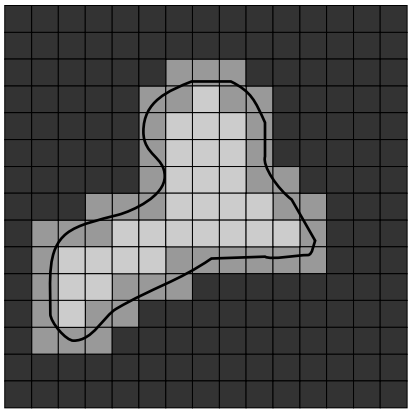


Figure 4: A level curve of a 2D scalar field passes through a finite set of cells. Only those grid points nearest to the level curve are relevant to the evolution of that curve.

Adalsteinson and Sethian [28] propose a *narrow-band* approach which follows this line of reasoning. The narrow-band technique constructs an embedding of the evolving curve or surface via a signed distance transform. The distance transform is truncated, i.e, computed over a finite width of only m points that lie within a specified distance to the level set. The remaining points are set to constant values to indicate that they do not lie within the narrow band, or *tube* as they call it. The evolution of the surface (they demonstrate it for curves in the plane) is computed by calculating the evolution of u only on the set of grid points that are within a fixed distance to the initial level set, i.e. within the narrow band. When the evolving level set approaches the edge of the band (see Figure 5), they calculate a new distance transform and a new embedding, and they repeat the process. This algorithm relies on the fact that the embedding is not a critical aspect of the evolution of the level set. That is, the embedding can be transformed or recomputed at any point in time, so long as such a transformation does not change the position of the kth level set, and the evolution will be unaffected by this change in the embedding.

Despite the improvements in computation time, the narrow-band approach is not optimal for several reasons. First it requires a band of significant width ($m = 12$ in the examples of [28]) where one would like to have a band that is only as wide as necessary to calculate the derivatives of u near the level set (e.g. $m = 2$). The wider band is necessary because the narrow-band algorithm trades off two competing computational costs. One is the cost of stopping the evolution and computing the position of the curve and distance transform (to sub-cell accuracy) and determining the domain of the band. The other is the cost of computing the evolution process over the entire band. The narrow-band method also requires additional techniques, such as smoothing, to maintain the stability at the boundaries of the band, where some grid points are undergoing the evolution and nearby neighbors are static.

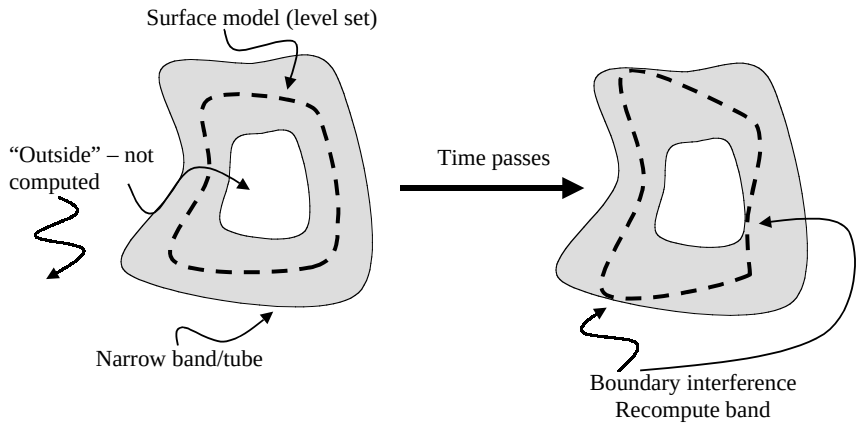


Figure 5: The narrow band scheme limits computation to the vicinity of the specific level set. As the level-set moves near the edge of the band the process is stopped and the band recomputed.

6.3 The Sparse-Field Method

The basic premise of the narrow band algorithm is that computing the distance transform is so costly that it cannot be done at every iteration of the evolution process. The strategy proposed here is to use an approximation to the distance transform that makes it feasible to recompute the neighborhood of the level-set model at each time step. Computation of the evolution equation is computed on a band of grid points that is only on point wide. The embedding is extended from the active points to a neighborhood around those points that is precisely the width needed at each time. This extension is done via a fast distance transform approximation.

This approach has several advantages. First, the algorithm does precisely the number of calculations needed to compute the next position of the level curve. It does not require explicitly recalculating the positions of level sets and their distance transforms. Because the number of points being computed is so small, it is feasible to use a linked-list to keep track of them. Thus, at each iteration the algorithm visits only those points adjacent to the k-level curve. For large 3D data sets, the very process of incrementing a counter and checking the status of all of the grid points is prohibitive.

The *sparse-field* algorithm is analogous to a locomotive engine that lays down tracks before it and picks them up from behind. In this way the number of computations increases with the surface area of the model rather than the resolution of the embedding. Also, the sparse-field approach identifies a single level set with a specific set of points whose values control the position of that level set. This allows one to compute external forces to an accuracy that is better than the grid spacing of the model, resulting in a modeling system that is more accurate for various kinds of "model fitting" applications.

The sparse-field algorithm takes advantage of the fact that a k-level surface, S, of a discrete image u (of any dimension) has a set of cells through which it passes, as shown in Figure 4. The set of grid points adjacent to the level set is called the *active set*, and the individual elements of this set are called *active points.* As a first-order approximation, the distance of the level set from the center of any active point is proportional to the value of u divided the gradient magnitude at that point. Because all of the derivatives (up to second order) in this approach are computed using nearest neighbor differences, only the active points and their neighbors are relevant to the evolution of the level-set at any particular time in the evolution process. The strategy is to compute the evolution given by equation 17 on the active set and then update neighborhood around the active set using a fast distance transform. Because active points must be adjacent to the level-set model, their positions lie within a fixed distance to the model. Therefore the values of u for locations in the active set must lie within a certain range. When active-point values move out of this *active range*

they are no longer adjacent to the model. They must be removed from the set and other grid points, those whose values are moving into the active range, must be added to take their place. The precise ordering and execution of these operations is important to the operation of the algorithm.

The values of the points in the active set can be updated using the up-wind scheme for first-order terms and central differences for the mean-curvature flow, as described in the previous sections. In order to maintain stability, one must update the neighborhoods of active grid points in a way that allows grid points to enter and leave the active set without those changes in status affecting their values. Grid points should be removed from the active set when they are no longer the nearest grid point to the zero crossing. If we assume that the embedding u is a discrete approximation to the distance transform of the model, then the distance of a particular grid point, $x_m = (i, j, k)$, to the level set is given by the value of u at that grid point. If the distance between grid points is defined to be unity, then we should remove a point from the active set when the value of u at that point no longer lies in the interval $[-\frac{1}{2}, \frac{1}{2}]$ (see Figure 6). If the neighbors of that point maintain their distance of 1, then those neighbors will move into the active range just x_m is ready to be removed.

There are two operations that are significant to the evolution of the active set. First, the values of u at active points change from one iteration to the next. Second, as the values of active points pass out of the active range they are removed from the active set and other, neighboring grid points are added to the active set to take their place. In [21] the author gives some formal definitions of active sets and the operations that affect them, which show that active sets will always form a boundary between positive and negative regions in the image, even as control of the level set passes from one set off active points to another.

Because grid points that are near the active set are kept at a fixed value difference from the active points, active points serve to control the behavior of non-active grid points to which they are adjacent. The neighborhoods of the active set are defined in *layers*, $L_{+1}, \ldots L_{+N}$ and $L_{-1}, \ldots L_{-N}$, where the i indicates the distance (city block distance) from the nearest active grid point, and negative numbers are used for the outside layers. For notational convenience the active set is denoted L_0.

The number of layers should coincide with the size of the footprint or neighborhood used to calculate derivatives. In this way, the inside and outside grid points undergo no changes in their values that affect or distort the evolution of the zero set. Most of the level-set work relies on surface normals and curvature, which require only second-order derivatives of ϕ. Second-order derivatives are calculated using a $3 \times 3 \times 3$ kernel (city-block distance 2 to the corners). Therefore only five layers are necessary (2 inside layers, 2 outside layers, and the active set). These layers are denoted L_1, L_2, L_{-1}, L_{-2}, and L_0.

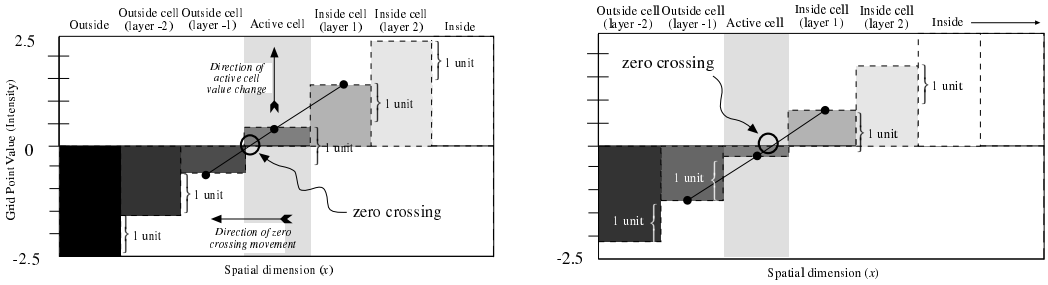


Figure 6: The status of grid points and their values at two different points in time show that as the zero crossing moves, *activity* is passed one grid point to another.

The active set has grid point values in the range $[-\frac{1}{2}, \frac{1}{2}]$. The values of the grid points in each neighborhood layer are kept 1 unit from the next layer closest to the active set (as in Figure 6). Thus the values of layer L_i fall in the interval $[i - \frac{1}{2}, i + \frac{1}{2}]$. For $2N + 1$ layers, the values of the grid points that are totally inside and outside are $N + \frac{1}{2}$ and $-N - \frac{1}{2}$, respectively. The procedure for updating the image and the active set based on surface movements is as follows:

1. For each active grid point, $x_m = (i, j, k)$, do the following:
   (a) Calculate the local geometry of the level set.
   (b) Compute the net change of u_{x_m}, based on the internal and external forces, using some stable (e.g., up-wind) numerical scheme where necessary.
2. For each active grid point x_j add the change to the grid point value and decide if the new value $u_{x_m}^{n+1}$ falls outside the $[-\frac{1}{2}, \frac{1}{2}]$ interval. If so, put x_m on lists of grid points that are changing status, called the *status list*; S_1 or S_{-1}, for $u_{x_m}^{n+1} > 1$ or $u_{x_m}^{n+1} < -1$, respectively.
3. Visit the grid points in the layers L_i in the order $i = \pm 1, \ldots \pm N$, and update the grid point values based on the values (by adding or subtracting one unit) of the next inner layer, $L_{i \mp 1}$. If more than one $L_{i \mp 1}$ neighbor exists then use the neighbor that indicates a level curve closest to that grid point, i.e., use the maximum for the outside layers and minimum for the inside layers. If a grid point in layer L_i has no $L_{i \mp 1}$ neighbors, then it gets demoted to $L_{i \pm 1}$, the next level away from the active set.
4. For each status list $S_{\pm 1}, S_{\pm 2}, \ldots, S_{\pm N}$ do the following:

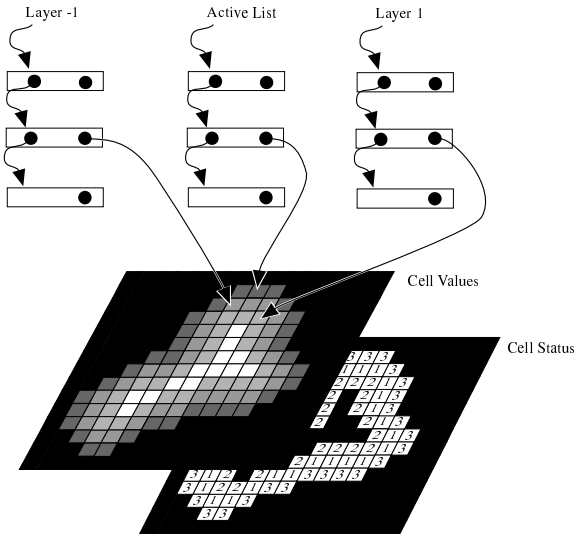


Figure 7: Linked-list data structures provide efficient access to those grid points with values and status that must be updated.

(a) For each element x_j on the status list S_i, remove x_j from the list $L_{i\mp1}$, and add it to the L_i list, or, in the case of $i = \pm(N+1)$, remove it from all lists.

(b) Add all $L_{i\mp1}$ neighbors to the $S_{i\pm1}$ list.

This algorithm can be implemented efficiently using linked-list data structures combined with arrays to store the values of the grid points and their states as shown in Figure 7. This requires only those grid points whose values are changing, the active points and their neighbors, to be visited at each time step. The computation time grows as m^{n-1}, where m is the number of grid points along one dimension of u (sometimes called the resolution of the discrete sampling). Computation time for dense-field approach increases as m^n. The m^{n-1} growth in computation time for the sparse-field models is consistent with conventional (parameterized) models, for which computation times increase with the resolution of the domain, rather than the range.

Another important aspect of the performance of the sparse-field algorithm is the larger time steps that are possible. The time steps are limited by the speed of the "fastest" moving level curve, i.e., the maximum of the force function. Because the sparse-field method calculates the movement of level sets over a subset of the image, time steps are bounded from below by those of the dense-field case, i.e.,

$$\sup_{x \in \mathcal{A} \subset X} (g(x)) \leq \sup_{x \in X} (g(x)), \tag{33}$$

where $g(x)$ is the space varying speed function and $\mathcal{A}$ is the active set.

Results from previous work [21] have demonstrated several important aspects of the sparse-field algorithm. First, the manipulations of the active set and surrounding layers allow the active set to "track" the deformable surface as it moves. The active set always divides the inside and outside of the objects it describes (i.e., it stays closed). Empirical results show significant increases in performance relative to both the computation of full domain and the narrow-band method, as proposed in the literature. Empirical results also show that the sparse-field method is about as accurate as both the full, discrete solution, and the narrow-band method. Finally, because the method positions level sets to sub-voxel accuracy it avoids aliasing problems and is more accurate then these other methods when it comes to *fitting* level-set models to other surfaces. This sub-voxel accuracy is important aspect of the implementation, and will significantly impact the quality of the results for the applications that follow.

7 Applications

This section describes several examples of how level-set surface models can be used to address problems in graphics, visualization, and computer vision. These examples are a small selection of those available in the literature. All of these examples where implemented using the sparse-field algorithm and the VISPack library, which is described in the section that follows.

7.1 Surface Morphing

This section summarizes the work of [29], which describes the use of level-set surface models to perform 3D shape metamorphosis. The *morphing* of 3D surfaces is the process of constructing a series of 3D models that constitute a smooth transition from one shape to another (i.e., a homotopy). Such a capability is interesting for creating animations and as a tool for geometric modeling. There is not yet a single, general method for generating such transitional shapes. However, there are several desirable aspects of morphing algorithms that allow us to compare the adequacy of different approaches to surface morphing. Several desirable properties of 3D surface morphing are:

1. The transition process should begin with an *initial* surface and end with a specified *target* surface.

2. The morphing algorithm should apply to a wide range of shapes and topologies.

3. Intermediate surfaces should undergo continuous 3D transitions (rather than continuity only in the image space).

4. A 3D morphing algorithm should incorporate user input easily but should degrade gracefully without it.

5. Transitional shapes should depend only on the surface geometry of the two input shapes and user input.

These requirements are not exhaustive, but they capture many of the practical aspects of 3D morphing.

In this section we show how level-set models provide an algorithm for 3D morphing which meets most of these criteria and compare favorably with existing algorithms. Furthermore, this algorithm is a natural extension of the mathematical principles discussed in previous sections. The strategy is to allow a free-form deformation of one surface (called the *initial* surface) using the signed distance transform of a second surface (the *target* surface). This free-form deformation is combined with an underlying coordinate transformation that gives either a rough global alignment of the two surfaces, or one-to-one relationships between a finite set of landmarks on both the initial and target surfaces. The coordinate transformation can be computed automatically or using user input (as in [30]).

Much of the previous 3D morphing work has focused on morphing parametric models [31, 32] and applies to only very limited classes of shapes and topologies. Several authors have described volumetric techniques. Hughes [33] demonstrates how volumes can provide topological flexibility in surface morphing. Lerios et al. [30] followed up with a volume-based scheme which incorporates user input via underlying coordinate transformations (a known generalization the image warping technique that is often used in image morphing). Neither of these approaches have dealt with the deeper issue of deforming the level sets of a volume, but rather rely on the properties of the embedding. Payne and Toga [34] as well as Cohen-Or *et al.* [35] fix the embedding problem by using a signed distance transform to create volumes from surfaces. However, interpolating distance transforms can introduce artifacts that violate the previously stated properties, and both of these methods use a discrete distance transform which introduces volume aliasing.

7.1.1 Free-Form Deformations

The distance transform gives the nearest Euclidean distance to a set of points, curve, or surface. For closed surfaces in 3D, the signed distance transform gives a positive distance for points inside and negative for points outside (one can also choose the opposite sign convention).

If two connected shapes overlap then the initial surface can expand or contract using the distance transform of the target. The steady state of such a deformation process is a shape consisting of the zero set of the distance transform of the target. That is, the initial object becomes the target. This is the basis of the proposed 3D morphing algorithm.

Let $D(\boldsymbol{x})$ be the signed distance transform of the target surface, B, and let A be the initial surface. The evolution process which takes a model S from A to B is defined by

$$\frac{\partial \boldsymbol{x}}{\partial t} = \boldsymbol{N}\ D(\boldsymbol{x}), \tag{34}$$

where $\boldsymbol{x}(t) \in \mathcal{S}_t$ and $\mathcal{S}_{t=0} = A$. The free-form deformations can be combined with an underlying coordinate transformation. The strategy is to use a coordinate transformation (for instance a translation and rotation) to position the two surfaces near each other. These transformations can capture gross similarities in shape as well as user input. A coordinate transformation is given by

$$\boldsymbol{x}' = T(\boldsymbol{x}, \alpha), \tag{35}$$

where $0 \leq \alpha \leq 1$ parameterizes a continuous family of these transformations that begins with identity, i.e. $\boldsymbol{x} = T(\boldsymbol{x}, 0)$. The evolution equation for a parametric surface is

$$\frac{\partial \boldsymbol{x}}{\partial t} = \boldsymbol{N}\ D(T(\boldsymbol{x}, 1)), \tag{36}$$

and the corresponding level-set equation is

$$\frac{\partial \Phi(\boldsymbol{x}, t)}{\partial t} = |\nabla \Phi(\boldsymbol{x}, t)|\ D(T(\boldsymbol{x}, 1)). \tag{37}$$

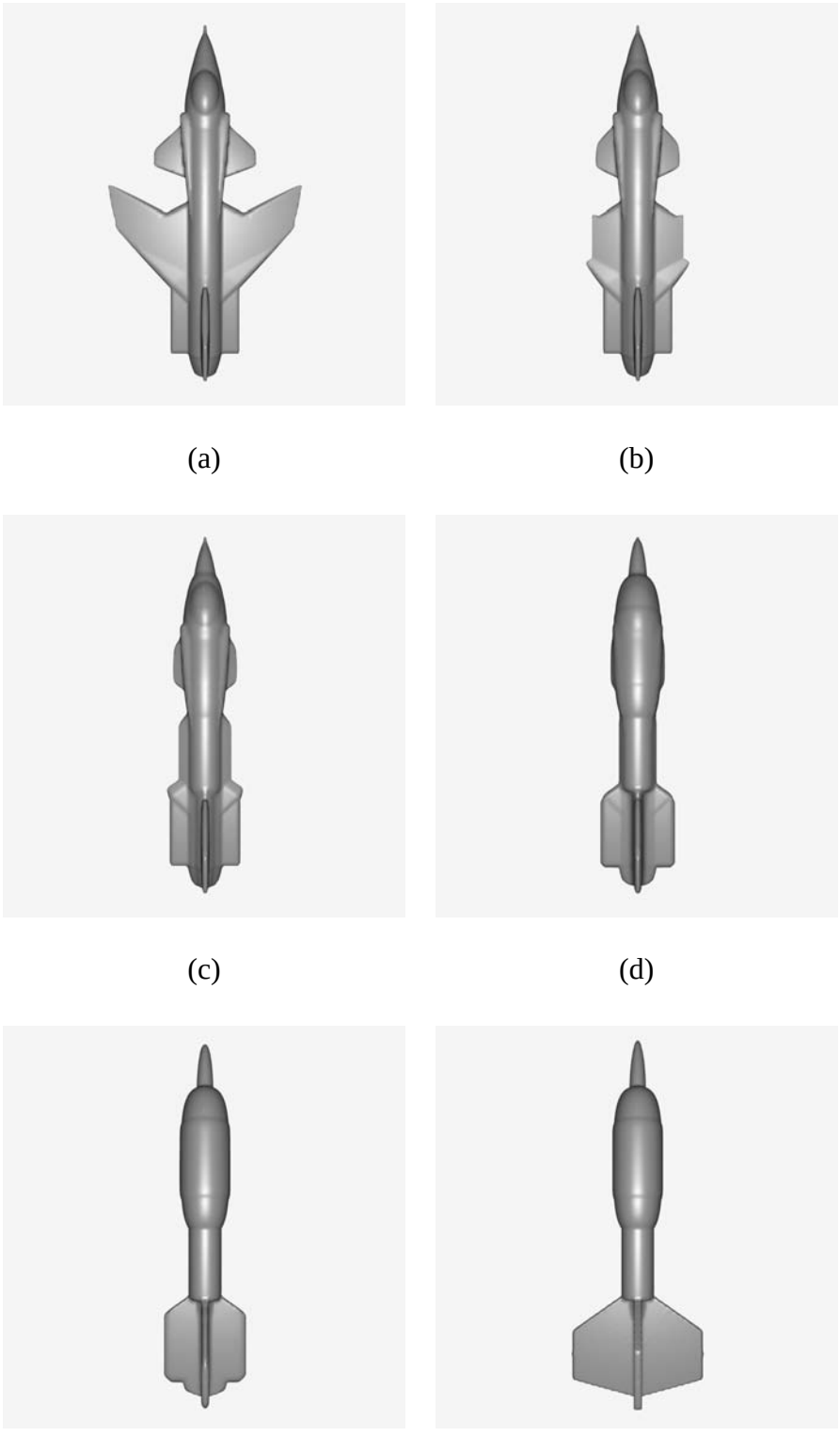
This process produces a series of transition shapes (parameterized by t). The coordinate transformation can be a global rotation, translation, or scaling, or it might be a *warping of the underlying 3D space* as was used by [30]. Incorporating user input is important for any surface morphing technique, because in many cases finding the best set of transition surfaces depends on context. Only users can apply semantic considerations to the transformation of one object to another. However, this underlying coordinate transformation

Figure 8: A 3D model of a jet that was built using Clockworks, a CSG modeling system.

can, in general, achieve only some finite similarity between the "warped" initial model and the target, and even this may require a great deal of user input. In the event that a user is not able or willing to define every important correspondence between two objects, some other method must "fill in" the gaps remaining between the initial and target surface. In [30] they propose alpha blending to achieve that smooth transition—really just a fading from one surface to the other. We are proposing the use of the free-form deformations, implemented with level-set models, to achieve a continuous transition between the shapes that result from the underlying coordinate transformation. We have also experimented with ways of automatically orienting and scaling objects, using 3D moments, in order to achieve a significant correspondence between two objects.

Figure 8 shows a 3D model of a jet that was built using Clockworks [36], a CSG modeling system. Lerios et al. [30] demonstrate the transition of a jet to a dart, which was accomplished using 37 user-defined correspondences, roughly a hundred user-defined parameters. Figure 9 shows the use of level-set models to construct a set of transition surfaces between a jet and a dart. The triangle mesh is extracted from the volume using the method of marching cubes [5]. These results are obtained without any user input. Distance transforms on the CSG models are computed near the level surface using an analytical description and extended into the volume using a level-set method [37].

The application in this section shows how level-set models moving according to the first-order term given in expression 2 in Table 1 can "fit" other objects by moving with a speed that depends on the signed distance transform of the target object. The application in the



(a) (b)

(c) (d)

(e) (f)

Figure 9: The deformation of the jet to a dart using a level-set model moving with a speed defined by the signed distance transform of the target object.

next section relies on expression 5 of Table 1, a second-order flow that depends on the principal curvatures of the surface itself.

7.2 Filleting and Blending Solid Objects

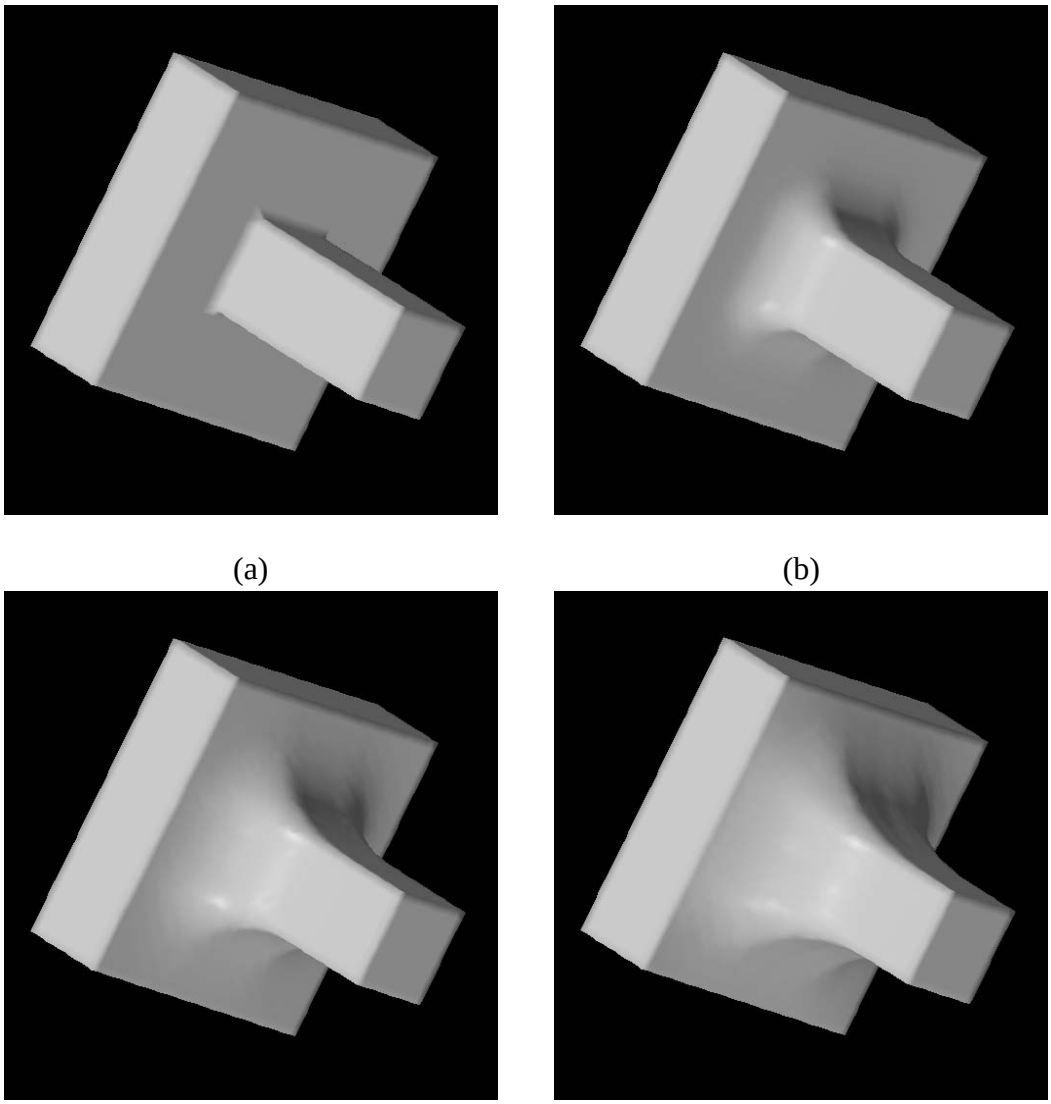
The construction of blending surfaces is an important tool in solid modeling. Geometric solid primitives and their intersections often produce sharp corners or creases that are often not consistent with the real-world objects that they are intended to represent. This section shows how blending can be described as a deformation process, where surfaces move under a geometric flow that can add or remove material based on local curvature information. The result is a method for solid object blending that does not depend on any particular model representation. Thus this method is not restricted to a specific class of shapes or topologies. Additionally, the results are invariant; they do not depend on arbitrary choices of coordinate systems or bases. The only requirement is that the blended objects must be closed surfaces with some known inside-outside function.

Surface blending techniques are typically tied very closely to the choice of geometric primitives. For instance, Middleditch and Sears [38] propose a set-theoretic method for blending solids which relies on low-order algebraic primitives. A fillet at the joint of two tori requires the solution of a degree 32 polynomial. Bloomenthal and Shoemake [39] propose a modeling system based on convolutions, which relies on a skeletonized representation of objects. In general the use of convolution to achieve deformations on implicit shapes results in shapes that reflect both the shape of the model and the embedding, Φ.

The blending method proposed in this section implements an interative smoothing scheme that smooths only along the level set; the final result is independent of the embedding. Consider the case of fillets. We propose that a fillet can be constructed from a process of "filling in" material in places of high curvature. The curvature of a level-set model can be calculated from the embedding, and the deformation of the level set is well defined by the curvature terms in Table 1.

The strategy is to construct a curvature term, k_p, that consists of only positive curvatures. [1] The principal curvatures of the level sets of Φ are functions of Φ and its derivatives. For a specific Φ the principal curvatures are functions of 3-space $k_1(\boldsymbol{x})$ and $k_2(\boldsymbol{x})$. For *adding* material the joint between two objects, we consider only the positive curvature components,

[1] The sign of curvature is defined by the direction of the normals— in this work normals point into the volume enclosed by the object.



(a) (b)

(c) (d)

Figure 10: Two rectangular solid models are joined by a volumetric fillet that is created from a positive curvature flow.

i.e.,

$$\frac{\partial \Phi}{\partial t} = |\nabla \Phi| k_p = |\nabla \Phi| k_1^+ + |\nabla \Phi| k_2^+, \tag{38}$$

where k^+ consists of only the positive parts of k and is defined as zero elsewhere. Because the use of separate curvature terms can cause over-shooting, the up-wind scheme (treating k_p as a space-varying velocity in the normal direction) is used for this evolution.

Figure 10 shows how the positive-curvature flow can be used to construct fillets. No knowledge of the underlying models is necessary. The fillets grow larger as more time passes. The physical extent or position of the fillet can be controlled by either specifying a region of action or by placing a small blob of deformable material in the joint that requires a fillet. Figure 11 shows how such a blending capability can be useful in animation. In this case a pair of superquadrics undergo a rigid transformation that controls their relative positions. Level-set models with a positive-curvature flow are used to create a smooth joint between these two primitives. Notice that the positive curvature method does not suffer from the growth or expansion artifacts that are often associated with distance-based blending methods [40].

Thus, a second-order flow can create smooth blends between objects in a way that does not require specific knowledge of the shapes or topologies of the object involved. The application in the next section, 3D scene reconstruction, shows how a combination of first-order and second-order terms from Table 1 are combined to create technique that fits models to data while maintaining certain smoothness constraints and thereby offsetting the effects of noise.

7.3 3D Reconstruction from Multiple Range Maps

Level-set models are useful for problems related to 3D reconstruction. Previous work has presented level-set results derived from noisy 3D data such as MRI [19] and ultrasound [41]. In [42] we have shown how the reconstruction of objects from multiple range maps can be formulated as a problem of finding the surface that optimizes the posterior probability given a set of measurements (noisy range maps) and some information about the a-priori probability of different kinds of surfaces. That optimization problem can be expressed as a volume integral which can be solved with level-set models. This section presents the mathematical expressions that result from those formulations and presents some new results: the reconstruction of entire scenes by fitting level-set models to the data from a scanning LADAR (laser ranging and detection) system.

A *range map* is a collection of range measurements taken along different directions (lines

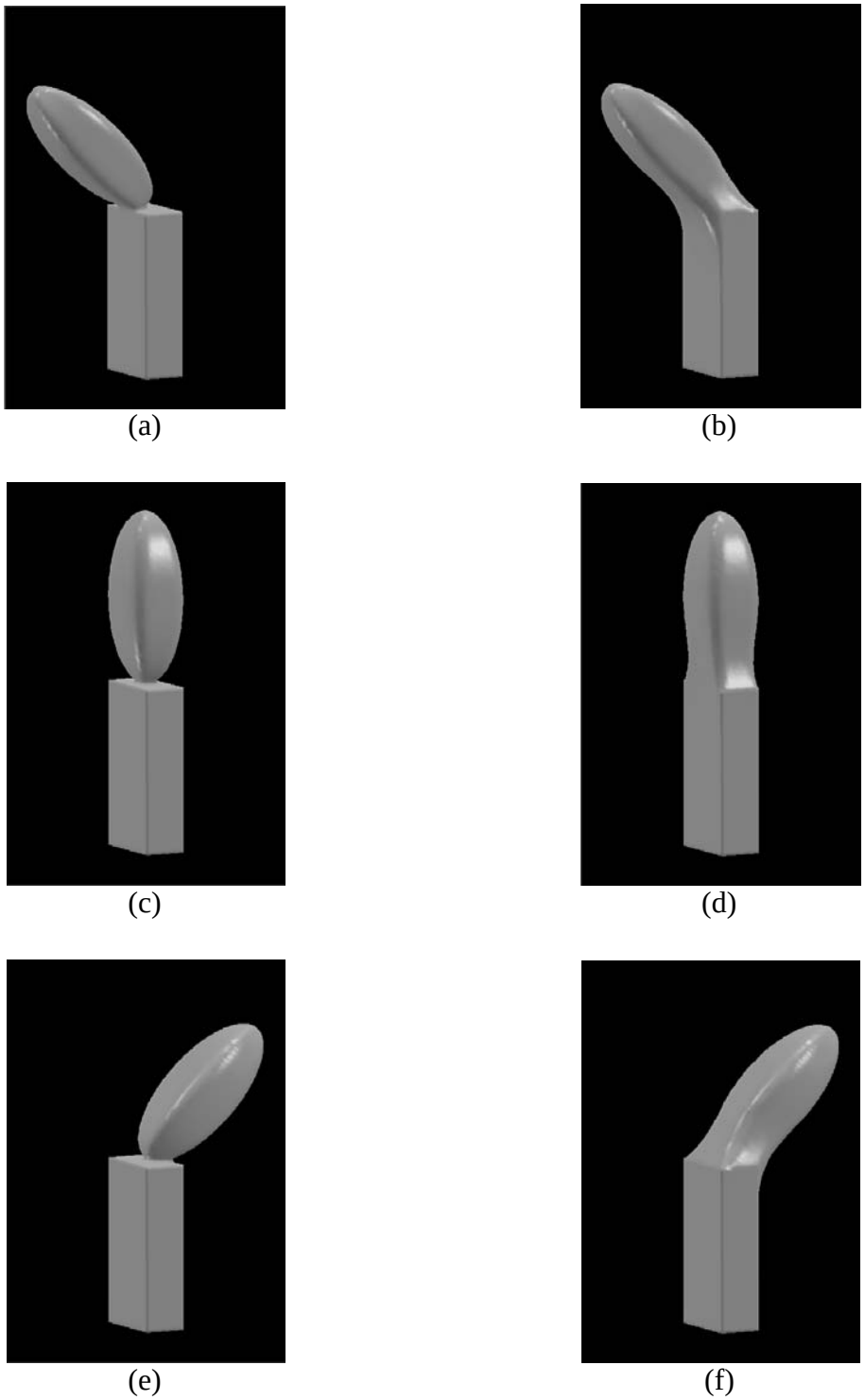


Figure 11: A short animation is created by specifying the relative motion between two superquadric components of an object. A positive-curvature flow (applied frame by frame to the joint between the two 3D models) creates a smooth, flexible object.

of sight) but from a single point of view. Range maps could come from any number of different sources including laser scanners, structured light depth systems, shape from stereo, or shape from motion. We assume that such range maps are noisy and uncertain. The goal is to combine a number of range maps from different points of view to create a 3D structure that reflects the collective confidence and depth measures.

Several examples in the literature have applied parametric models to this task. Turk and Levoy [43], for instance, "zip" together triangle meshes in order to construct 3D objects from sequences of range maps from a laser range finder. They perform minor adjustments to the surface position in order account for ambiguity in the range maps. Their approach assumes very little noise in the input, which is reasonable given the high quality of their range maps. Chen and Medioni [44] use a parametric (triangle mesh) model which expands inside a sequence of range maps. Curless and Levoy [45] describe a volume-based technique for combining range data. They use the signed distance transform to encode volume elements with data that represent the averages (with some allowance for outliers) of multiple measurements. Surfaces of objects are the level sets of volumes. Related approaches are given in [46, 47]. Bajaj et. al. [48] use a Delaunay triangulation to impose a topology on a set of unordered 3D points and then fit trivariate Bernstein-Bezier patches—i.e. a higher-order implicit model—to the data. Muraki [2] uses implicit or blobby models to reconstruct objects from range data. The individual blobs are spherically symmetric 3D potentials that are combined linearly so that they blend together. The resulting models, with approximately 400 primitives are quite coarse.

This work differs from previous work in two ways. First, rather than heuristics, our reconstruction strategy is based on a strategy that solves for the optimal surface estimate. This optimal estimate includes information about one's expectations of the likelihood of different surfaces. The result is not a closed-form solution, but an iterative process that seeks to fit a level-set model to the data while enforcing a kind of smoothness on the data.

7.3.1 Objective function for multiple range maps

The evolution equation for the estimation of optimal surfaces is shown in [42] to consist of two parts:

$$\frac{\partial \boldsymbol{x}}{\partial t} = -G(\boldsymbol{x})\boldsymbol{N} + \rho(\mathcal{S}). \tag{39}$$

This first part, $-G(\boldsymbol{x})\boldsymbol{N}$, is the data term, which is a movement with variable speed (as in expression 2 from Table 1) that is the cumulative effect from all of the individual range maps. The second part is the prior, which describes the likelihood of the surface indepen-

dent of the data. The data term is

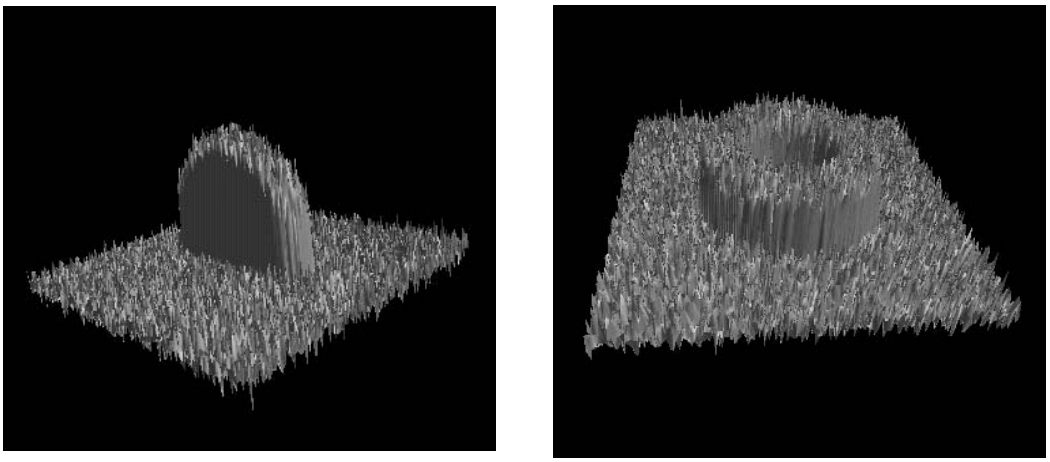
$$G(\boldsymbol{x}) = \sum_j c^{(j)}(\boldsymbol{x})\ D^{(j)}(\boldsymbol{x})\ \omega\left(D^{(i)}(\boldsymbol{x})\right)\ \gamma^{(j)}(\boldsymbol{x}), \tag{40}$$

where D_j is the signed distance along the line of sight from a range measurement in range map j associated passing through $\boldsymbol{x}$. The function $\omega : \mathbb{R} \mapsto \mathbb{R}$ is a windowing function that limits the penalty of any one range measurement, and $c(\cdot)$ is a confidence function, which is inversely proportional to the level of noise in the range measurement associated with the same line of sight. The term $\gamma(\cdot)$ is an integration constant that takes into account the curvilinear coordinate system of the range scanner.

Thus, a set of range maps creates a scalar function of 3D, which describes the movement of a surface model as it seeks the optimal surface position. In the absence of a prior, $\rho = 0$, the zero set of this function is the final position (steady state) of that evolving surface. Thus, in the absence of a prior, one could sample $g(\boldsymbol{x})$ and obtain an approximation to the optimal surface estimate. This strategy results in an algorithm that is very much like that of [45].

There are several reasons for going to an iterative scheme for finding optimal solutions. First is the use of a prior. In surface reconstruction, even a very low level of noise can degrade the quality of the rendered surfaces in the final result, and in such cases better reconstructions can be obtained by introducing a prior. Second is aliasing. Discretizing $g(\boldsymbol{x})$ and finding the zero crossings will cause aliasing in those places where the transition from positive to negative is particularly steep. A deformable model can place the surface much more precisely. The third reason for going to an iterative scheme is that despite the windowing function $\omega(\boldsymbol{x})$ there is interference between different range maps at places of high curvature. This problem is addressed by introducing a nonlinearity which is solved in an iterative scheme given by equation 39. In the work described in [21], the solution of the linear problem, the zero set of $g(\boldsymbol{x})$, serves as the initial estimate for the nonlinear, iterative optimization strategy that results from the inclusion of a prior and a nonlinear term that compensates for lack of any explicit model of self occlusions.

Equation 39 includes a *prior*, which is a likelihood function on surface shape. A reasonable choice of prior is one that models objects with less surface area as more likely than objects with more surface area. Alternatively, one could say that given a set of surfaces that are near the data, the algorithm should choose a surface that has less area. Often, but not always, this will be the smoother surface. The $\rho(\mathcal{S})$ that results from this prior is the mean curvature. Therefore the evolution of the surface, using the level-set formulation, that seeks to maximize the posterior probability (given a set of range maps and a prior that penalizes



(a) (b)

Figure 12: Range maps: Synthetic range data 200×200 pixels with 20% Gaussian white noise of a torus end (a) and side (b).

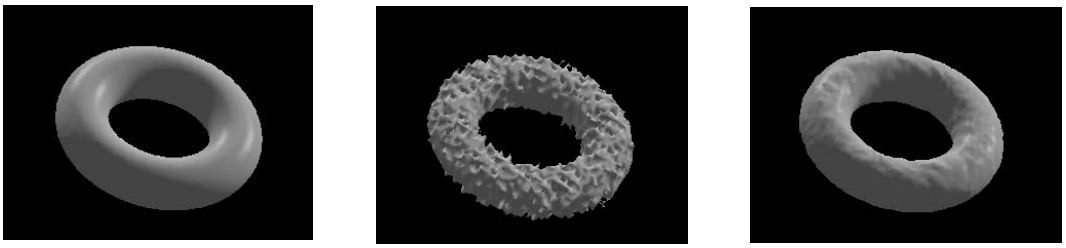
surface area) is

$$\frac{\partial \Phi(\boldsymbol{x},t)}{\partial t} = |\nabla \Phi(\boldsymbol{x})| \sum_j \left(D^{(j)}(\boldsymbol{x})\, \omega\left(D^{(i)}(\boldsymbol{x})\right) \times\ \gamma^{(j)}(\boldsymbol{x})\ c^{(j)}(\boldsymbol{x}) \frac{\left(\nabla\Phi \cdot \boldsymbol{n}^{(j)}(\boldsymbol{x})\right)^{+}}{\nabla\Phi \cdot \boldsymbol{n}^{(j)}(\boldsymbol{x})} \right) + \beta H, \tag{41}$$

where $\boldsymbol{n}^{(j)}(\boldsymbol{x})$ is the line of sight from a range finder to a 3D point, $\boldsymbol{x}$, β is a free parameter that controls the level of smoothing in the model, and H is the expression for the mean curvature given in equation 8.

Figure 12 shows a pair of simulated range maps constructed from an analytical description of a torus. These 200×200 pixel range maps are corrupted with additive Gaussian noise that has a standard deviation of 20% (as a function of the smaller of the two radii). Six synthetic noise-corrupted viewpoints of a torus are combined to create a level-set reconstruction of a torus. Figure 13(a) shows the initial model (80×80×40 voxels) used for fitting a level-set models to the range data. Figure 13(b) shows the result of the level-set models that uses 13(a) as an initial state and has a value of β equal to 0.5. The result is a reasonable reconstruction of the noiseless model (Figure 13(c)) which combines the six points of view and the smoothing function.

Figure 14(a) shows a range map taken with the Perceptron model P5000, an infra-red, time-of-flight laser range finder with a pan-tilt mechanism. Figure 14(b) shows the amplitudes associated with the return signal (an *intensity*), and 14(c) shows a surface plot of the range



(a) (b) (c)

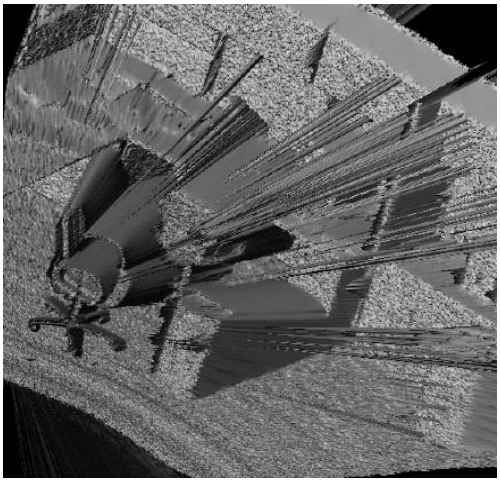
Figure 13: (a) An analytically-defined model of a torus. (b) An initial model ($80\times80\times40$ voxels) is constructed by combining six points of view of a torus and solving for $g(\boldsymbol{x}) = 0$. (c) The model, which is attracted to the range data but subject to internal forces, evolves and settles into a smoother steady state.

map to demonstrate the degree of noise (additive and outliers). Figure 14(d) shows the confidence values associated with those range measurements. These confidence values are derived from empirical data about the level of noise in the range finder (which depends on the return amplitude), and some analysis, from first principles, about the effects of uncertainty in the 3D positions of the scans and the model — which results in the lower confidence at edges as described in [42]. We combined twelve such views from different locations in the room to generate the results that follow.

Figure 15(a) shows the initial estimate based on the zero crossings of $g(\boldsymbol{x})$, and 15(b) shows the result of 32 iterations with the prior term and the correction for the surface normal direction. The size of the volume is $300 \times 150 \times 180$ voxels, and the resolution is 1.8 cm/voxel. These results show the ability of the statistically-based approach to overcome the noise in the scanner, and they show that the inclusion of iterative, model-fitting scheme helps create more accurate reconstructions. The resolution of the model falls below that of the scans, because it was limited by the random-access-memory available on our workstation. Some small features, such as the arm rests of the chairs, are lost because of the inaccuracies in the registration of the individual range maps.

(a)

(b)



(c)

(d)

Figure 14: (a) One of twelve range maps (b) The associated amplitude map (c) A surface plot of the range data to show the level of noise. (d) The confidence measures associated with those range values.

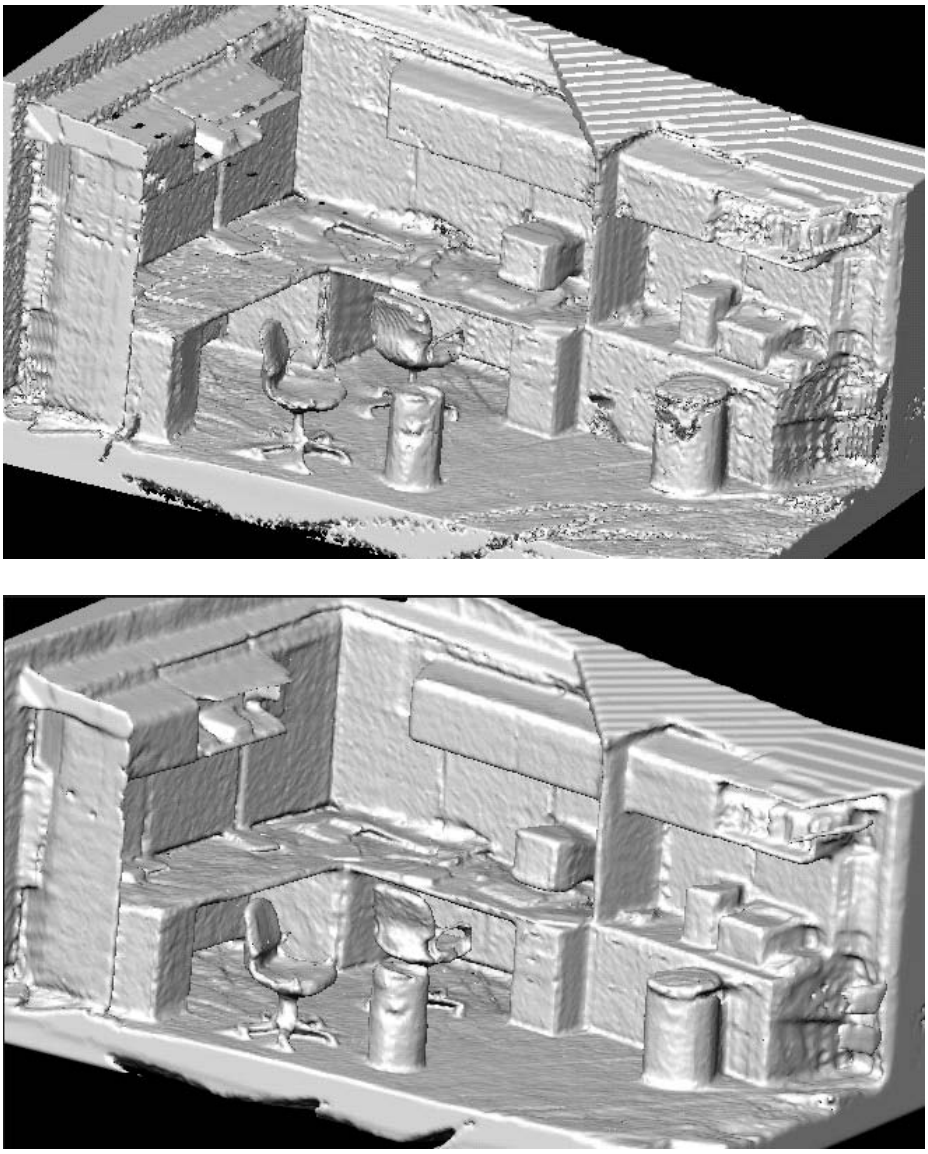


Figure 15: (top) The 3D reconstruction resulting from the zero crossings of $g(\boldsymbol{x})$ gives some averaging, but includes no prior. (bottom) The result of 32 iterations with the iterative scheme includes the prior and excludes influences of data on surfaces that face away from the scanner.

8 VISPACK

8.1 Introduction

VISPACK is a set of *C++*, object-oriented libraries for image processing, volume processing, and level-set surface modeling. It consists of five libraries: Matrix, Image, Volume, Util, and Voxmodel (level-set modeling). These libraries can be used separately or together when creating applications.

VISPACK incorporates eight basic design attributes. These are

Data Handles/Copy on Write: VISPack is an object-oriented library, and as such we allow the objects to handle memory management, and relieve the programmer (in most cases) from having to worry pointers and the corresponding memory allocation/deallocation problems. For this we use the data handles with a *copy on write protocol*. Copy constructors perform a shallow copy with reference counting until a *non const* operation on the underlying buffers forces a deep copy. Thus deep copies are performed only when necessary, but all memory is maintained by the objects and objects behave as "variables" rather than pointers.

Modified Data Hiding: Access to data in objects is generally through access methods, however, pointers to buffers for fast implementations are available.

Templates: VISPack utilizes the templating construct of C++ virtually throughout. Many of the objects, including images, volumes, lists, and arrays, are intended to support a wide range of data types. Thus, via templating programmers can define the pixels of different images of different types, such as floating point, 24-bit color, and 16-bit greyscale.

Use of Standard File Formats: When appropriate VISPack uses standard file formats. We choose formats that are well known and have publicly available libraries that can be distributed with our libraries. The matrix library uses a simple text format. The image library uses TIFF and FITS file formats. Because no standard format exists for saving volumes of data we do use a *raw* file format.

Operator Overloading: Proper use of operator overloading gives users a convenient way to execute operations on an object. When compined with the copy-on-write convention, operator overloading allows programmers to treat many heavy-weight objects (e.g. images and volumes) as variables. For instance, the following code computes non-maximal edges in a on a filtered volume.

```
Volume<float> dx, dy, dz;
Volume<float> vol_gauss = vol.gauss(0.5);
Volume<float> vol_out = (((dx = vol_gauss.dx()).power(2)
       *vol_gauss.dx(2)
     + ((dy = vol_gauss.dy()).power(2)*vol_gauss.dy(2)
     + ((dz = vol_gauss.dz()).power(2)*vol_gauss.dz(2)
     + dx*dy*(dx).dy() + dx*dz*(dx).dz())
     + dy*dz*(dy).dz()) ).zeroCrossings()
        && ((dx.power(2) + dy.power(2)) > T*T));
```

8.2 Level-Set Surface-Modeling Library

The Level-Set Surface-Modeling (LSSM) Library is an implementation of the level-set technique [10, 13] specifically for deforming surface models embedded in volumes. The implementation uses the sparse-field method described in [20]. The library implements all of the basic numerical algorithms and handles all of the data structures required to perform LSSM. The strategy for using this library is to subclass the object `VoxModel`, set some parameters, define a set of simple virtual functions that control the deformation process, initialize the model, and then direct the model to iteratively deform according to those equations. This section describes the relationship between the mathematics of previous sections and the VISPack library. Its also presents an example of using VISPack libarary to do 3D shape metamorphosis as described in Section 7.1.

8.2.1 Surface Deformation

The LSSM library allows one to solve for surface deformations, as a function of time, for general level-set surface movements of the form:

$$\frac{\partial \boldsymbol{x}}{\partial t} = \alpha \boldsymbol{F}(\boldsymbol{x}, \boldsymbol{N}(\boldsymbol{x})) + \beta G(\boldsymbol{x}, \boldsymbol{N}(\boldsymbol{x}))\boldsymbol{N}(\boldsymbol{x}) + \gamma \boldsymbol{N}(\boldsymbol{x}) + \eta E\left(k_1(\boldsymbol{x}), k_2(\boldsymbol{x})\right), \qquad (42)$$

where $\boldsymbol{x}$ is a point on the surface. This equation is solved by representing the surface as the kth level set of an implicit function $\phi(\boldsymbol{x}, t) : \mathbb{R}^3 \times \mathbb{R}^+ \mapsto \mathbb{R}$. This gives

$$\frac{\partial \phi}{\partial t} = \alpha \boldsymbol{F}(\boldsymbol{x}, \nabla\phi)) \cdot \nabla\phi + \beta G(\boldsymbol{x}, \nabla\phi)|\nabla\phi| + \gamma|\nabla\phi| + \eta E(D\phi, \mathrm{D}^2\phi), \qquad (43)$$

where $\mathrm{D}\phi$ and $\mathrm{D}^2\phi$ are collections first and second derivatives of ϕ, respectively. This equation is solved on a discrete grid using an *up-wind* scheme gradient calculations, central differences for the curvature, and forward finite differences in time. The LSSM library uses the *sparse-field* method described in Section 6.3 and in [21].

Thus, the LSSM library offers the following capabilities:

1. Creates an initial model (with associated active set) from a volume.
2. Calculates $\Delta u^n_{i,j,k}$ and Δt using virtual functions (defined by subclasses) that describe $\boldsymbol{F}$ and G, and parameters (values set by the subclass) α, β, γ, and η.
3. Performs an update on the values of $u^n_{i,j,k}$.
4. Maintains the list of active grid points and updates the *layers* around those points in order to maintain a neighborhood from which to calculate subsequent updates.
5. Provides access to the volume that defines $u^n_{i,j,k}$ and the linked list of active grid points.

Given the volume defining $u^n_{i,j,k}$, one can then rely on the functionality of the volume library for subsequent processing, file I/O, or surface extraction.

8.2.2 Structure and Philosophy of the LSSM Library

The library is organized (mostly for ease of development) into a base class, `LevelSetModel`, and a derived class, `VoxModel`. The base class does all of the book keeping associated with the active set and surrounding *layers*, the link lists associated with those sets, and initializing the model. Thus it adds and removes voxels from the active set (and surrounding layers) in response to an update operation. The base class assumes that the subclasses know how to update individual voxels. Applications are built by subclassing `VoxModel` and redefining a small set of virtual functions that control the movement of the model.

The subclass, `VoxModel`, performs update on the grid points in the active set of the form given in Equation 18, using functions $\boldsymbol{F}$ and G and parameters α, β, γ, and η. It also calculates the maximum Δt that ensures stability. Thus a user who wishes to perform a surface deformation using the LSSM library, would create subclass of `VoxModel` and define the appropriate virtual functions and set the parameters to achieve the desired behavior.

8.2.3 The `LevelSetModel` Object

The `LevelSetModel` contains a volume of values, a volume of status flags, five lists (one active list, two inside lists, and two outside lists), and three parameters that determine the origin of the coordinate system form which the model performs its calculations.

There are two constructors, `LevelSetModel()` and `LevelSetModel( const VISVolume<float> &)`. The first simply initializes the data structure, and the second also set the values of the model volume (`_values`) to the input. Once the values have been set, one can create an initial volume from those values by calling `constructLists()`, which can also take a floating-point argument that controls the scaling of the input relative to a local distance transform near the zero set.

The list that keeps track of the active set, called `_active_list`, keeps track of the location of those grid points and a single floating-point value, which stores the change in their values from one iteration to the next.

Another important methods for users of this object is `update(float)`, which changes the grey-scale values of the grid for the active set according to the values stored in `_active_list`, and updates the status of elements on the active list as well as the values and status of nearby layers (2 inside and 2 outside). The floating point argument is the value of Δt from Equation 18, and the return value is the maximum change that occurred on the active set. Finally, the method `iterate()` calls the virtual method `calculate_change`, a virtual function which sets the values of $\Delta u^n_{i,j,k}$ and returns the maximum value of Δt for stability, and then calls `update`. For this object the function `calculate_change` performs some trivial (i.e., useless) operation.

8.2.4 The `VoxModel` Object

The `VoxModel` object is a subclass of `LevelSetModel`, and it add three things to the base class.

1. `calculate_change()` is redefined to implement the surface deformation described in Equation 43.
2. The virtual functions are declared for F (called `force`) and G (called `grow`). These functions are defined to return zero for this object.

3. The parameters that control the relative influence of the various terms are read from file by a routine `load_params`.

4. A method `rescale(float)` is defined, which resamples the volume of grid-point values into a new volume with different resolution and redefines the lists (and thereby the model) in this new volume. This method is for performing coarse-to-fine deformation procedures.

8.3 Example: 3D Shape Metamorphasis

The `Morph` object allows one to construct a sequence of volumes or surface meshes using the 3D shape metamorphasis technique described in Section 7.1, which was first proposed by Whitaker and Breen [20]. This technique relies distance transforms for both the source and target objects and uses a LSSMs to manipulate the shape of the source so that it coincides with the target. The surface deformation that describes this behavior is

$$\frac{\partial \boldsymbol{x}}{\partial t} = \beta G\left(T(\boldsymbol{x})\right) \boldsymbol{N}(\boldsymbol{x}), \tag{44}$$

where $G(\boldsymbol{x})$ is simply the distance transform (or some monotonic function thereof) of the target, and T is a coordinate transformation that aligns the source and target objects. The level-set formulation of this is

$$\frac{\partial \phi(\boldsymbol{x}, t)}{\partial t} = \beta G\left(T(\boldsymbol{x})\right) |\nabla \phi|. \tag{45}$$

The morphing process consists of several steps:

1. Read in distance transforms (in the form of volumes) for both source and target.

2. Initialize the LSSM by fitting it to the zero set of the source distance transform.

3. Update the LSSM according to Equation 45.

4. Save intermediate volumes/surfaces at regular intervals.

The remainder of this section lists the code and comments for three files, morph.h (which declares the `Morph` object), morph.C (which defines the methods) and main.C (which performs all of the I/O and uses the `Morph` object to construct a sequence of shapes.

8.4 Morph.h

```
//
// morph.h
//
//

#ifndef iris_morph_h
#define iris_morph_h

#include "voxmodel/voxmodel.h"
#include "matrix/matrix.h"

#define INIT_STATE 0
#define MORPH_STATE 1
//
// This is the morph object.  It uses all of the machinery of the base
// class to manipulate level sets.  It needs to have an initial volume
// and a final volume (which would typically be the distance transform,
// it might need a 3D transformation, and it needs to redefine the
// virtual function "grow", which takes 6 floats as input, the position
// followed by the normal vectors (all will calculated and passed into
// this method by the base class).  It might also have a state, that
// indicates whether or not it's been initialized.
//
// Functions not defined here should be defined in "morph.C"
//
class Morph: public VoxModel
{
  protected:
    VISVolume<float> _dist_source;
    VISVolume<float> _dist_target;
    VISMatrix _transform;
//
// This is the function that is used by the base class to manipulate the
level
// set.  You can define it to by anything you want.  For this object, it
will
// return a value from the distance transform of the target.
//
```

```
    virtual float grow(float x, float y, float z,
         float nx, float ny, float nz);

// There are two states.  In the first state, the model is trying to fit
// to the input data.  In this way the models starts by looking just like

// the input data
    int _state;

  public:

    Morph(const Morph& other)
    {
 _dist_target = other._dist_target;
 _initial = other._initial;
 _state = MORPH_STATE;
 _transform = VISVISMatrix(3, 3);
 _transform.identity();
// initialize();
    }

    Morph(VISVolume<float> init, VISVolume<float> d)
 :VoxModel()
    {
 _dist_target = d;
 _initial = init;
 _state = MORPH_STATE;
 _transform = VISVISMatrix(3, 3);
 _transform.identity();
// initialize();
    }

    void initialize();

// for this object I assume that the transform is just a matrix.
// but it could be anything
    void transform(const VISVISMatrix& t)
    { _transform = t;}

    const VISVISMatrix& transform()
    { return(_transform);}
```

```
    void distance(const VISVolume<float> d)
    { _dist_target = d;}
    VISVolume<float> distance()
    { return(_dist_target);}

};
#endif
```

8.5 Morph.C

```
#include "morph.h"
#include "util/geometry.h"
#include "util/mathutil.h"


//
// this is the virtual function, that  is the guts of it all.
//

float Morph::grow(float x, float y, float z,
    float nx, float ny, float nz)
{

// this says you are in the morph state (things have been initialized)
    if (_state == MORPH_STATE)
 {
 float xx, yy, zz;
 VISPoint p(4u);
 p.at(0) = x;
 p.at(1) = y;
 p.at(2) = z;
 p.at(3) = 1;
 VISPoint p_tmp;
// this is where you could put some other transform.
 p_tmp = _transform*p;

 xx = p_tmp.x();
 yy = p_tmp.y();
```

```
 zz = p_tmp.z();

// make sure you are not out of the bounds
// of your distance volume.
 if (_dist_target.checkBounds(xx, yy, zz))
// if not, get the distance (use trilinear interpolation).
     return(_dist_target.interp(xx, yy, zz));
 else
     return(0.0f);
    }
else
    {
// if you are still initializing, then move toward the zero set of
// your initial case
 if (_initial.checkBounds(x, y, z))
     return(_initial.interp(x, y, z));
 else
     return(0.0f);
    }
}

// this makes the model look like the input.
#define INIT_ITERATIONS 5
void Morph::initialize()
{
    _values = _initial;
    int state_tmp = _state;
    _state = INIT_STATE;
    construct_lists(DIFFERENCE_FACTOR);
// these couple of iterations are required to make sure that the zero
// sets of the model match the zero sets of the
//
    for (int i = 0; i < INIT_ITERATIONS; i++)
 {
// limit the dt to 1.0 so that the model settles in to a solution
     update(::min(calculate_change(), 1.0f));
 }
    _state = state_tmp;
}
```

8.6 Main.C

```
#include "vol/volume.h"
#include "vol/volumefile.h"
#include "image/imagefile.h"
#include "morph.h"
#include <string.h>


const int V_HEIGHT = (40);
const int V_WIDTH = (40);
const int V_DEPTH = (40);

#define XY_RADIUS (12)  // this matches the 2.5D data generated in
torus.C
#define T_RADIUS (4)  // this matches the 2.5D data generated in torus.C
#define S_RADIUS (12)  // radius of a sphere

#define B_WIDTH (20.0f)
#define B_HEIGHT (60.0f)
#define B_DEPTH (20.0f)

#define B_CENTER_X (12.0f)
#define B_CENTER_Y (32.0f)
#define B_CENTER_Z (12.0f)

float sphere(unsigned x, unsigned y, unsigned z);
float torus(unsigned x, unsigned y, unsigned z);
float cube(unsigned x, unsigned y, unsigned z);


// This is a program that does the morph.  If you give it two
// arguments, it reads the initial model and the dist trans for the
// final model from the two file names given, otherwise, it makes a
sphere
// and deforms it into a torus

main(int argc, char** argv)
{
```

```
VISVolume<float> vol_source, vol_target;
VISVolumeFile vol_file;
int i;
char fname[80];

vol_source = VISVolume<float>(25,65,25);
vol_source.evaluate(cube);

if (argc > 2)
    {
// read in the sourceing model
 vol_source = VISVolume<float>(vol_file.read_float(argv[1]));
// read in the dist trans of the final model
     vol_target = VISVolume<float>(vol_file.read_float(argv[2]));
    }
else
// make up some volumes
    {
 vol_source = VISVolume<float>(V_WIDTH, V_HEIGHT, V_DEPTH);
 vol_source.evaluate(sphere);
 vol_target = VISVolume<float>(V_WIDTH, V_HEIGHT, V_DEPTH);
 vol_target.evaluate(torus);
    }

// create morph object
Morph morph(vol_source, vol_target);
// loads in some parameters (for morphing these are all zero but one)
// i.e.
//
//
//
//
morph.load_parameters("morph_params");
morph.initialize();
vol_file.write_float(morph.values(), "morph0.flt");

float dt;

// do 150 iterations for your model to get from start to finish
// probably don't need this many iterations
```

```
for (i = 0; i < 150; i++)
    {
 dt = morph.calculate_change();
// limit dt to 0.5 so that model never overshoots goal
 dt = min(dt, 0.5f);
 morph.update(dt);

 printf("iteration %d dt %f\n", i, dt);

 if (((i + 1)%10) == 0)
 {
// save every tenth volume
     sprintf(fname, "morph_out.%d.dat", i + 1);
     vol_file.write_float(morph.values(), fname);
 }
    }

// save a surface model (i.e. marching cubes).
vol_file.march(0.0f, morph.values(), ‘‘morph_final.iv’’);

printf("done\n");

}
```

References

[1] J. Blinn, "A generalization of algebraic surface drawing," *ACM Trans. on Graphics*, vol. 1, pp. 235–256, March 1982.

[2] S. Muraki, "Volumetric shape description of range data using "blobby model"," in *SIGGRAPH '91 Proceedings* (T. W. Sederberg, ed.), pp. 227–235, July 1991.

[3] G. Taubin, "An accurate algorithm for rasterizing algebraic curves and surfaces," *IEEE Computer Graphics & Applications*, March 1994.

[4] D. Breen, S. Mauch, and R. Whitaker, "3d scan conversion of csg models into distance, closest-point and colour volumes," in *Volume Graphics* (M. Chen, A. Kaufman, and R. Yagel, eds.), pp. 135–158, London: Springer, 2000.

[5] W. Lorenson and H. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *Computer Graphics*, vol. 21, no. 4, pp. 163–169, 1982.

[6] M. Levoy, "Display of surfaces from volume data," *IEEE Computer Graphics and Applications*, vol. 9, no. 3, pp. 245–261, 1990.

[7] R. A. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering," in *SIGGRAPH '88 Proceedings*, pp. 65–74, August 1988.

[8] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International Journal of Computer Vision*, vol. 1, pp. 321–323, 1987.

[9] D. Terzopoulos and K. Fleischer, "Deformable models," *The Visual Computer*, vol. 4, pp. 306–331, December 1988.

[10] S. Osher and J. Sethian, "Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations," *Jrnl. of Comp. Phys.*, vol. 79, pp. 12–49, 1988.

[11] J. Sethian, *Level Set Methods and Fast Marching Methods*. Cambridge: Cambridge University Press, second ed., 1999.

[12] J. A. Sethian, "A fast marching level set method for monotonically advancing fronts," *Proc. Nat. Acad. Sci.*, vol. 93, no. 4, pp. 1591–1595, 1996.

[13] J. A. Sethian, *Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision, and Material Sciences*. Cambridge University Press, 1996.

[14] S. Osher and R. Fedkiw, "Level set methods: An overview and some recent results," Tech. Rep. 00-08, UCLA Center for Applied Mathematics, Department of Mathematics, University of California, Los Angeles, 2000.

[15] L. Alvarez and J.-M. Morel, "A morphological approach to multiscale analysis: From principles to equations," in *Geometry-Driven Diffusion in Computer Vision* (B. M. ter Haar Romeny, ed.), pp. 4–21, Kluwer Academic Publishers, 1994.

[16] V. Caselles, R. Kimmel, and G. Sapiro, "Geodesic active contours," in *Fifth Int. Conf. on Comp. Vision*, pp. 694–699, IEEE, IEEE Computer Society Press, 1995.

[17] B. B. Kimia and S. W. Zucker, "Exploring the shape manifold: the role of conservation laws.," in *Shape in Picture: the mathematical description of shape in greylevel images* (Y.-L. O, A. Toet, H. Heijmans, D. H. Foster, and P. Meer, eds.), Springer-Verlag, 1992.

[18] R. Malladi, J. A. Sethian, and B. C. Vemuri, "Shape modeling with front propagation: A level set approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 2, pp. 158–175, 1995.

[19] R. T. Whitaker and D. T. Chen, "Embedded active surfaces for volume visualization," in *SPIE Medical Imaging 1994*, (Newport Beach, California), 1994.

[20] R. Whitaker and D. Breen, "Level-set models for the deformation of solid objects," in *The Third International Workshop on Implicit Surfaces*, pp. 19–35, Eurographics, 1998.

[21] R. T. Whitaker, "A level-set approach to 3D reconstruction from range data," *Int. Jrnl. of Comp. Vision*, vol. October, no. 3, pp. 203–231, 1998.

[22] R. T. Whitaker, "Algorithms for implicit deformable models," in *Fifth Intern. Conf. on Comp. Vision*, IEEE, IEEE Computer Society Press, 1995.

[23] S. Kichenassamy, A. Kumar, P. Olver, A. Tannenbaum, and A. Yezzi, "Gradient flows and geometric active contour models," in *Fifth Int. Conf. on Comp. Vision*, pp. 810–815, IEEE, IEEE Computer Society Press, 1995.

[24] A. Yezzi, S. Kichenassamy, A. Kumar, P. Olver, and A. Tannenbaum, "A geometric snake model for segmentation of medical imagery," *IEEE Transactions on Medical Imaging*, vol. 16, pp. 199–209, April 1997.

[25] L. Lorigo, O. Faugeraus, W. Grimson, R. Keriven, and R. Kikinis, "Segmentation of bone in clinical knee MRI using texture-based geodesic active contours," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI '98)* (W. Wells, A. Colchester, and S. Delp, eds.), pp. 1195–1204, October 1998.

[26] S. Osher and J. Sethian, "Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations," *Jrnl. of Comp. Phys.*, vol. 79, pp. 12–49, 1988.

[27] X. Xue and R. Whitaker, "Variable-conductance, level-set curvature for image denoising," in *IEEE International Conference on Image Processing*, p. To Appear., October 2001.

[28] D. Adalstein and J. A. Sethian, "A fast level set method for propagating interfaces," *Jrnl. of Comp. Phys.*, pp. 269–277, 1995.

[29] D. Breen and R. Whitaker, "A level-set approach to 3D shape metamorphosis," *IEEE Transactions on Visualization and Computer Graphics*, p. To Appear., 2001.

[30] A. Lerios, C. D. Garfinkle, and M. Levoy, "Feature-Based volume metamorphosis," in *SIGGRAPH '95 Conference Proceedings* (R. Cook, ed.), Annual Conference Series, pp. 449–456, Addison Wesley, Aug. 1995.

[31] J. Kent, W. Carlson, and R. Parent, "Shape transformation for polyhedral objects," in *SIGGRAPH '92 Proceedings*, pp. 47–54, July 1992.

[32] J. Rossignac and A. Kaul, "AGRELS and BIPs: Metamorphosis as a bezier curve in the space of polyhedra," *Computer Graphics Forum (Eurographics '94 Proceedings)*, vol. 13, pp. C–179–C–184, September 1994.

[33] J. F. Hughes, "Scheduled Fourier volume morphing," in *Computer Graphics (SIGGRAPH '92 Proceedings)* (E. Catmull, ed.), vol. 26, pp. 43–46, July 1992.

[34] B. Payne and A. Toga, "Distance field manipulation of surface models," *IEEE Computer Graphics and Applications*, vol. 12, no. 1, pp. 65–71, 1992.

[35] D. Cohen-Or, D. Levin, and A. Solomivici, "Three-dimensional distance field metamorphosis," *ACM Transactions on Graphics*, vol. 17, no. 2, pp. 116–141, 1998.

[36] P. Getto and D. Breen, "An object-oriented architecture for a computer animation system," *The Visual Computer*, vol. 6, pp. 79–92, March 1990.

[37] D. Breen, S. Mauch, and R. Whitaker, "3D scan conversion of CSG models into distance volumes," in *Proceedings of the 1998 Symposium on Volume Visualization*, pp. 7–14, ACM SIGGRAPH, October 1998.

[38] A. Middleditch and K. Sears, "Blend surfaces for set theoretic volume modeling systems," in *SIGGRAPH '85 Proceedings*, pp. 161–170, July 1985.

[39] J. Bloomenthal and K. Shoemake, "Convolution surfaces," in *SIGGRAPH '91 Proceedings* (T. W. Sederberg, ed.), pp. 251–257, July 1991.

[40] M. Desbrun and M.-P. Gascuel, "Animating soft substances with implicit surfaces," in *SIGGRAPH '95 Proceedings*, pp. 287–290, August 1995.

[41] R. T. Whitaker, "Volumetric deformable models: Active blobs," in *Visualization In Biomedical Computing 1994* (R. A. Robb, ed.), (Mayo Clinic, Rochester, Minnesota), pp. 122–134, SPIE, 1994.

[42] R. T. Whitaker, "A level-set approach to 3D reconstruction from range data," *Int. Jrnl. of Comp. Vision*, vol. October, no. 3, pp. 203–231, 1998.

[43] G. Turk and M. Levoy, "Zippered polygon meshes from range images," in *Proc. of SIGGRAPH '94*, pp. 311–318, ACM SIGGRAPH, August 1994.

[44] Y. Chen and G. Médioni, "Fitting a surface to 3-D points using an inflating ballon model," in *Second CAD-Based Vision Workshop* (A. Kak and K. Ikeuchi, eds.), vol. 13, pp. 266–273, IEEE, 1994.

[45] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proc. of SIGGRAPH '96*, pp. 303–312, ACM SIGGRAPH, August 1996.

[46] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," *Computer Graphics*, vol. 26, no. 2, pp. 71–78, 1992.

[47] A. Hilton, A. J. Stoddart, J. Illingworth, and T. Windeatt, "Reliable surface reconstruction from multiple range images," in *Euro. Conf. on Comp. Vision*, Springer-Verlag, 1996.

[48] C. Bajaj, F. Bernardini, and G. Xu, "Automatic reconstruction of surfaces and scalar fields from 3D scans," in *SIGGRAPH '95 Proceedings*, pp. 109–118, August 1995.

Geometric Surface Processing via Normal Maps

Tolga Tasdizen
School of Computing, University of Utah
and
Ross Whitaker
School of Computing, University of Utah
and
Paul Burchard
Department of Mathematics, UCLA
and
Stanley Osher
Department. of Mathematics, UCLA

We propose that the generalization of signal and image processing to surfaces entails filtering the normals of the surface, rather than filtering the positions of points on a mesh. Using a variational strategy, penalty functions on the surface geometry can be formulated as penalty functions on the surface normals, which are computed using geometry-based shape metrics and minimized using fourth-order gradient descent partial differential equations (PDE). In this paper, we introduce a two step approach to implementing geometric processing tools for surfaces: (i) operating on the normal map of a surface, and (ii) manipulating the surface to fit the processed normals. Iterating this two-step process, we can efficiently implement geometric fourth-order flows by solving a set of coupled second-order PDEs. The computational approach uses level set surface models; therefore, the processing does not depend on any underlying parameterization. This paper will demonstrate that the proposed strategy provides for a wide range of surface processing operations, including edge-preserving smoothing and high-boost filtering. Furthermore, the generality of the implementation makes it appropriate for very complex surface models, e.g. those constructed directly from measured data.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Curve, surface, solid and object representations.*

Additional Key Words and Phrases: surface fairing, geometric surface processing, anisotropic diffusion, high-boost filtering, level sets.

1. INTRODUCTION

The fundamental principles of signal processing give rise to a wide range of useful tools for manipulating and transforming signals and images. The generalization of these principles

Author's address: Tolga Tasdizen, 50 S. Central Campus Drive, School of Computing, University of Utah Salt Lake City, UT 84112-9205. Phone: (801) 585-3742. E-mail: tolga@sci.utah.edu
Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.
© 20YY ACM 0000-0000/20YY/0000-0001 $5.00

to the processing of 3D surfaces has become an important problem in computer graphics, visualization, and vision. For instance, 3D range sensing technologies produce high resolution descriptions of objects, but they often suffer from noise. Medical imaging modalities such as MRI and CT scans produce large volumes of scalar or tensor measurements, but surfaces of interest must be extracted through some segmentation process or fitted directly to the measurements. These surfaces typically contain topological artifacts such as holes and unconnected pieces.

The goal of this paper is to introduce a new surface processing strategy that is flexible, general, and geometric. By *flexible* we mean that the framework should provide a basis for a broad variety of capabilities, including surface processing tools that resemble the state-of-the-art in image processing algorithms. The proposed methods should apply to a *general* class of surfaces. Users should be able to process complex surfaces of arbitrary and changing topology, and obtain meaningful results with very little *a priori* knowledge about the shapes. By *geometric* we mean that output of surface processing algorithms should depend on *surface shape and resolution*, but should be independent of arbitrary decisions about the representation or parameterization.

The work presented in this paper is based on the proposition that the natural generalization of image processing to surfaces is via the *surface normal vectors*. Thus, a smooth surface is one that has smoothly varying normals. Penalty functions on the surface normals typically give rise to fourth-order partial differential equations (PDE). Our strategy is to use a two step approach: (i) operating on the normal map of a surface, and (ii) manipulating the surface to fit the processed normals. Iterating this two-step process, we can efficiently implement fourth-order flows by solving a set of coupled second-order PDEs. In this light, the differences between surface processing and image processing are threefold:

(1) Normals are vector valued and constrained to be unit length; the processing techniques must accommodate this.

(2) Normals live on a manifold (the surface) and cannot be processed using a flat metric, as is typically done with images.

(3) Normals are coupled with the surface shape, thus the normals should drag the surface along as their values are modified during processing.

This paper presents an implementation that represents surfaces as the level sets of volumes and computes the processing of the normals and the deformation of the surfaces as solutions to a set of PDEs. In some applications, such as animation, models are manually generated by a designer, and the parameterization is not arbitrary but is an important aspect of the geometric model. In these cases, mesh-based processing methods offer a powerful set of tools, such as hierarchical editing [Guskov et al. 1999], which are not yet possible with the proposed representation. However, in other applications, such as 3D segmentation and surface reconstruction [Malladi et al. 1995; Whitaker 1998], the processing is data driven, surfaces can deform quite far from their initial shapes and change topology; hence, user intervention is not practical. Furthermore, when considering processes other than isotropic smoothing, such as nonlinear smoothing, the creation or sharpening of small features can exhibit noticeable effects of the mesh topology—the creation of new features requires changes in the mesh parameterization. In contrast, the underlying grid for level sets is independent of the surface shape; therefore, the only limitation for the creation of new features is the resolution of the grid. Hence, the use of a level set formulation en-

ables us to achieve a "black box" behavior and build surface processing techniques that are especially useful when processing measured data.

We have introduced an anisotropic diffusion for surfaces based on processing the normal map in [Tasdizen et al. 2002]. This paper discusses the mathematical foundations of the normal map processing strategy in detail and provides details of the numerical implementation as well as introducing high-boost filtering of normals as a new surface processing tool. The specific contributions are:

(1) a novel approach based on surface normals for geometric processing of surfaces;
(2) a numerical method for solving geometric fourth-order level set equations for surfaces in two simpler steps, thereby avoiding the explicit computation of unstable high-order derivatives; and
(3) examples of three geometric surface processing algorithms with applications to complex data sets.

The rest of this paper is organized as follows. We will discuss related surface processing work in Section 2. In Section 3, we formulate our splitting approach for solving geometric fourth-order level set equations for surfaces. In the limit, this approach is equivalent to solving the full, fourth-order flow, Appendix (A), but it generalizes to a wide range of processes and makes no assumptions about the shapes of the solutions. In Section 4, we show results for isotropic and anisotropic diffusion. To demonstrate the flexibility of the proposed framework, we will also show results of high-boost surface filtering implemented with our framework in Section 5. The numerical implementations of our approach is covered in Appendix (B). Conclusions and directions for future work will be discussed in Section 6.

2. RELATED WORK

The majority of surface processing research has been in the context of *surface fairing* with the motivation of smoothing surfaces to create aesthetically pleasing models. Surface fairing can be accomplished either by minimizing an energy function that favors smooth surfaces [Moreton and Séquin 1992; Welch and Witkin 1992; Halstead et al. 1993; Welch and Witkin 1994] or by applying smoothing filters [Taubin 1995; Desbrun et al. 1999; Guskov et al. 1999]. Energy minimization is a global method whereas filtering uses local neighborhoods. In the rest of this section, we review related work in these two categories. An approach that falls between these two extremes is based on Wiener filtering which utilizes arbitrary local spectral properties of the mesh [Alexa 2002].

Energy functions can depend on the geometry of the surface or the parameterization. Geometric functions make use of invariants such as principal curvatures, which are parameterization independent, intrinsic properties of the surface. Therefore, geometric approaches produce results that are not affected by arbitrary decisions about the parameterization; however, geometric invariants are nonlinear functions of surface derivatives that are computationally expensive to evaluate. Parameterization dependent functions are linear substitutes for geometric invariants.

One way to smooth a surface is to incrementally reduce its surface area. This can be accomplished by mean curvature flow (MCF), a second-order PDE,

$$\frac{\partial \mathbf{x}}{\partial t} = -H\mathbf{N} = -\left(\frac{\kappa_1 + \kappa_2}{2}\right)\mathbf{N} \tag{1}$$

where κ_1, κ_2 are the principal curvatures and H is the mean curvature at a point $\mathbf{x}$ on the surface, $\mathbf{N}$ is the surface normal, and the parameter t tracks the deforming surface shape. For parameterized surfaces, the membrane energy function, a linear substitute for surface area, is

$$\int_\Omega X_u^2 + X_v^2 \, du \, dv \tag{2}$$

where $X(u,v)$ and Ω are surface parameterization and its domain, respectively. The variational derivative of (2) is the Laplacian

$$\Delta X = X_{uu} + X_{vv} \, , \tag{3}$$

which is a linear substitute for mean curvature; however, they are equivalent only if the parameterization is orthonormal everywhere. These methods generally produce unsatisfactory results due to inherent limitations such as inability to preserve features, a systematic shrinking, and the introduction of high-curvature singularities.

A second-order energy function is the integral of *total curvature* over the surface S

$$\int_S \kappa_1^2 + \kappa_2^2 \, dS \tag{4}$$

which has been shown to deform surfaces into spheres when minimized [Polden 1997]. We will refer to (4) as the *total curvature penalty* which should not be confused with the local quantity *total curvature*. The total curvature penalty is a geometric (invariant) property of the surface that can be minimized by a fourth-order PDE which is very difficult to solve. The mesh fairing approach of [Welch and Witkin 1994], which minimizes (4), fits local polynomial basis functions to local neighborhoods for the computation of total curvature. These polynomial basis functions range from full quadratic polynomials to constrained quadratics and planar approximations. Depending on the complexity of the local neighborhood, the algorithm must choose, at each location, which basis to employ. Ambiguities result at locations where multiple bases provide equally good representations.

If we penalize the parameterization (i.e. non-geometric), equation (4) becomes the thin plate energy function

$$\int_\Omega X_{uu}^2 + 2X_{uv}^2 + X_{vv}^2 \, du \, dv \tag{5}$$

where X and Ω are as defined for (2). The variational derivative of (5) is the linear biharmonic operator

$$\Delta^2 X = X_{uuuu} + 2X_{uuvv} + X_{vvvv} \tag{6}$$

which is a fourth-order operator used for surface fairing [Welch and Witkin 1992].

Moreton and Séquin propose a geometric energy function that penalizes the variation of principle curvatures [1992]. This function has a sixth-order variational derivative which requires very large computation times. The analysis and implementation of general energy functions above second order remains an open problem, which is beyond the scope of this paper. Evidence in this paper and elsewhere [Desbrun et al. 1999; Schneider and Kobbelt 2000] suggests that fourth-order geometric flows form a sufficient foundation for a general, geometric surface processing system.

Taubin pioneers the linear filter based approaches to surface fairing. He observes that simple Gaussian filtering associated with the membrane energy causes shrinkage [1995].

He eliminates this problem by designing a low pass filter using a weighted average of the Laplacian (3) and the biharmonic operator (6). The weights must be fine-tuned to obtain the non-shrinking property. Analyzed in the frequency domain, this low-pass filter can be seen as a Gaussian smoothing shrinking step followed by an unshrinking step. Taubin shows that any polynomial transfer function in the frequency domain can be implemented with this method [1996]. A related approach in which surfaces are smoothed by simultaneously solving the membrane (2) and thin plate (5) energy functions is proposed in [Kobbelt et al. 1998]. Desbrun et al. use implicit integration to build a computationally efficient method for mesh fairing [1999]. Guskov et al. defines down- and up-sampling tools and smoothing filters for irregular meshes to build a multiresolution mesh processing framework [1999]. Their work is based on a generalized low pass filter which uses a non-uniform relaxation operator that minimizes a locally weighted quadratic energy of second-order differences on the mesh.

The techniques proposed in this paper are also related to that of [Chopp and Sethian 1999], who derive the intrinsic Laplacian of curvature for an implicit curve, and solve the resulting fourth-order nonlinear PDE. However, their method does not generalize to implicit surfaces. Moreover, they argue that the numerical methods used to solve second-order flows are not practical, because they lack long term stability. They propose several new numerical schemes, but none are found to be completely satisfactory due to their slow computation and inability to handle singularities. As a generalization of this PDE for surfaces, Schneider and Kobbelt propose using the intrinsic Laplacian of mean curvature, $\Delta_B H$, for meshes, where Δ_B is the Laplace-Beltrami operator, i.e. the Laplacian for parameterized surfaces [2000]. However, that approach works only for meshes, and relies on analytic properties of the steady-state solutions, $\Delta_B H = 0$, by fitting surface primitives that have those properties. Thus, the formalism does not generalize well to applications, such as surface reconstruction, where the solution is a combination of measured data and a fourth-order smoothing term. Also, it does not apply to other types of smoothing processes, such as anisotropic diffusion that minimizes nonlinear feature-preserving penalties. We solve a more general class of surface flows with a variational basis in an effective, stable splitting method.

An example of a splitting strategy can be found in [Ballester et al. 2001], where the authors penalize the smoothness of a vector field while simultaneously forcing the gradient directions of a gray scale image to closely match the vector field. The penalty function on the normal field is proportional to the divergence of the normal vectors. It forms a high-order interpolation function, which is shown to be useful for image inpainting—recovering missing patches of data in 2D images. The strategy of simultaneously penalizing the divergence of a normal field and the mismatch of this field with the image gradient is closely related to the total curvature penalty function used in this paper. However, our formulation emphasizes the processing of normals on an arbitrary surface manifold (rather than the flat geometry of an image), with an explicit relationship to fourth-order surface flows. Furthermore, this paper establishes new directions for surface flows— toward edge-preserving surface smoothing and feature enhancement. Our proposed PDE splitting approach is related to the methods in [Schneider and Kobbelt 2000], which we discuss in detail in Section 3.

Our splitting method requires diffusing unit-length vectors on a non-flat manifold. Perona proposes a method for diffusing orientation-like quantities on flat manifolds [1998]. This method solves the problem of diffusing 2D unit vectors as a 1D problem of angle

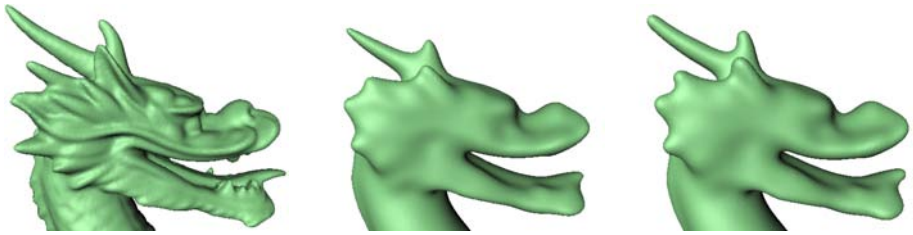


Fig. 1. Second- vs. fourth-order surface smoothing. From left to right: Original model, mean curvature flow, and isotropic fourth-order surface flow.

diffusion; however, it is not rotationally invariant and it does not generalize to the diffusion of unit vectors in higher dimensions. Several authors use harmonic maps theory to solve the diffusion of unit vectors defined on higher dimensional non-flat manifolds, e.g. surfaces [Bertalmio et al. 2001; Tang et al. 2000]. Hence, their methods are closely related to some of the problems we solve in this, but they do not provide a solution to the coupling between surface shape and the unit vectors because their goal is not surface processing.

3. MINIMIZING TOTAL CURVATURE

One of the underlying strategies of our approach is to use *geometric surface processing*, where the output of the process depends only on the shape of the input surface, and does not contain artifacts from the underlying parameterization. The motivation for this strategy is discussed in detail in [Schneider and Kobbelt 2001], where the influence of the parameterization on surface fairing results is clearly shown, and higher-order nonlinear geometric flows are proposed as the solution.

As an illustration of the importance of higher-order geometric processing, consider the results in Figure 1, which demonstrates the differences between processing surfaces with mean curvature flow (MCF) and the isotropic fourth-order PDE that minimizes the total curvature penalty (4). The amount of smoothing for both processes in this example were chosen to be qualitatively similar, and yet important differences can be observed on the smaller features of this model. MCF has shortened the horns of the original model, and yet they remain sharp—not a desirable behavior for a "smoothing" process. This behavior for MCF is well documented as a pinching off of cylindrical objects and is expected from the variational point of view: MCF minimizes surface area and therefore will quickly eliminate smaller parts of a model. Sapiro discusses volume preserving forms of second-order flows [2001], but these processes compensate by enlarging *the object as a whole*, which exhibits, qualitatively, the same behavior on small features. The isotropic fourth-order PDE, on the other hand, preserves the structure of these features much better *while* smoothing them as can be seen in Figure 1. Note that all of the surfaces in this paper are represented and processed volumetrically. To display the results, we render a mesh obtained with the Marching Cubes algorithm [Lorenson and Cline 1987].

We propose a two-step solution based on letting the surface shape to lag the normals as they are filtered and then refitting the surface to the normals by a separate process. For general fourth-order surface flows such as isotropic and anisotropic diffusion, both of these steps involve solving second-order PDEs. The first second-order PDE is used for minimizing a penalty function on the normals. The other second-order PDE minimizes the discrepancy between the modified normals and the surface; in other words, it refits the

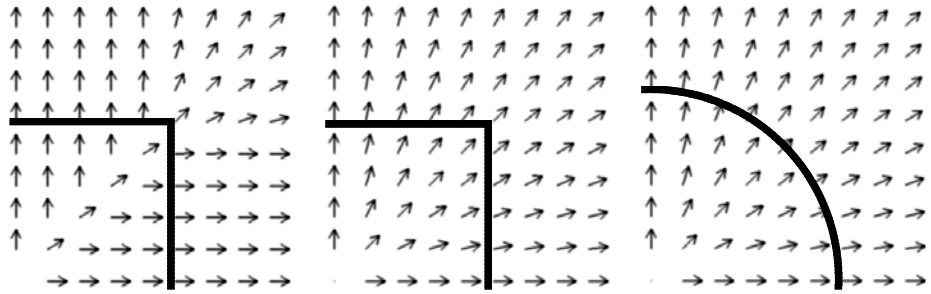


Fig. 2. Shown here in 2D, the process begins with a shape and constructs a normal map from the distance transform (left), modifies the normal map according to a PDE derived from a penalty function (center), and re-fits the shape to the normal map (right).

surface to the normals. Figure 2 shows this three step process graphically in 2D—shapes give rise to normal maps, which, when filtered give rise to new normal maps, which finally give rise to new shapes. The rest of this section is organized as follows. Level set methods are introduced in Section 3.1. We formulate total curvature as a function of the normal map and derive gradient descent minimizations for general functions of total curvature in Section 3.2; this gives rise to the first PDE mentioned above. The surface refitting process is discussed in Section 3.3; this gives rise to the other second-order PDE.

3.1 Level set methods

In this section, we briefly introduce the notation of level set methods. We can describe the deformation of a regular surface using the 3D velocity of each of its constituent points, *i.e.*, $\partial \mathbf{x}(t)/\partial t$ for all $\mathbf{x} \in S$. We represent the deforming surfaces implicitly as a function of the parameter t

$$S = \{\mathbf{x}(t) \mid \phi(\mathbf{x}(t), t) = 0\}, \tag{7}$$

where ϕ is the embedding function. Surfaces defined in this way divide a volume into two parts: inside ($\phi > 0$) and outside ($\phi < 0$). It is common to choose ϕ to be the signed distance transform of S, or an approximation thereof. The surface remains a level set of ϕ over time, and thus taking the total derivative with respect to time (using the chain rule) gives

$$\partial \phi/\partial t = -\nabla \phi \cdot \partial \mathbf{x}/\partial t \tag{8}$$

Because $\nabla \phi$ is proportional to the surface normal, $\partial \mathbf{x}/\partial t$ affects ϕ only in the direction of the normal— motion in any other direction is merely a change in the parameterization. This family of PDEs and the upwind scheme for solving them on a discrete grid is the methods of *level sets* [Osher and Sethian 1988]. For instance, using this framework and $\partial \mathbf{x}/\partial t$ from (1), the PDE on ϕ that describes the motion of a surface by mean curvature is

$$\partial \phi/\partial t = -\nabla \phi \cdot H\mathbf{N} = -\|\nabla \phi\| H. \tag{9}$$

Surface integrals of penalty functions have corresponding volume integrals which quantify the associated properties for the embedded surfaces of ϕ. A general surface penalty function based on the total curvature penalty (4) can be written for level set surfaces as

$$\mathscr{G}_\phi = \int_U G\left(\kappa_1^2 + \kappa_2^2\right) \| \nabla \phi \| \, d\mathbf{x}, \tag{10}$$

where $U \subset \Re^3$ is the volumetric domain of ϕ. When G is the identity function, (10) reduces to the total curvature penalty. The rest of this paper discusses methods for minimizing this penalty function in a stable and computationally efficient manner.

3.2 Total curvature of normal maps

In this section, we formulate total curvature of a surface from its normal map. Then, we derive the variational PDEs on the normal map that minimize functions of total curvature. When using implicit representations, one must account for the fact that derivatives of functions defined on the surface are computed by projecting their 3D derivatives onto the surface tangent plane. The 3×3 projection matrix for the implicit surface normal is

$$\mathbf{P} = \nabla\phi \otimes \nabla\phi / ||\nabla\phi||^2, \tag{11}$$

where $\otimes$ is the tensor product defined as $\mathbf{a} \otimes \mathbf{a} = \mathbf{a}\mathbf{a}^T$. Consequently, the projection matrix onto the surface tangent plane is $\mathbf{I} - \mathbf{P}$, where $\mathbf{I}$ is the identity matrix.

The local geometry of a surface can be described with the first and second fundamental forms, (I) and (II), respectively [DoCarmo 1976]. The eigenvalues of the matrix $(I)^{-1}(II)$, which we refer to as the *shape matrix*, are the principal curvatures of the surface independent of the parameterization. For an implicit surface, the shape matrix is obtained by differentiating the normal map and projecting the derivative onto the surface tangent plane. We define the differential of the normal map

$$\nabla\mathbf{N} = \left(\nabla\mathbf{N}_{(1)} \;\; \nabla\mathbf{N}_{(2)} \;\; \nabla\mathbf{N}_{(3)}\right)^T, \tag{12}$$

as the matrix whose rows are the gradient vectors of the components of $\mathbf{N}$ which we have denoted by $\mathbf{N}_{(i)}$ for $i = 1, 2$, and 3. Then the shape matrix is the projection $\nabla\mathbf{N}(\mathbf{I} - \mathbf{P})$, which measures the intrinsic change in the normals by mapping the differentials of $\mathbf{N}$ on to the tangent planes of ϕ. The Euclidean norm of the shape matrix is the sum of squared principal curvatures, i.e. total curvature,

$$\kappa^2 = \kappa_1^2 + \kappa_2^2 = ||(\nabla\mathbf{N})(\mathbf{I} - \mathbf{P})||^2. \tag{13}$$

We can use (13) to define an energy of the normal map that is analogous to the general energy function of ϕ defined in (10)

$$\mathscr{G}_{\mathbf{N}} = \int_U G(||(\nabla\mathbf{N})(\mathbf{I} - \mathbf{P})||^2)\, d\mathbf{x}. \tag{14}$$

The first variation of this energy with respect to the normals is a second order PDE. It is crucial to observe that, even though the projection operator $\mathbf{P}$ is a function of ϕ, it independent of $\mathbf{N}$ because we fix ϕ as we process $\mathbf{N}$. Hence, $\mathbf{P}$ does not increase the order of the first variation of (14). In contrast, taking the first variation of (10) with respect to ϕ directly, would have yielded a much harder to solve fourth order PDE on ϕ.

As we process the normal map to minimize (14), letting ϕ lag, we must ensure that the normal vectors maintain the unit length constraint. Solutions to constrained optimization problems defined on non-flat manifolds are discussed in [Bertalmio et al. 2001; Tang et al. 2000]. Using the method of Lagrange multipliers, we obtained the first variation of the constrained energy as

$$\frac{d\mathscr{G}}{d\mathbf{N}} = -(\mathbf{I} - \mathbf{N} \otimes \mathbf{N})\nabla \cdot \left[g\left(||(\nabla\mathbf{N})(\mathbf{I} - \mathbf{P})||^2\right)\nabla\mathbf{N}(\mathbf{I} - \mathbf{P})\right] \tag{15}$$

where g is the derivative of G with respect to its argument, κ^2. We will discuss several choices for G in Section 4. The projection operator, $\mathbf{I}-\mathbf{N}\otimes\mathbf{N}$, forces the changes to $\mathbf{N}$ to be perpendicular to itself in accordance with the unit length constraint. This operator is different from the other projection operator $\mathbf{I}-\mathbf{P}$ due to the decoupling of $\mathbf{N}$ and ϕ. Finally, the gradient descent PDE for the normals is $\partial\mathbf{N}/\partial t = -d\mathscr{G}/d\mathbf{N}$.

3.3 Surface evolution via normal maps

We have shown how to evolve the normals to minimize functions of total curvature; however, the final goal is to process the surface, which requires deforming ϕ. Therefore, the next step is to relate the deformation of the level sets of ϕ to the evolution of $\mathbf{N}$. Suppose that we are given the normal map $\mathbf{N}$ to some set of surfaces, but not necessarily level sets of ϕ—as is the case if we filter $\mathbf{N}$ and let ϕ lag. We can manipulate ϕ so that it fits the normal field $\mathbf{N}$ by minimizing a penalty function that quantifies the discrepancy between the gradient vectors of ϕ and the target normal map. That penalty function is

$$\mathscr{D}_\phi = \int_U D(\phi)\, d\mathbf{x}, \text{ where } D(\phi) = \sqrt{\nabla\phi\cdot\nabla\phi} - \nabla\phi\cdot\mathbf{N}. \tag{16}$$

The integrand, which is always a positive scalar, is proportional to the *sine* of the angle between the gradient vectors of ϕ and the target normal vectors.

The first variation of this penalty function with respect to ϕ is

$$\frac{d\mathscr{D}}{d\phi} = -\nabla\cdot\left[\frac{\nabla\phi}{||\nabla\phi||} - \mathbf{N}\right] = -\left[H^\phi - H^\mathbf{N}\right] \tag{17}$$

where H^ϕ is the mean curvature of the level set surface and $H^\mathbf{N}$ is half the divergence of the normal map. Then, the gradient descent PDE that minimizes (16) is $d\phi/dt = -||\nabla\phi||d\mathscr{D}/d\phi$. The factor of $||\nabla\phi||$, which is typical with level set formulations [Sethian 1999], comes from the fact that we are manipulating the shape of the level set, which is embedded in ϕ, as in (8). According to (17), the surface moves as the difference between its own mean curvature and that of the normal field.

The proposed splitting strategy for solving fourth-order level-set flows entails processing the normals and allowing ϕ to lag and then be refitted later, in a separate process. We have derived a gradient descent for the normal map based on a certain class of penalty functions that use the total curvature defined in Section 3.2. This process is denoted in Figure 3 as the $d\mathscr{G}/\mathbf{N}$ loop. The surface refitting to the normal map is formulated as a gradient descent in (17). This process is the $d\mathscr{D}/d\phi$ loop in Figure 3. The overall algorithm shown in Figure 3 repeats these two steps to minimize the penalty functions in terms of the surface. We refer to both of these processes, back-to-back, as one iteration of our algorithm. In Appendix (A) we will show that the overall process of concatenating these two second-order PDEs is equivalent to the fourth-order flow on the original surface. An alternate splitting approach for solving the same fourth-order level-set flows would be to simultaneously solve both second-order PDEs $d\mathscr{G}/d\mathbf{N}$ and $d\mathscr{D}/d\phi$ using a Lagrange multiplier instead of concatenating them as we have done. This approach was taken by Ballester et al. to solve the image inpainting problem [2001]. However, this approach uses a weighted sum of the two second-order PDEs; therefore, it is not clear whether it solves the original fourth-order flow. Moreover, in our case, due to the significant computational overhead of setting up the diffusion of normal vectors, it is more efficient to concatenate the two second-order PDEs and to do multiple consecutive iterations of $d\mathscr{G}/d\mathbf{N}$.

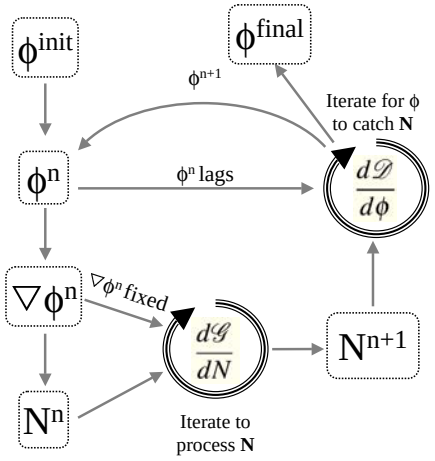


Fig. 3. Flow chart

4. ISOTROPIC AND ANISOTROPIC DIFFUSION

The flexible normal map energy minimization and surface refitting methodology introduced in Section 3 allows us to experiment with various forms of G in (14) that give rise to different classes of penalty functions. The choice of $G(\kappa^2) = \kappa^2$ leads to an isotropic diffusion. This choice works well for smoothing surfaces and eliminating noise, but it also deforms or removes important features. This type of smoothing is called isotropic because it corresponds to solving the heat equation on the normal map with a constant, scalar conduction coefficient, which is the same as Gaussian smoothing, for images. Isotropic diffusion is not particularly effective if the goal is to denoise a surface that has an underlying structure with fine features. This scenario is common when extracting surfaces from 3D imaging modalities, such as magnetic resonance imaging (MRI), in which the measurements are inherently noisy.

The problem of preserving features while smoothing away noise has been studied extensively in computer vision. Anisotropic diffusion introduced in [Perona and Malik 1990] has been very successful in dealing with this problem in a wide range of images. Perona & Malik (P&M) proposed to replace Laplacian smoothing, which is equivalent to the heat equation $\partial I/\partial t = \nabla \cdot \nabla I$, with a nonlinear PDE

$$\partial I/\partial t = \nabla \cdot \left[g\left(\| \nabla I \|^2 \right) \nabla I \right], \tag{18}$$

where I is generally the grey-level image. This PDE is the first variation of

$$\int_U G\left(\| \nabla I \|^2 \right) \, dx \, dy, \tag{19}$$

where g in (18), the derivative of G with respect to $\| \nabla I \|^2$, is the edge stopping function, and U is the image domain. P&M suggested using $g(x) = e^{-\|\nabla I\|^2/2\mu}$, where μ is a positive, free parameter that controls the level of contrast of edges that can affect the smoothing process. Notice that $g(\| \nabla I \|)$ approaches 1 for $\| \nabla I \| \ll \mu$ and 0 for $\| \nabla I \| \gg \mu$. Edges are generally associated with large image gradients, and thus diffusion across edges is

stopped while regions that are relatively flat undergo smoothing. Solutions to (18) can actually exhibit an inverse diffusion near edges, and can enhance or sharpen smooth edges that have gradients greater than μ [Perona and Malik 1990].

A great deal of research has focused on modified second-order flows that produce better results than MCF. Using level set models, several authors have proposed smoothing surfaces by weighted combinations of principle curvatures. For instance, Whitaker has proposed a nonlinear reweighting scheme that favors the smaller curvature and preserves cylindrical structures [1994]. Lorigo et al. propose a smoothing by the minimum curvature [2000]. A variety of other combinations have been proposed [Sapiro 2001]. A similar set of curvature-based algorithms have been developed for surface meshes. For instance, Clarenz et al. propose a modified MCF as an anisotropic diffusion of the surface [2000]. They threshold a weighted sum of the principle curvatures to determine the surface locations where edge sharpening is needed. Tangential displacement is added to the standard MCF at these locations for sharpening the edges. Although, this flow produces results that tend to preserve sharp features, it is not a strict generalization of [Perona and Malik 1990] anisotropic diffusion from images to surfaces. Another mesh-based modified MCF is proposed in [Ohtake et al. 2000] where a threshold on the mean curvature is used to stop over-smoothing. Taubin proposes a "linear anisotropic" Laplacian operator for meshes that is based on a separate processing of the normals [2001]. It is essentially a reweighting of the Laplacian. In a different context, anisotropic diffusion as a modified surface area minimization for height functions was proposed in [Desbrun et al. 2000].

These level set and mesh based methods are all modifications of curvature flows, and are therefore all second-order processes. Because they are based on reweightings of curvature, these methods always smooth the surface in one direction or another. They do not exhibit a sharpening of details, which is achieved by the P&M equation (for images) through an inverse diffusion process. Hence, these methods are not satisfactory generalizations of the P&M anisotropic diffusion equation. The generalization of P&M diffusion to surfaces requires a higher-order geometric flow which is achieved from variational principles by choosing the appropriate function of total curvature in (14). For instance,

$$G(\kappa^2) = 2\mu^2\left(1 - e^{-\frac{\kappa^2}{2\mu^2}}\right), \text{ and } g(\kappa^2) = e^{-\frac{\kappa^2}{2\mu^2}}, \tag{20}$$

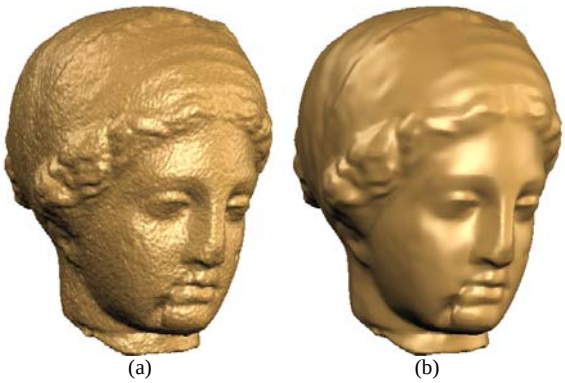
where g is the derivative of G with respect to κ^2. The first variation with respect to the surface normals gives a vector-valued anisotropic diffusion on the level set surface—a straightforward generalization of (18). This flow preserves or enhances areas of high curvature, which we will call *creases*. Creases are the generalization of edges in images to surfaces [Eberly 1996].

4.1 Results

Figure 4(a) illustrates an example of the skin surface, which was extracted, via isosurfacing, from an MRI data set. Notice that the roughness of the skin is noise, an artifact of the measurement process. This model is also quite complex because, despite our best efforts to avoid it, the isosurfaces include many convoluted passages up in the sinuses and around the neck. Isotropic diffusion, shown in Figure 4(b), is marginally effective for denoising the head surface. Notice that the sharp edges around the eyes, nose, lips and ears are lost in this process. The differences between anisotropic diffusion and isotropic diffusion can clearly be observed in Figure 4(c). There is no noticeable difference in the results of the

(a) (b) (c)

Fig. 4. Processing results on the MRI head model: (a) original isosurface, (b) isotropic diffusion, and (c) anisotropic diffusion. The small protrusion under the nose is a physical marker used for registration.



(a) (b)

Fig. 5. (a) Noisy venus head model, and (b) smoothed version after three iterations of anisotropic diffusion.

two processes around the smooth areas of the original model such as the forehead and the cheeks; however, very significant differences exist around the lips and the eyes. The creases in these areas, which have been eliminated by isotropic diffusion, are preserved by the anisotropic process. Note that the free parameter μ in (20) was fixed at 0.1 for all of the results shown in this paper. Unlike, in P&M image diffusion, this parameter does not need to be changed for different surface models. In the context of P&M image diffusion, the units of μ are in gray levels; consequently, the optimal choice of μ is image dependent. However, for surfaces, the units are in curvature, which is data independent. This makes it possible to choose a μ value that gives consistent results over a broad range of surfaces.

The computation time required for one iteration of the main processing loop operating on this model is approximately 15 minutes on a *1.7 Ghz Intel processor* for both isotropic and anisotropic diffusion. The results shown in Figure 4(b) and (c) are both after three iterations which translates to around 45 minutes of processing time. The generality of the proposed approach comes at the cost of significant computation time. However, the method is practical with state-of-the-art computers and is well-poised to benefit from parallel computing architectures, due to its reliance on local, iterative computations.

Another example of denoising by anisotropic diffusion is shown in Figure 5. Noise was added to the original model, which in this case is a $221 \times 221 \times 161$ volume. After three iterations of the main processing loop the noise was successfully removed while preserving

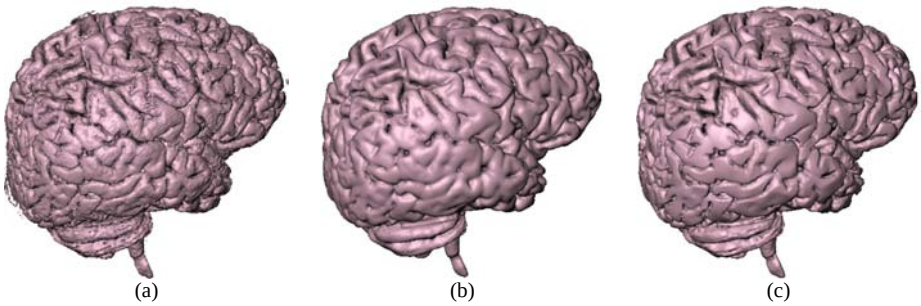


Fig. 6. (a) Original brain isosurface from MRI data set, (b) result of MCF, and (c) after 5 iterations of anisotropic diffusion.

the features of the original model. The quality of these results compares favorably with results from the same model shown in [Clarenz et al. 2000]. Our results demonstrate better preservation of fine, sharp details, such as those around the eyes and in the hair. The computation times per iteration for this example are approximately five minutes compared to 15 minutes per iteration for the example in Figure 4. This is indicative of the relatively high degree of complexity of the MRI based model in the previous example.

Figure 6(a) shows a different isosurface (the cortex) extracted from the same MRI scan as the model in Figure 4. The complexity of this model, i.e. the many tightly nested folds, make it ill-suited for mesh based deformations. Also, the main cortical surface has many detached pieces, an artifact of the segmentation process. As an indication of this complexity, we note that object enclosed by the cortical surface has more than 700 connected components. The approach proposed in this paper can automatically simplify topologically noisy features due to the level set implementation — an important aspect of denoising measured surfaces.

The examples in Figure 7 demonstrates another aspect of the proposed method. Although the original model in Figure 7(a) was constructed as a volume directly from 3D range data [Curless and Levoy 1996], it does not exhibit significant noise. Hence, smoothing is done with the purpose of simplification of the original model rather than denoising in these examples. Running isotropic diffusion for many steps creates a linear scale space where details in the model are progressively eliminated in accordance to their scale; the scales on the skin and the horns have been eliminated in Figure 7(b) and Fig 7(c), respectively. When running the proposed method for anisotropic diffusion, however, surfaces *tend* toward solutions that have piecewise smooth normals with sharp discontinuities in the normal map—analogous to the behavior of the P&M equation for intensity images. Such properties in the normal map correspond to surfaces consisting of planar patches and smooth patches bounded by sharp creases. Thus, the proposed method generates a feature preserving scale space, very much like that of P&M for images. These results, which are shown in Figure 7(d) and (e), support our proposition that processing the normals of a surface is the natural generalization of image processing. The non-linear progression of elimination of details from the smallest scale to the largest also suggests applications of this method to surface compression and multi-resolution modeling.

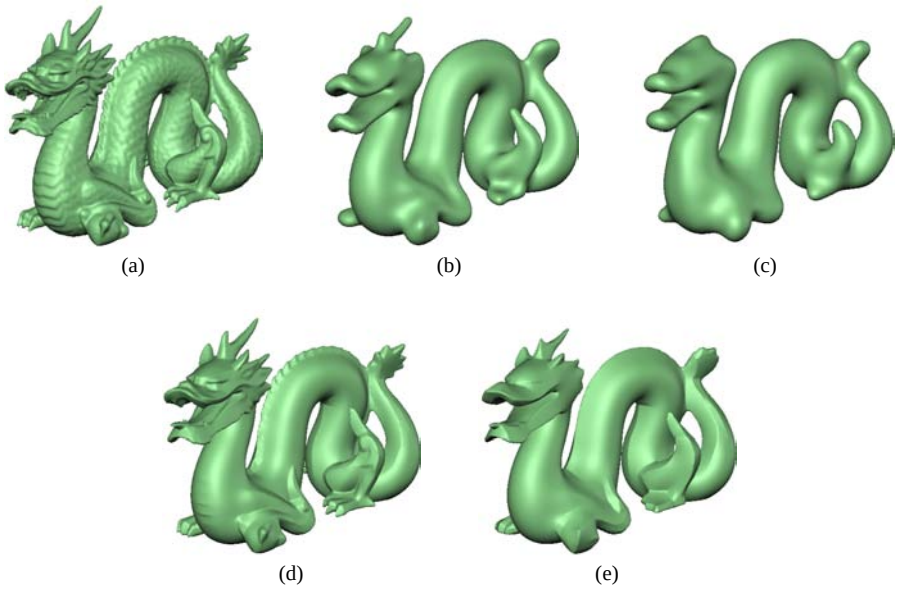


Fig. 7. (a) Original model. Isotropic diffusion: (b) after 10 iterations, and (c) after 20 iterations. Anisotropic diffusion: (a) after 10 iterations, and (b) after 20 iterations.

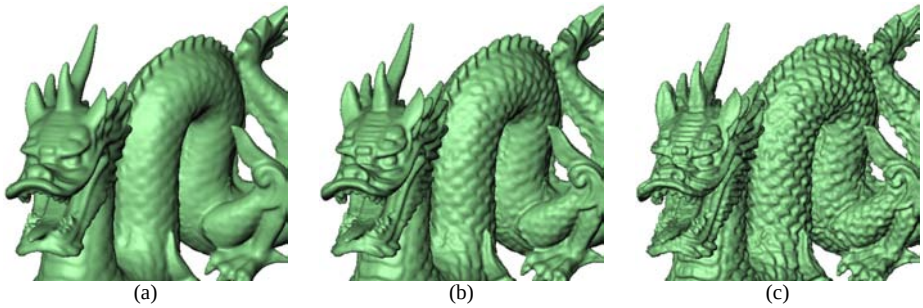


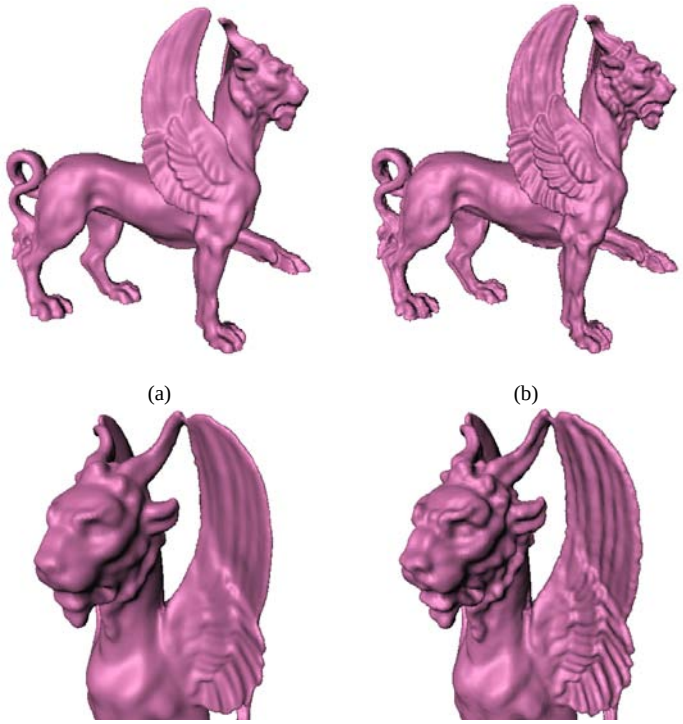
Fig. 8. (a) Original model, (b) after 1, and (c) 2 iterations of high-boost filtering.

5. HIGH-BOOST FILTERING

The surface processing framework introduced in Section 3 is flexible and allows for the implementation of even more general image processing methods. We demonstrate this by describing how to generalize image enhancement by high-boost filtering of surfaces.

A high-boost filter has a frequency transform that amplifies high frequency components. In image processing, this can be achieved by *unsharp masking* [Gonzalez and Woods 1992]. Let the low-pass filtered version of an image I be $\tilde{I}$. The high-frequency components are $I_{hf} = I - \tilde{I}$. The high-boost output is the sum of the input image and some fraction of its high-frequency components:

$$I_{\text{out}} = I + \alpha I_{hf} = (1+\alpha)I - \alpha\tilde{I}, \tag{21}$$



(a) (b) (c) (d)

Fig. 9. High-boost filtering: (a) original model, (b) after filtering, (c) close-up of original, and (d) filtered model.

where α is a positive constant that controls the amount of high-boost filtering.

This same algorithm applies to surface normals by a simple modification to the flow chart in Figure 3. Recall that the $d\mathscr{G}/d\mathbf{N}$ loop produces $\mathbf{N}^{n+1}$. Define a new set of normal vectors by

$$\mathbf{N}' = \frac{(1+\alpha)\mathbf{N}^n - \alpha\mathbf{N}^{n+1}}{||(1+\alpha)\mathbf{N}^n - \alpha\mathbf{N}^{n+1}||}. \tag{22}$$

This new normal map is then input to the $d\mathscr{D}/d\phi$ refitting loop. The effect of (22) is to extrapolate from the previous set of normals in the direction opposite to the set of normals obtained by isotropic diffusion. Recall that isotropic diffusion will smooth areas with high curvature and not significantly affect already smooth areas. Processing the loop with the modification of (22) will have the effect of increasing the curvature in areas of high curvature, while leaving smooth areas relatively unchanged. Thus, we are able to obtain high quality surface enhancement on fairly complex surfaces of arbitrary topology, as can be observed in Figs. 8 and 9. The scales on the skin and the ridge back are enhanced. Also, note that different amounts of enhancement can be achieved by controlling the number of iterations of the main loop. The degree of low-pass filtering used to obtain $\mathbf{N}^{n+1}$ controls the size of the features that are enhanced. Figure 9 shows another example of high-boost filtering; notice the enhancement of features particularly on the wings.

6. DISCUSSION

The *natural* generalization of image processing to surfaces is via the normals. The lowest-order differential invariant of images is the gradient magnitude, and minimizing a quadratic penalty of this quantity produces the diffusion equation, which gives rise to Gaussian blurring. The lowest-order differential invariants of surface shape are the principal curvatures. Likewise, the curvature of a 3D surface is a function of the gradient of the surface normal as shown in Section 3.2. In this light, total curvature, which is the Euclidean norm of the Jacobian of the vector field of surface normals, is the natural generalization of the squared-gradient-magnitude smoothness penalty for images. *Thus, for surfaces, first variation of the isotropic total curvature penalty, rather than MCF, is the equivalent of Gaussian blurring.*

Variational processes on the surface curvature have corresponding variational formulations on the surface normals. The generalization of image-processing to surface normals, however, requires that we process the normals using a metric on the surface manifold, rather than a simple, flat metric, as we do with images. By processing the normals separately, we can solve a pair of coupled second-order equations instead of a fourth-order equation. Typically, we allow one equation (the surface) to lag the other, but as the lag gets very small, it should not matter. In this framework, the diffusion of the surface normals (and corresponding motions of the surface) is equivalent to the particular fourth-order flow that minimizes the surface total curvature penalty function.

The method generalizes because we can do virtually anything we wish with the normal map. A generalization of anisotropic diffusion to a constrained, vector-valued function, defined on a manifold, gives us a smoothing process that preserves creases. If we want to enhance the surface, we can enhance the normals and refit the surface.

We solve these equations using implicit surfaces, representing the implicit function on a discrete grid, modeling the deformation with the method of level sets. This level set implementation allows us to separate the shape of the model from the processing mechanism. Because of the implementation, the method applies equally well to any surface that can be represented in a volume. Consequently, our results show a level of surface complexity that goes beyond that of previous methods.

Future work will study the usefulness of other interesting image processing techniques such as *total variation* [Rudin et al. 1992; Burchard 2002] and local contrast enhancement. To date, we have dealt with post processing surfaces, but future work will combine this method with segmentation and reconstruction techniques. The current shortcoming of this method is the computation time, which is significant. However, the process lends itself to parallelism, and the advent of cheap, specialized, vector-processing hardware promises significantly faster implementations.

ACKNOWLEDGMENTS

This work is supported by the Office of Naval Research under grant #N00014-01-10033 and the National Science Foundation under grant #CCR0092065.

REFERENCES

ADALSTEINSON, D. AND SETHIAN, J. A. 1995. A fast level set method for propogating interfaces. *J. Computational Physics*, 269–277.

ALEXA, M. 2002. Wiener filtering of meshes. In *Shape Modeling International.* 51–57.

BALLESTER, C., BERTALMIO, M., CASELLES, V., SAPIRO, G., AND VERDERA, J. 2001. Filling-in by joint interpolation of vector fields and gray levels. *IEEE Trans. on Image Processing 10*, 8 (August), 1200–1211.

BERTALMIO, M., CHENG, L.-T., OSHER, S., AND SAPIRO, G. 2001. Variational problems and partial differential equations on implicit surfaces. *J. Computational Physics 174*, 759–780.

BURCHARD, P. 2002. Total variation geometry 1: Concepts and motivation. Tech. rep., UCLA CAM-02-01. January.

CHOPP, D. L. AND SETHIAN, J. A. 1999. Motion by intrinsic laplacian of curvature. *Interfaces and Free Boundaries 1*, 1–18.

CLARENZ, U., DIEWALD, U., AND RUMPF, M. 2000. Anisotropic geometric diffusion in surface processing. In *Proceedings of IEEE Visualization.* 397–405.

CURLESS, B. AND LEVOY, M. 1996. A volumetric method for building complex models from range images. In *Proceedings of SIGGRAPH'96*.

DESBRUN, M., MEYER, M., SCHRÖDER, M., AND BARR, A. 1999. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of SIGGRAPH'99*. 317–324.

DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. H. 2000. Anisotropic feature-preserving denoising of height fields and bivariate data. In *Graphics Interface.*

DOCARMO, M. P. 1976. *Differential Geometry of Curves and Surfaces.* Prentice Hall.

EBERLY, D. 1996. *Ridges in Image and Data Analysis.* Kluwer Academic Publishers.

GONZALEZ, R. C. AND WOODS, R. E. 1992. *Digital Image Processing.* Addison-Wesley Publishing Company.

GUSKOV, I., SWELDENS, W., AND SCHRÖDER, P. 1999. Multiresolution signal processing for meshes. In *Proceedings of SIGGRAPH'99*. 325–334.

HALSTEAD, M., KASS, M., AND DEROSE, T. 1993. Efficient, fair interpolation using catmull-clark surface. In *Proceedings of SIGGRAPH'93*. 35–44.

KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P. 1998. Interactive multi-resolution modeling on arbitrary meshes. In *Proceedings of SIGGRAPH'98*. 105–114.

LORENSON, W. AND CLINE, H. 1987. Marching cubes: A high resolution 3d surface reconstruction algorithm. In *Proceedings of SIGGRAPH'87*. 163–169.

LORIGO, L., FAUGERAS, O., GRIMSON, E., KERIVEN, R., KIKINIS, R., NABAVI, A., AND WESTIN, C.-F. 2000. Co-dimension 2 geodesic active contours for the segmentation of tubular strucures. In *Proceedings of Computer Vision and Pattern Recognition.*

MALLADI, R., SETHIAN, J. A., AND VEMURI, B. C. 1995. Shape modeling with front propagation: A level set approach. *IEEE Trans. on Pattern Analysis and Machine Intelligence 17*, 2, 158–175.

MORETON, H. P. AND SÉQUIN, C. H. 1992. Functional optimization for fair surface design. In *Proceedings of SIGGRAPH'92*. 167–176.

OHTAKE, Y., BELYAEV, A. G., AND BOGAEVSKI, I. A. 2000. Polyhedral surface smoothing with simultaneous mesh regularization. In *Geometric Modeling and Processing.*

OSHER, S. AND SETHIAN, J. A. 1988. Fronts propagating with curvature dependent speed: Algorithms based on hamilton-jacobi formulation. *J. Computational Physics 79*, 12–49.

PENG, D., MERRIMAN, B., OSHER, S., ZHAO, H.-K., AND KANG, M. 1999. A pde based fast local level set method. *J. Computational Physics 155*, 410–438.

PERONA, P. 1998. Orientation diffusion. *IEEE Trans. on Image Processing 7*, 3, 457–467.

PERONA, P. AND MALIK, J. 1990. Scale space and edge detection using anisotropic diffusion. *IEEE Trans. on Pattern Analysis and Machine Intelligence 12*, 7 (July), 629–639.

POLDEN, A. 1997. Compact surfaces of least total curvature. Tech. rep., University of Tubingen, Germany.

RUDIN, L., OSHER, S., AND FATEMI, C. 1992. Nonlinear total variation based noise removal algorithms. *Physica D 60*, 259–268.

SAPIRO, G. 2001. *Geometric Partial Differential Equations and Image Analysis.* Cambridge University Press, Chapter Geometric Curve and Surface Evolution.

SCHNEIDER, R. AND KOBBELT, L. 2000. Generating fair meshes with g^1 boundary conditions. In *Geometric Modeling and Processing Proceedings.* 251–261.

SCHNEIDER, R. AND KOBBELT, L. 2001. Geometric fairing of irregular meshes for free-form surface design. *Computer Aided Geometric Design 18*, 359–379.

SETHIAN, J. A. 1999. *Level Set Methods and Fast Marching Methods Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science.* Cambridge University Press.

TANG, B., SAPIRO, G., AND CASELLES, V. 2000. Diffusion of general data on non-flat manifolds via harmonic maps theory: The direction diffusion case. *International Journal of Computer Vision 36,* 2, 149–161.

TASDIZEN, T., WHITAKER, R., BURCHARD, P., AND OSHER, S. 2002. Geometric surface smoothing via anisotropic diffusion of normals. In *Proceedings of IEEE Visualization.* 125–132.

TAUBIN, G. 1995. A signal processing approach to fair surface design. In *Proceedings of SIGGRAPH'95*. 351–358.

TAUBIN, G. 2001. Linear anisotropic mesh filtering. Tech. Rep. RC22213, IBM Research Division. October.

TAUBIN, G., ZHANG, T., AND GOLUB, G. 1996. Optimal surface smoothing as filter design. In *European Conf. on Computer Vision.* Vol. 1. 283–292.

WELCH, W. AND WITKIN, A. 1992. Variational surface modeling. In *Proceedings of SIGGRAPH'92*. 157–166.

WELCH, W. AND WITKIN, A. 1994. Free-form shape design using triangulated surfaces. In *Proceedings of SIGGRAPH'94*. 247–256.

WHITAKER, R. T. 1994. Volumetric deformable models: Active blobs. In *Visualization In Biomedical Computing,* R. A. Robb, Ed. SPIE, 122–134.

WHITAKER, R. T. 1998. A level-set approach to 3D reconstruction from range data. *Int. J. Computer Vision 29,* 3, 203–231.

A. MATHEMATICAL FOUNDATION

In this section, we will derive the equivalence of the proposed algorithm, in the limit, to minimizing the original energy function $\mathscr{G}_\phi$ defined in (10). Let us rewrite this energy function by observing that the principal curvatures are functions of the derivatives of ϕ

$$\mathscr{G}_\phi = \int_U G(\phi) \parallel \nabla\phi \parallel \, d\mathbf{x}. \tag{23}$$

Let $d\phi : \Re^3 \rightarrow \Re$ be a volume of incremental changes applied to $\phi : \Re^3 \rightarrow \Re$. The change to $\mathscr{G}$ induced by $d\phi$ can be expressed as the volume integral of the total derivative of the penalty function, $dG(\phi) \parallel \nabla\phi \parallel$, which is the product of $d\phi$ and the variation of the penalty function with respect to ϕ

$$d\mathscr{G}_\phi = \int_U \frac{d\left(G \parallel \nabla\phi \parallel\right)}{d\phi} \, d\phi \, d\mathbf{x}. \tag{24}$$

Applying the product rule to $\frac{d(G\|\nabla\phi\|)}{d\phi}$, we obtain

$$d\mathscr{G}_\phi = \underbrace{\int_U \frac{dG}{d\phi} \parallel \nabla\phi \parallel \, d\phi \, d\mathbf{x}}_{d\mathscr{G}_{\phi,1}} + \underbrace{\int_U G \frac{d \parallel \nabla\phi \parallel}{d\phi} \, d\phi \, d\mathbf{x}}_{d\mathscr{G}_{\phi,2}}. \tag{25}$$

The total derivative $dG(\phi) \parallel \nabla\phi \parallel$ can be written in terms of the surface normals by using the equality

$$\frac{dG}{d\phi} \, d\phi = \frac{dG}{d\mathbf{N}} \cdot d\mathbf{N}, \tag{26}$$

given that the normal map is a function of ϕ. Then, the first term in (25) can be written as

$$d\mathscr{G}_{\phi,1} = \int_U \frac{dG}{d\mathbf{N}} \cdot d\mathbf{N} \parallel \nabla\phi \parallel \, d\mathbf{x}. \tag{27}$$

To simplify (27), we derive $d\mathbf{N}$ as a function of $\mathbf{N}$ and $d\nabla\phi$

$$d\mathbf{N} = d\frac{\nabla\phi}{\| \nabla\phi \|} = d\frac{\nabla\phi}{(\nabla\phi \cdot \nabla\phi)^{1/2}} \tag{28}$$

$$= \frac{d\nabla\phi}{\| \nabla\phi \|} - \frac{(d\nabla\phi \cdot \mathbf{N})\,\mathbf{N}}{\| \nabla\phi \|}. \tag{29}$$

Equation (29) follows from (28) by using the chain rule for the differentiation, and substituting $\mathbf{N}$ back for $\nabla\phi / \| \nabla\phi \|$. Substituting (29) for $d\mathbf{N}$ in (27), we get

$$d\mathscr{G}_{\phi,1} = \int_U \left(\frac{dG}{d\mathbf{N}} \cdot d\nabla\phi - (d\nabla\phi \cdot \mathbf{N}) \ \frac{dG}{d\mathbf{N}} \cdot \mathbf{N} \right) \ d\mathbf{x}. \tag{30}$$

We are only interested in processes that maintain the unit length constraint of the normal map; therefore, $dG/d\mathbf{N} \cdot \mathbf{N} = 0$, and (30) is reduced to

$$d\mathscr{G}_{\phi,1} = \int_U \frac{dG}{d\mathbf{N}} \cdot \nabla d\phi \ d\mathbf{x}, \tag{31}$$

where we also use the linearity of differentiation to make the substitution $d\nabla\phi = \nabla d\phi$. We treat this energy minimization as an *adiabatic* problem, which means that energy flow across the boundary of U is zero. Hence, using *Neumann* boundary conditions for U and integration by parts, we obtain

$$d\mathscr{G}_{\phi,1} = \int_U \nabla \cdot \frac{dG}{d\mathbf{N}} \, d\phi \ d\mathbf{x}. \tag{32}$$

We now examine the second term in (25), $d\mathscr{G}_{\phi,2}$. As in Section 3.2, we treat G as a function of $\mathbf{N}$; therefore, due to the decoupling between $\mathbf{N}$ and ϕ, G can be considered independent of ϕ. Using this assumption, we can rewrite $d\mathscr{G}_{\phi,2}$ as

$$d\mathscr{G}_{\phi,2} = \int_U \frac{dG^{\mathbf{N}} \| \nabla\phi \|}{d\phi} \, d\phi \ d\mathbf{x}, \tag{33}$$

where the superscript on $G^{\mathbf{N}}$ is to mean that G is fixed with respect to ϕ. Taking the first variation of $dG^{\mathbf{N}} \| \nabla\phi \|$ yields

$$d\mathscr{G}_{\phi,2} = \int_U \nabla \cdot \left(G^{\mathbf{N}} \frac{\nabla\phi}{\| \nabla\phi \|} \right) \ d\phi \ d\mathbf{x}, \tag{34}$$

where we use the fact that $d \| \nabla\phi \| / d\nabla\phi = \nabla\phi / \| \nabla\phi \|$. Finally, combining equations (24), (32), and (34), we can derive the desired relationship between the variations with respect to ϕ and $\mathbf{N}$

$$\frac{d\,(G \| \nabla\phi \|)}{d\phi} = \nabla \cdot \left(\frac{dG}{d\mathbf{N}} + G^{\mathbf{N}} \frac{\nabla\phi}{\| \nabla\phi \|} \right). \tag{35}$$

Let us now consider the flow achieved by processing (15) and (17) back to back in one iteration of the main loop in Figure 3 again. At the beginning of iteration n, the normals are computed from ϕ^n. If we evolve the normals for one step according to (15), instead of processing them multiple iterations, the new normals are

$$\mathbf{N}^{n+1} = \mathbf{N}^n - \frac{dG}{d\mathbf{N}}, \tag{36}$$

where we write $\frac{dG}{d\mathbf{N}}$ instead of $\frac{d\mathscr{G}}{d\mathbf{N}}$ because we are referring to the update for $\mathbf{N}$ at a specific point in space. If we immediately apply (17) to fit ϕ to this new normal map, we get

$$\frac{dD}{d\phi} = H^{\phi^n} - \nabla \cdot \left(\mathbf{N}^n - \frac{dG}{d\mathbf{N}} \right), \tag{37}$$

where D is the local function defined in (16). Because $\mathbf{N}^n$ is derived directly from ϕ^n, we have $\nabla \cdot \mathbf{N} = H^{\phi^n}$, which gives the rule in our algorithm to make up this infinitesimal lag:

$$\frac{dD}{d\phi} = \nabla \cdot \frac{dG}{d\mathbf{N}}. \tag{38}$$

Comparing with (35), we find the rule to descend on the energy as a function of ϕ

$$\frac{d\left(G \parallel \nabla\phi \parallel\right)}{d\phi} = \frac{dD}{d\phi} + \nabla \cdot \left(G^{\mathbf{N}} \frac{\nabla\phi}{\parallel \nabla\phi \parallel} \right). \tag{39}$$

This establishes the mathematical foundation of our method. However, in our experiments, we have found that the contribution of the second term is very small and it does not change the results qualitatively. Therefore, we drop it for the sake of computational efficiency, and implement only $\frac{dD}{d\phi}$ as described in Section 3.

B. NUMERICAL IMPLEMENTATION

By embedding surface models in volumes, we have converted equations that describe the movement of surface points to nonlinear PDEs defined on a volume. The next step is to discretize these PDEs in space and time. In this paper, the embedding function ϕ is defined on the volume domain U and time. The PDEs are solved using a discrete sampling with forward differences along the time axis.

For brevity, we will discuss the numerical implementation in 2D— the extension to 3D is straightforward. The function $\phi : U \mapsto \Re$ has a discrete sampling $\phi[p,q]$, where $[p,q]$ is a grid location and $\phi[p,q] = \phi(x_p, y_q)$. We will refer to a specific time instance of this function with superscripts, i.e. $\phi^n[p,q] = \phi(x_p, y_q, t_n)$. In our calculations, we need three different approximations to first-order derivatives: forward, backward, and central differences. We denote the type of discrete difference using superscripts on a difference operator, i.e., $\delta^{(+)}$ for forward differences, $\delta^{(-)}$ for backward differences, and δ for central differences. For instance, the differences in the x direction on a discrete grid with unit spacing are

$$\begin{aligned} \delta_x^{(+)}\phi[p,q] &\triangleq \phi[p+1,q] - \phi[p,q], \\ \delta_x^{(-)}\phi[p,q] &\triangleq \phi[p,q] - \phi[p-1,q], \text{and} \\ \delta_x\phi[p,q] &\triangleq \frac{\phi[p+1,q] - \phi[p-1,q]}{2}. \end{aligned} \tag{40}$$

The application of these difference operators to vector-valued functions denotes componentwise differentiation.

In describing the numerical implementation, we will refer to the flow chart in Figure 3 for one iteration of the main loop. Hence, the first step in our numerical implementation is the calculation of the surface normal vectors from ϕ^n. Recall that the surface is a level set of ϕ^n as defined in (7). Hence, the surface normal vectors can be computed as the unit

vector in the direction of the gradient of ϕ^n. The gradient of ϕ^n is computed with central differences as

$$\nabla\phi^n \approx \begin{pmatrix} \delta_x\phi^n \\ \delta_y\phi^n \end{pmatrix}; \tag{41}$$

and the normal vectors are initialized as

$$\mathbf{N}^{u=0} = \nabla\phi^n / ||\nabla\phi^n|| . \tag{42}$$

Because ϕ^n is fixed and allowed to lag behind the evolution of $\mathbf{N}$, the time steps in the evolution of $\mathbf{N}$ are denoted with a different superscript, u. For this evolution, $\partial\mathbf{N}/\partial t = -d\mathscr{G}/d\mathbf{N}$, which is derived in (15).

We now describe how to numerically compute $d\mathscr{G}/d\mathbf{N}$. This computation is implemented with smallest support area operators, which use the smallest possible neighborhood of voxels to compute the required output. The Laplacian of a function can be computed in two steps by first applying the gradient operator and then the divergence operator. In 2D, the gradient of the normals produces a 2×2 matrix, which we call the *flux* matrix. Next, the divergence operator collapses the flux matrix to a 2×1 vector. The "columns" of the flux matrix are computed independently as

$$\overset{x+}{\mathbf{M}}{}^u \approx \delta_x^{(+)}\mathbf{N}^u - \left(\delta_x^{(+)}\phi^n\right)\overset{x+}{\mathbf{C}}{}^u, \tag{43}$$

$$\overset{y+}{\mathbf{M}}{}^u \approx \delta_y^{(+)}\mathbf{N}^u - \left(\delta_y^{(+)}\phi^n\right)\overset{y+}{\mathbf{C}}{}^u \tag{44}$$

where the time index n remains fixed as we increment u. The positions of $\mathbf{M}_u$, which is computed with forward differences, are staggered off the grid by half a pixel, see Figure 10. For instance, $\overset{x+}{\mathbf{M}}{}^u$ uses information from positions $[p+1,q]$ and $[p,q]$; hence, it exists at $[p+1/2,q]$. We use the following notation: for some function α, $\overset{x+}{\alpha}$, and $\overset{y+}{\alpha}$ will denote the function computed at $[p+1/2,q]$ and $[p,q+1/2]$, respectively.

To compute the intrinsic derivatives of $\mathbf{N}^u$ on the level sets of ϕ^n, $(\delta_x^{(+)}\phi^n)\,\overset{x+}{\mathbf{C}}{}^u$, and $(\delta_y^{(+)}\phi^n)\,\overset{y+}{\mathbf{C}}{}^u$ are subtracted from the regular derivatives of $\mathbf{N}^u$. The variables $\overset{x+}{\mathbf{C}}{}^u$ and $\overset{y+}{\mathbf{C}}{}^u$ are computed as follows

$$\overset{x+}{\mathbf{C}}{}^u = \overset{x+}{\nabla}\mathbf{N}^u \cdot \overset{x+}{\nabla}\phi^n \,/\, \|\overset{x+}{\nabla}\phi^n\|^2, \tag{45}$$

$$\overset{y+}{\mathbf{C}}{}^u = \overset{y+}{\nabla}\mathbf{N}^u \cdot \overset{y+}{\nabla}\phi^n \,/\, \|\overset{y+}{\nabla}\phi^n\|^2, \tag{46}$$

where the matrix $\nabla\mathbf{N}$ is as defined in (12). In (45), the dot product between the matrix $\overset{x+}{\nabla}\mathbf{N}^u$ and the vector $\overset{x+}{\nabla}\phi^n \,/\, \|\overset{x+}{\nabla}\phi^n\|^2$ denotes the vector whose components are the dot products of the rows of $\overset{x+}{\nabla}\mathbf{N}^u$ and the vector $\overset{x+}{\nabla}\phi^n \,/\, \|\overset{x+}{\nabla}\phi^n\|^2$. The same applies to (46). The variables (45) and (46) must be computed at the same locations as $\overset{x+}{\mathbf{M}}{}^u$ and $\overset{y+}{\mathbf{M}}{}^u$, respectively. These computations are done with the smallest support area operators, using the symmetric 2×3 grid of samples around each staggered point. For instance, the staggered gradients of ϕ,

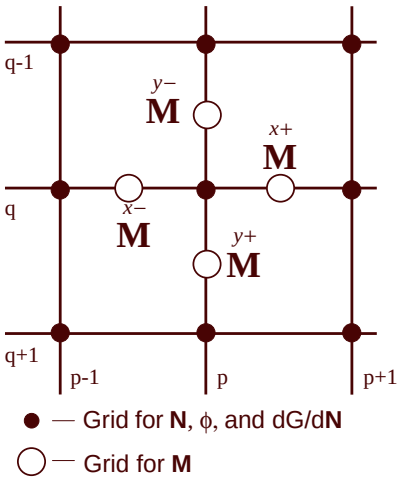


Fig. 10. Computational grid.

which are needed for the evaluating (45) and (46), are computed as

$$\begin{aligned}
\overset{x+}{\nabla}\phi^n = \nabla\phi^n[p+\frac{1}{2},q] &\approx \begin{pmatrix} \delta_x^{(+)}\phi[p,q] \\ \frac{1}{2}\left(\delta_y\phi[p,q]+\delta_y\phi[p+1,q]\right) \end{pmatrix}, \\
\overset{y+}{\nabla}\phi^n = \nabla\phi^n[p,q+\frac{1}{2}] &\approx \begin{pmatrix} \frac{1}{2}\left(\delta_x\phi[p,q]+\delta_x\phi[p,q+1]\right) \\ \delta_y^{(+)}\phi[p,q] \end{pmatrix}
\end{aligned} \tag{47}$$

The staggered gradient matrices of the normals, $\overset{x+}{\nabla}\mathbf{N}^u$ and $\overset{y+}{\nabla}\mathbf{N}^u$, which are also needed for evaluating (45) and (46), are computed with the same stencil.

After the computation of the flux, backwards differences are used to compute the divergence operation in (15). For isotropic diffusion,

$$\Delta^u = \delta_x^{(-)}\,\overset{x+}{\mathbf{M}}{}^u + \delta_y^{(-)}\,\overset{y+}{\mathbf{M}}{}^u, \tag{48}$$

and from (15)

$$-\left[\frac{d\mathscr{G}}{d\mathbf{N}}\right]^u \approx \left(\mathbf{I}-\mathbf{N}^u\otimes\mathbf{N}^u\right)\Delta^u = \Delta^u - \left(\Delta^u\cdot\mathbf{N}^u\right)\mathbf{N}^u. \tag{49}$$

The results of the backwards differencing are defined at the original ϕ grid location $[p,q]$ because they undo the forward staggering in the flux locations. Therefore, both components of Δ and thus $d\mathscr{G}/d\mathbf{N}$ are located on the original grid for ϕ.

To evaluate (15) for anisotropic diffusion, we also need to compute $g(\kappa^2)$ at the precise locations where the flux (43) and (44) are located. Hence, we compute the total intrinsic curvature of the normals

$$\begin{aligned}
\overset{x+}{\kappa}{}^2 &= \|\overset{x+}{\nabla}\mathbf{N}^u\|^2 - \overset{x+}{\mathbf{C}}{}^u\cdot\overset{x+}{\mathbf{C}}{}^u, \\
\overset{y+}{\kappa}{}^2 &= \|\overset{y+}{\nabla}\mathbf{N}^u\|^2 - \overset{y+}{\mathbf{C}}{}^u\cdot\overset{y+}{\mathbf{C}}{}^u,
\end{aligned} \tag{50}$$

where $\|\cdot\|^2$ is the Euclidean norm, the sum of the squares of all elements of the matrix.

Then, the divergence for anisotropic diffusion is computed as

$$\Delta^u = \delta_x^{(-)} \left[g(\overset{x+}{\kappa^2}) \overset{x+}{\mathbf{M}^u} \right] + \delta_y^{(-)} \left[g(\overset{y+}{\kappa^2}) \overset{y+}{\mathbf{M}^u} \right], \tag{51}$$

and the tangential projection is applied to this vector as in (49).

Starting with the initialization in (42) for $u = 0$, we iterate

$$\mathbf{N}^{u+1} = \mathbf{N}^u - \left[\frac{d\mathscr{G}}{d\mathbf{N}} \right]^u \tag{52}$$

for a fixed number of steps, 25 iterations for the examples in this paper. In other words, we do not aim at minimizing the energy given in (14) in the $d\mathscr{G}/d\mathbf{N}$ loop of Figure 3; we only reduce it. The minimization of total mean curvature as a function of ϕ is achieved by iterating the main loop in Figure 3.

Once the evolution of $\mathbf{N}$ is concluded, ϕ is refitted to the new normal vectors according to (17). We denote the evolved normals by $\mathbf{N}^{n+1}$. To solve (17) we must calculate H^ϕ and $H^{\mathbf{N}^{n+1}}$, which is the induced mean curvature of the normal map; in other words, it is the curvature of the hypothetical target surface that fits the normal map. Curvature from a field of normals is given by

$$H^{\mathbf{N}^{n+1}} \approx \delta_x \mathbf{N}_{(x)}^{n+1} + \delta_y \mathbf{N}_{(y)}^{n+1}, \tag{53}$$

where we have used central differences on the components of the normal vectors that are denoted by the subscripts (x) and (y). The quantity $H^{\mathbf{N}^{n+1}}$ needs to be computed once at initialization as the normal vectors remain fixed during the refitting phase. Let v be the time step index in the $d\mathscr{D}/d\phi$ loop. H^{ϕ^v} is the mean curvature of the moving level set surface at time step v and is calculated from ϕ with the smallest area of support

$$H^{\phi^v} \approx \delta_x^{(-)} \frac{\delta_x^{(+)} \phi^v}{\overset{x+}{\|\nabla \phi^v\|}} + \delta_y^{(-)} \frac{\delta_y^{(+)} \phi^v}{\overset{y+}{\|\nabla \phi^v\|}} \tag{54}$$

where the gradients in the denominators are staggered to match the locations of the forward differences in the numerator. The staggered gradients of ϕ in the denominator are calculated using the 2×3 neighborhood as in (47).

The PDE in (17) is solved with a finite forward differences, but with the upwind scheme for the gradient magnitude [Osher and Sethian 1988], to avoid overshooting and maintain stability. The up-wind method computes a one-sided derivative that looks in the up-wind direction of the moving wave front, and thereby avoids overshooting. Moreover, because we are interested in only a single level set of ϕ, solving (17) over all of U is not necessary. Different level sets evolve independently, and we can compute the evolution of ϕ only in a narrow band around the level set of interest and re-initialize this band as necessary [Adalsteinson and Sethian 1995; Peng et al. 1999]. See [Sethian 1999] for more details on numerical schemes and efficient solutions for level set methods.

Using the upwind scheme and narrow band methods, ϕ^{v+1} is computed from ϕ^v according to (17) using the curvatures computed in (53) and (54). This loop is iterated until the energy in (16) ceases to decrease; let v_{final} denote the final iteration of this loop. Then we set ϕ for the next iteration of the main loop (see Figure 3) as $\phi^{n+1} = \phi^{v_{final}}$ and repeat the entire procedure. The number of iterations of the main loop is a free parameter that generally determines the extent of processing.

A Streaming Narrow-Band Algorithm: Interactive Computation and Visualization of Level Sets

Aaron E. Lefohn, Joe M. Kniss, Charles D. Hansen, Ross T. Whitaker

(Invited Paper)

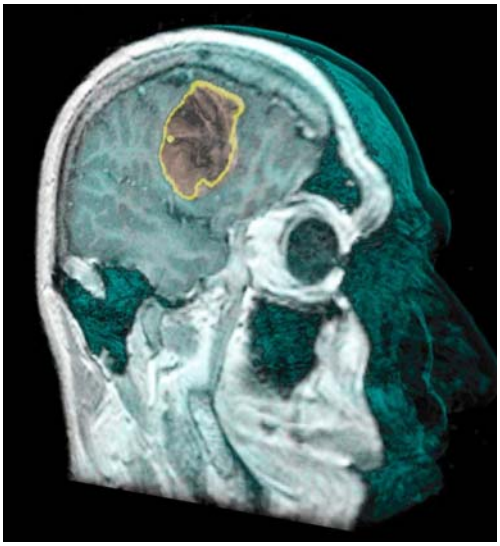


Fig. 1. Interactive level-set segmentation of a brain tumor from a $256 \times 256 \times 198$ MRI with volume rendering to give context to the segmented surface. A clipping plane shows the user the source data, the volume rendering, and the segmentation simultaneously. The segmentation and volume rendering parameters are set by the user probing data values on the clipping plane.

***Abstract*— Deformable isosurfaces, implemented with level-set methods, have demonstrated a great potential in visualization and computer graphics for applications such as segmentation, surface processing, and physically-based modeling. Their usefulness has been limited, however, by their high computational cost and reliance on significant parameter tuning. This paper presents a solution to these challenges by describing graphics processor (GPU) based algorithms for solving and visualizing level-set solutions at interactive rates. The proposed solution is based on a new, streaming implementation of the narrow-band algorithm. The new algorithm packs the level-set isosurface data into 2D texture memory via a multi-dimensional virtual memory system. As the level-set moves, this texture-based representation is dynamically updated via a novel GPU-to-CPU message passing scheme. By integrating the level-set solver with a real-time volume renderer, a user can visualize and intuitively steer the level-set surface as it evolves. We demonstrate the capabilities of this technology for interactive volume segmentation and visualization.**

***Index Terms*— Deformable Models, Image Segmentation, Volume Visualization, GPU, Level Sets, Streaming Computation, Virtual Memory**

All authors are associated with the Scientific Computing and Imaging Institute at the University of Utah.
e-mail:{lefohn|jmk|hansen|whitaker}@sci.utah.edu

I. Introduction

Level-set methods [1] rely on partial differential equations (PDEs) to model deforming isosurfaces. These methods have applications in a wide range of fields such as visualization, scientific computing, computer graphics, and computer vision [2], [3]. Applications in visualization include volume segmentation [4], surface processing [5], and surface reconstruction [6].

The use of level sets in visualization can be problematic. Level sets are relatively slow to compute and they typically introduce several free parameters that control the surface deformation and the quality of the results. Setting these free parameters can be difficult because, in many scenarios, a user must wait minutes or hours to observe the results of a parameter change. Although efforts have been made to take advantage of the sparse nature of the computation, the most highly optimized solvers are still far from interactive. This paper proposes a solution to the above problems by mapping the level-set PDE solver to a commodity graphics processor.

While the proposed technology has a wide range of uses within visualization and elsewhere, this paper focuses on a particular application: the analysis and visualization of volume data. By accelerating the PDE solver to interactive rates and coupling it to a real-time volume renderer, it is possible to visualize and steer the computation of a level-set surface as it moves toward interesting regions within a volume. The volume renderer provides visual context for the evolving level set due to the global nature of the transfer function's opacity and color assignment. Also, the results of a level-set segmentation can specify a region-of-interest for the volume renderer [7].

The main contributions of this paper are:

- An integrated system demonstrating that level-set computations can be intuitively controlled by coupling a real-time volume renderer with an interactive solver
- A GPU-based 3D level-set solver that is approximately 15 times faster than previous optimized solutions
- A multi-dimensional virtual memory scheme for GPU texture memory that supports computation on time-dependent, sparse data
- Real-time volume rendering directly from a packed, 2D texture format. The technique also enables volume rendering from a data set represented as a *single* set of 2D slices.
- A message passing scheme between the GPU and CPU that uses automatic mipmap generation to create compact, encoded messages
- Efficient computation of a volumetric distance transform on the GPU

II. Background and Related Work

A. Level Sets

This paper describes a new solver for an implicit representation of deformable surface models called the method of *level sets* [1]. The use of level sets has been widely documented in the visualization literature, and several works give comprehensive reviews of the method and the associated numerical techniques [2], [3]. Here we merely review the notation and describe the particular formulation that is relevant to this paper.

An implicit model represents a surface as the set of points $S = \{\bar{x} | \phi(\bar{x}) = 0\}$, where $\phi : \mathbb{R}^3 \mapsto \mathbb{R}$. Level-set methods relate the motion of that surface to a PDE on the volume, i.e.

$$\partial\phi/\partial t = -\nabla\phi \cdot \bar{v}, \tag{1}$$

where $\bar{v}$ describes the motion of the surface. Note that $\bar{v}$ can vary in both space and time. Within this framework one can implement a wide range of deformations by defining an appropriate $\bar{v}$. This velocity term is often a combination of several other terms, including data-dependent terms, geometric terms (e.g. curvature), and others. In many applications, these velocities introduce free parameters, and the proper tuning of those parameters is critical to making the level-set model behave in a desirable manner. Equation (1) is the general form of the level-set equation, which can be tuned for wide variety of problems and which motivates the architecture of our solver.

The proposed solver addresses the issues surrounding the solutions of (1). For this paper, however, we restrict the discussion on the particular form of this equation that is suitable for the segmentation application described in Sect. VI-A. This special case of (1) occurs when $\bar{v} = G(\bar{x}, \bar{t})\bar{n}$, where $\bar{n}$ is the surface normal and G is a scalar field, which we refer to as the *speed* of the level set. In this case (1) becomes

$$\partial\phi/\partial t = -|\nabla\phi|G. \tag{2}$$

Equation (2) describes a surface motion in the direction of the surface normal, and thus the volume enclosed by the surface expands or contracts, depending on the sign and magnitude of G.

Another important special case occurs when G, in (2), is the mean curvature of the level-set surface. The mean curvature of the level sets of ϕ are expressed as

$$H = \frac{1}{2}\nabla \cdot \frac{\nabla\phi}{|\nabla\phi|}. \tag{3}$$

In volume segmentation and surface reconstruction this mean curvature term is typically combined with an application-specific data term in order to obtain a smooth result that reflects interesting properties in the data.

There is a special case of (1) in which the surface motion is strictly inward or outward. In such cases the PDE can be solved somewhat efficiently using the *fast marching method* [3] and variations thereof [8]. However, this case covers only a very small subset of interesting speed functions. In general, we are concerned with solutions that allow the model to expand and contract as well as include a curvature term.

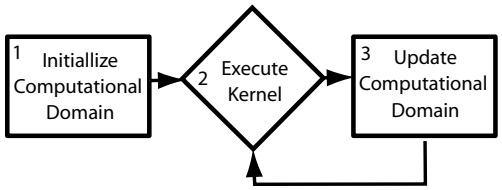


Fig. 2. The three fundamental steps in a sparse-grid solver. Step 1 initializes the sparse computational domain. Step 2 executes the computational kernel on each element in the domain. Step 3 updates the domain if necessary. Steps 2 and 3 are repeated for each solver iteration.

Efficient algorithms for solving the more general equation rely on the observation that at any one time step the only parts of the solution that are important are those adjacent to the moving surface (near points where $\phi = 0$). This observation places level-set solvers as part of a larger class of solvers that efficiently operate on time-dependent, sparse computational domains—i.e. a subset of the original problem domain (Figure 2).

Two of the most common CPU-based level-set solver techniques are the *narrow-band* [9] and *sparse-field* [6], [10] methods. Both approaches limit the computation to a narrow region near the isosurface yet store the complete computational domain in memory. The narrow-band approach implements the initialization and update steps in Figure 2 (Steps 1 and 3) by updating the embedding, ϕ, on a band of 10-20 pixels around the model, using a signed distance transform implemented with the fast marching method [3]. The band is reinitialized whenever the model (defined as a particular level set) approaches the edge. In contrast, the sparse-field method only traverses the complete domain during the initialization step of the algorithm in Figure 2. The sparse-field approach keeps a linked list of active data elements. The list is incrementally updated via a distance transform after each iteration. Even with this very narrow band of computation, update rates using conventional processors on typical resolutions (e.g. 256^3 voxels) are not interactive. This is the motivation behind our GPU-based solver. Although the new solver borrows ideas from both the narrow-band and sparse-field algorithms, it implements a new solution that conforms to the architectural restrictions of GPUs.

B. Scientific Computation on Graphics Processors

Graphics processing units have been developed primarily for the computer gaming industry, but over the last several years researchers have come to recognize them as a low cost, high performance computing platform. Two important trends in GPU development, increased programmability and higher precision arithmetic processing, have helped to foster new non-gaming applications.

For many data-parallel computations, graphics processors out-perform central processing units (CPUs) by more than an order of magnitude because of their *streaming* architecture [11] and dedicated high-speed memory. In the streaming model of computation, arrays of input data are processed identically by the same computation *kernel* to produce output data streams.

In contrast to vector architectures, the computation kernel in a streaming architecture may consist of many (possibly thousands) of instructions and use temporary registers to hold intermediate values. The GPU takes advantage of the data-level parallelism inherent in the streaming model by having many identical processing units execute the computation in parallel.

Currently GPUs must be programmed via graphics APIs such as OpenGL or DirectX. Therefore all computations must be cast in terms of computer graphics primitives such as vertices, textures, texture coordinates, etc. Figure 3 depicts the computation pipeline of a typical GPU. Vertices and texture coordinates are first processed by the vertex processor. The rasterizer then interpolates across the primitives defined by the vertices and generates *fragments* (i.e. pixels). The fragment processor applies textures and/or performs computations that determine the final pixel value. A *render pass* is a set of data passing completely through this pipeline. It can also be thought of as the complete processing of a stream by a given kernel (i.e. a *ForEach* call).

Grid-based computations are solved by first transferring the initial data into texture memory. The GPU performs the computation by rendering graphics primitives that access this texture. In the simplest case, a computation is performed on all elements of a 2D texture by drawing a quadrilateral that covers the same number of grid points (pixels) as the texture. Memory addresses that identify each fragment's data value as well as the location of its neighbors are given as texture coordinates. A fragment program (the kernel) then uses these addresses to read data from texture memory, perform the computation, and write the result back to texture memory. A 3D grid is processed as a sequence of 2D slices. This computation model has been used by a number of researchers to map a wide variety of computationally demanding problems to GPUs. Examples include matrix multiplication, finite element methods, multi-grid solvers, and others [12]–[14]. All of these examples demonstrate a homogeneous sequence of operations over a densely populated grid structure.

Strzodka et al. [15] were the first to show that the level-set equations could be solved using a graphics processor. Their solver implements the two-dimensional level-set method using a time-invariant speed function for flood-fill-like image segmentation, without the associated curvature. Lefohn and Whitaker demonstrate a full three dimensional level-set solver, with curvature, running on a graphics processor [16]. Neither of these approaches, however, take advantage of the sparse nature of level-set PDEs and therefore they perform only marginally better (e.g. twice as fast) than sparse or narrow band CPU implementations.

This paper presents a GPU computational model that supports *time-dependent, sparse grid* problems. These problems are difficult to solve efficiently with GPUs for two reasons. The first is that in order to take advantage of the GPU's parallelism, the streams being processed must be large, contiguous blocks of data, and thus grid points near the level-set surface model must be *packed* into a small number of textures. The second difficulty is that the level set moves with each time step, and thus the packed representation must readily adapt to the changing position of the model. This requirement is in contrast to the recent sparse matrix solvers [17], [18] and previous work on rendering with compressed data [19], [20]. Recent work by Sherbondy et al. [21] describes an alternative time-dependent, sparse GPU computation model which is discussed in Section VI-C.

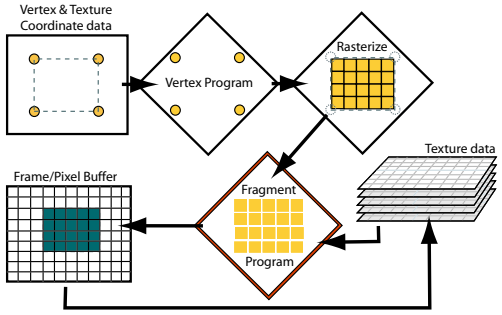


Fig. 3. The modern graphics processor pipeline.

C. Hardware-Accelerated Volume Rendering

Volume rendering is a flexible and efficient technique for creating images from 3D data [22]–[24]. With the advent of dedicated hardware for rasterization and texturing, interactive volume rendering has become one of the most widely used techniques for visualizing moderately sized 3D rectilinear data [25], [26]. In recent years, graphics hardware has become more programmable, permitting rendering features with an image quality that rival sophisticated software techniques [27], [28]. In this paper, we describe a novel volume rendering system that leverages programmable graphics hardware to render the packed level-set solution data.

III. A Virtual Memory Address Scheme for Sparse Computation

The limited computational capabilities of modern GPUs, their data-parallel streaming architecture, and our goal of interactive performance impose some important design restrictions on the proposed solver. For instance, the data-parallel computation model requires *homogeneous operations* on the entire computational domain, and memory constraints require us to process *and store* only the active domain on the computational processor (i.e. the GPU). Furthermore, GPUs do not support *scatter* write operations, and the communication bandwidth between the GPU and CPU is insufficient to allow transmission of any significant portion of the computational domain. Our new streaming, narrow-band level-set solver works efficiently within these restrictions and leverages GPU capabilities by packing the active computational domain into 2D texture memory. The GPU solves the 3D, level-set PDE directly on this packed format and quickly updates the packed representation after each solver iteration.

Re-mapping the computational domain (a subset of a volume) to take advantage of the GPU's capabilities has the unfortunate effect of making the computational kernels extremely complicated—that is difficult to design, debug, and modify. The kernel programmer must take the physical memory layout into consideration each time the kernel addresses memory.

Other researchers have successfully re-mapped computational domains to efficiently leverage the GPU's capabilities [12], [17], [18], [29], but they invariably describe these complex kernels in terms of the physical memory layout. This section presents a solution to this problem for level-set computation that allows kernels to access memory as if it were stored in the original, 3D domain—irrespective of the 2D physical layout used on the GPU. Our solution is an extension to the virtual memory systems used in modern operating systems.

A. Traditional Virtual Memory Overview

Nearly all modern operating systems contain a virtual memory system [30]. The purpose of virtual memory is to give the programmer the illusion that the application has access to a contiguous memory address space, while allowing the operating system to allocate memory for each process on demand, in manageable increments, from whatever physical resources happen to be available. Note that there are two meanings of virtual memory. The first is the mapping from a logical address space to a physical address space. The second is the mechanism for mapping logical memory onto a physical memory hierarchy (e.g. main memory, disk, etc). For this discussion, virtual memory only refers to the former definition.

Virtual memory works by adding a level of indirection between physical memory and the memory accessed by an application. Most conventional virtual memory systems divide physical and virtual memory into equally sized *pages*. The data addressed by an application's contiguous virtual address space will often be stored in many, disconnected physical memory pages. A *page table* tracks the mapping from virtual to physical memory pages. When an application requests memory, the system allocates physical memory pages and updates the page table. Note that the virtual and physical pages are identically sized.

When an application accesses memory via a virtual address, the system must first perform a virtual-to-physical address translation. The virtual address, VA, is first converted to a virtual page number, VPN. The system uses the page table to convert the VPN to a physical page address, PPA. The PPA is the physical address of the first element in a page. Finally, the memory system obtains the physical address, PA, by adding the PPA to the offset, OFF. The OFF is the linear distance between the virtual address and the beginning of the virtual page which contains it. The address computation is

$$\begin{aligned} \mathrm{VPN} &\leftarrow \tfrac{\mathrm{VA}}{S[\mathrm{P}]} \\ \mathrm{PPA} &\leftarrow \mathrm{PageTable}(\mathrm{VPN}) \\ \mathrm{OFF} &\leftarrow \mathrm{mod}(\mathrm{VA}, \mathrm{S[P]}) \\ \mathrm{PA} &\leftarrow \mathrm{PPA} + \mathrm{OFF}, \end{aligned} \tag{4}$$

where S[P] is the size of a memory page.

B. Multi-Dimensional Virtual Memory for GPUs

The virtual memory system used in our solver is a multi-dimensional extension of the traditional virtual memory system described in Section III-A.

Traditional virtual memory systems use one-dimensional virtual and physical address spaces. Our system uses a 3D virtual and a 2D physical memory address space. We use a 3D virtual memory space because the level-set computation is inherently volumetric. The 2D physical memory address space is motivated by the fact that GPUs are optimized to process 2D memory regions. By using a 2D physical address space, we are able to process the *entire* active volumetric domain simultaneously. This maximizes the benefit of the parallel, SIMD architecture of the GPU. We also make the simplifying assumption that virtual and physical pages are identical in dimension and size. Thus, the virtual space is not partitioned equally in all axes: 2D pages must be stacked in 3D to populate the problem domain as seen in Figure 4. Our system uses pages of size $S[\mathrm{P}] = (16, 16)$. This size represents a good compromise between a tight fit to the narrow computational domain and the overhead of managing and computing pages. Empirical results validate this choice.

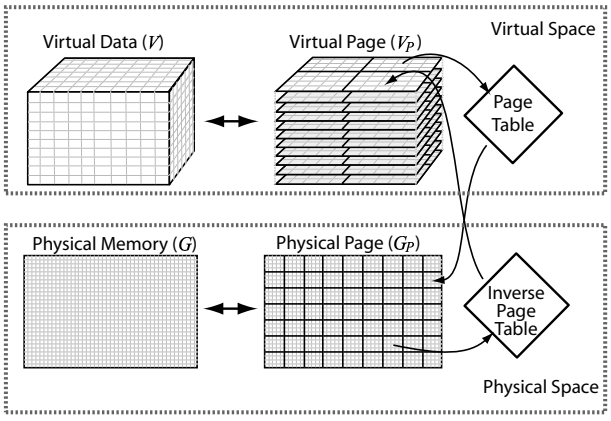


Fig. 4. The multi-dimensional virtual and physical memory spaces used in our virtual memory system. The original problem space is V, the virtual address space. The virtual page space, $\mathrm{V_P}$, is a subdivided version of V. Virtual memory pages are mapped to the physical page space, $\mathrm{G_P}$, by the *page table*. The *inverse page table* maps physical pages in $\mathrm{G_P}$ to virtual pages in $\mathrm{V_P}$. The collection of all elements in $\mathrm{G_P}$ constitute G, the physical memory of the hardware.

We now introduce notation for the various address spaces in our system. We notate the space of K-length vectors of integers as $\mathbb{Z}^K$. The set of all voxels in the 3D virtual address space (i.e. the problem domain) is defined as $\mathrm{V} \subset \mathbb{Z}^3$. Each of the virtual memory pages is a set of contiguous voxels in V; the space of all virtual pages is $\mathrm{V_P}$ (Figure 4). Similarly, the physical address space, $\mathrm{G} \subset \mathbb{Z}^2$, is subdivided into pages to form the physical page space, $\mathrm{G_P}$. The elements within a virtual or physical page are addressed identically using elements of $\mathrm{P} \subset \mathbb{Z}^2$. We also define a size operator for the 2D and 3D spaces described above. For X in $\{\mathrm{V}, \mathrm{V_P}, \mathrm{G}, \mathrm{G_P}, \mathrm{P}\}$, we define $S[X]$ to be a 2-vector or 3-vector (according to the dimension of X) giving the number of elements along each axis of the space X. Note that $S[\mathrm{V_P}] = \mathrm{S[V]/S[P]}$ and $S[\mathrm{G_P}] = \mathrm{S[G]/S[P]}$ (using component-wise division).

Virtual-to-physical address translation in a multi-dimensional virtual memory system works analogously to the 1D algorithm. Virtual addresses are now 3D position vectors in V and physical addresses are 2D vectors in G. The page table is a 3D table that returns 2D physical page

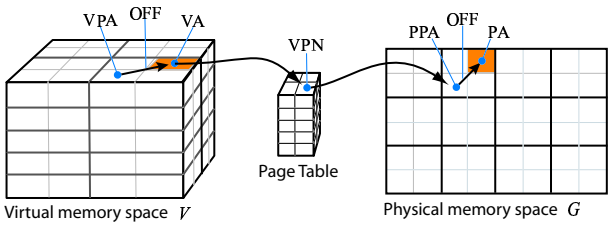


Fig. 5. The virtual-to-physical address translation scheme in our multi-dimensional virtual memory system. A 3D virtual address, VA, is first translated to a virtual page number, VPN. A page table translates the VPN to a physical page address, PPA. The PPA specifies the origin of the physical page containing the physical address, PA. The offset is then computed based on the virtual address and used to obtain the final 2D physical address, PA.

addresses. With these multi-dimensional definitions in mind, Eq (4) still applies to the vector-valued quantities. Figure 5 shows an example multi-dimensional address translation.

For the level-set solver in this paper, the multi-dimensional virtual memory system is implemented in part by the CPU and in part by the GPU. The CPU manages the page table, handles memory allocation/deallocation requests, and translates VPNs to PPAs. The GPU issues memory allocation/deallocation requests and computes physical addresses. We further divide the GPU tasks between the various processors on the GPU. The fragment processor creates memory allocation/deallocation requests. The address translation implementation uses the vertex processor and rasterizer to compute all PAs. Sections III-C and III-D describe the architectural and efficiency reasons for assigning the various virtual memory tasks to specific processors.

C. Virtual-to-Physical Address Translation

This section explains the details of the virtual-to-physical address scheme used in our GPU-based virtual memory system. Because the translation algorithm is executed each time the kernel accesses memory, its optimization is fundamental to the success of our method.

The simplest and most general way to implement the virtual-to-physical address translation for a GPU-based virtual memory system is to directly implement the computation in (4) and store the page table on the GPU as a 3D texture. A significant benefit of this approach is that it is completely general. Unfortunately, without dedicated memory-management hardware to accelerate the translation, this scheme suffers from several efficiency problems. First, the page table lookup means that a *dependent* texture read is required for each memory access. A dependent texture is defined as using the result of one texture lookup to index into another. This may cause a significant loss in performance on current GPUs. Second, storing the page table on the GPU consumes limited texture memory. The third problem is that a divide, modulus, and addition operation are required for each memory access. This consumes costly and limited fragment program instructions. Note that Section III-D discusses other problems with storing the page table on the GPU related to the limited capabilities of current GPU architectures.

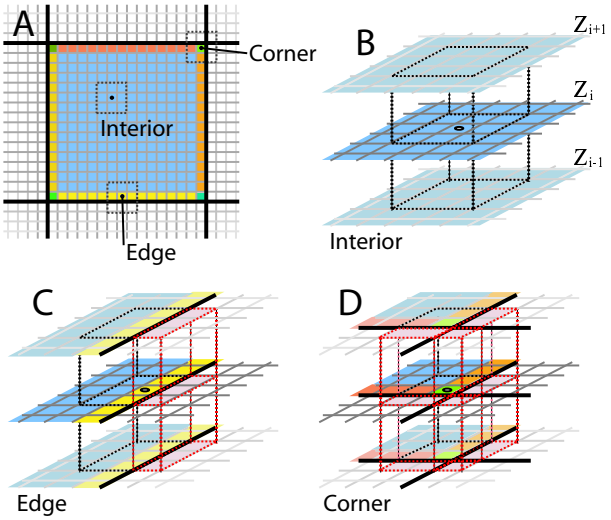


Fig. 6. The *substream* boundary cases used to statically resolve the conditionals arising from $3 \times 3 \times 3$ neighbor accesses across memory page boundaries. The nine *substream* cases are: interior, left edge, right edge, top edge, bottom edge, lower-left corner, lower-right corner, upper-right corner, and upper-left corner (a). The interior case accesses its neighbors from only three memory pages (b). The edge cases require six pages (c), and the corner cases require twelve memory pages (d). Note that for reasonably large page sizes, the more cache-friendly interior case has by far the highest number of data elements.

We can avoid the memory and computational inefficiencies that arise from storing the page table on the GPU by examining the pattern of virtual addresses required by the application's fragment program. In the case of our level-set solver, the fragment programs only use virtual addresses within a $3 \times 3 \times 3$ neighborhood of each active data element. This means that each active memory page will only access adjacent virtual memory pages (Figure 6). Moreover, we show that this simplified translation case makes it possible to lift the entire address translation from the fragment processor to the vertex processor and rasterizer.

Once we resolve the virtual addresses used by a fragment program, we can determine which virtual pages each active page will access. With this *relative page* information, the GPU can perform the virtual-to-physical address translation without a page table in texture memory. The CPU makes this possible by sending the PPAs for all required pages to the GPU as texture coordinates. The GPU can then use the relative neighbor offset vectors to decide which adjacent page contains the requested value (see Figure 6(a)).

The GPU's task of deciding which adjacent page contains a specific neighbor value unfortunately requires a significant amount of conditional logic. This logic must classify each data element into one of nine boundary cases: one of the four corners, one of the four edges, or an interior element (see Figure 6). Unfortunately current fragment processors do not support conditional execution. This logic could alternatively be encoded into a texture; however, this would again force the use of an expensive dependent texture read. Just as statically resolving virtual addresses allowed us to optimize the GPU computation, all active data elements can be pre-classified into the nine boundary cases. The result is that all

memory addresses used in each case will lie on the same pages relative to each active page (see Figure 6). In other words, the memory-page-locating logic has been statically resolved by pre-classifying data elements into their respective boundary cases. The data elements for these *substream* cases are generated by drawing unique geometry for each case. The corner substream cases are represented as points, the edges as lines, and the interior regions as quadrilaterals.

Kapasi et al. [31] describe an efficient solution to conditional execution in streaming architectures. Their solution is to route stream elements to different processing elements based on the code branch. Substreams are merely a static implementation of this data routing solution to conditional execution. The advantage is that the computation kernel run on each substream contains no conditional logic and is optimized specifically for that case. Our solution additionally gains from optimized cache behavior for the most common, interior, case (77% of the data points in a 16×16 page). The interior data elements require only three memory pages to access all neighbors (Figure 6(b)). In comparison, reading all neighbors for an edge element requires loading six pages (Figure 6(c)). The corner cases require twelve pages from disparate regions of physical memory(Figure 6(d)). The corner cases account for less than 2% of the active data elements.

With the use of substreams, the GPU can additionally optimize the address computation by computing physical addresses with the vertex processor rather than the fragment processor. Because all data elements (i.e. fragments) use exactly the same relative memory addresses, the offset and physical address computation steps of (4) can be generated by interpolating between substream vertex locations. The vertex processor and rasterizer can thus perform the entire address translation. This optimization distributes computational load to under-utilized processing units and reduces the number of limited and expensive fragment instructions.

D. Bootstrapping the Virtual Memory System

This section describes the steps required to initialize the GPU virtual memory system. To begin, the application specifies the page size, S[P], the virtual page space size, $\mathrm{S}[\mathrm{V_P}]$, and the fundamental data type to use (i.e. 32-bit floating point, 16-bit fixed point, etc.). The virtual memory system then allocates an initial physical memory buffer on the GPU. It also creates a page table, an inverse page table, a geometry engine, and a stack of free pages on the CPU. The decision to place the aforementioned data structures on the CPU is based on the efficiency concerns described in Section III-C as well as GPU architectural restrictions. These restrictions include: the GPU's lack of random write access to memory, lack of writable 3D textures, lack of dynamically sized output buffers, and limited GPU memory.

The page table is defined to store a `MemoryPage` object that contains the vertices and texture coordinates required by the GPU to access the physical memory page. The inverse page table is designed to store a VPN vector for each active physical page. Figure 5 shows these mappings. Note that the page table and inverse page table were referred to as the *unpacked map* and *packed map* respectively in Lefohn et al. [32].

The vertices and texture coordinates stored in the `MemoryPage` object are actually pointers into the geometry engine. The geometry engine has the capability of quickly rendering (i.e. processing) any portion of the physical memory domain. Thus the geometry engine must generate the substreams for the set of active physical pages. The last initialization step is the creation of the free-page stack. The virtual memory system simply pushes all physical pages (i.e. pointers to `MemoryPage` objects) defined by the geometry engine onto a stack.

The application issues GPU physical memory allocation and deallocation requests to the virtual memory system. Upon receiving a virtual page request, the system pops a physical page from the free-page stack, updates the page tables, and returns a `MemoryPage` pointer to the application. The reverse process occurs when the application deallocates a virtual memory page.

The level-set solver generates memory page allocation and deallocation requests after each solver iteration based on the form of the current solution. Section IV-D describes how the solver uses the GPU to efficiently create these memory requests.

IV. Sparse GPU Level-Set Solver

This section now explains our GPU level-set solver implementation using the virtual memory system and level-set equations presented in Section III and Section II-A. Note that the details of the level-set discretization are found in Lefohn et al. [33].

A. Initialization of Computational Domain

The solver begins by initializing the sparse computational domain (Step 1 in Figure 2). An initial level-set volume is passed to the level-set solver by the host application. The sparse domain initialization involves identifying active memory pages in the input volume, allocating GPU memory for each active page, then sending the initial data to the GPU.

The solver identifies active virtual pages by checking each data element for a non-zero derivative value in any of the six cardinal directions. If any element in a page contains non-zero derivatives, the entire page is activated. The initialization code then requests a GPU memory page from the virtual memory system for each active page. The level-set data is then *drawn* into GPU memory using the vertex locations in each `MemoryPage` object.

This scheme is effective only because the input level-set volume is assumed to be a clamped distance transform– meaning that regions on or near the isosurface have non-zero gradients while regions outside or inside the surface have gradients of zero. The outside voxels have a value of zero (black) and the inside ones have a value of one (white). Section IV-B explains how the distance transform embedding is maintained throughout the level-set computation.

The inactive virtual pages do not need to be represented in physical memory. If an active data element queries an inactive value, however, an appropriate value needs to be returned. Because all inactive regions are either uniformly black or

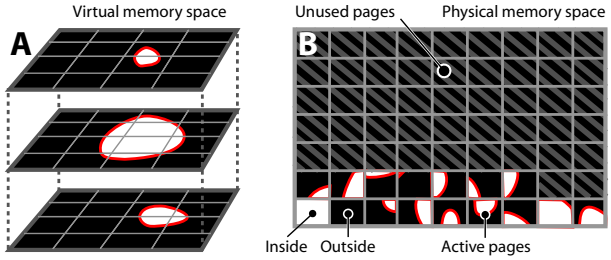


Fig. 7. The level-set solver's use of the paged virtual memory system. All *active* pages (i.e. those that contain non-zero derivatives) in the virtual page space (a) are mapped to unique pages of physical memory (b). The inactive virtual pages are mapped to the static *inside* or *outside* physical page. Note that the only data stored on the GPU is that represented by (b).

white, we solve this boundary condition problem by defining a special, inactive page state. A virtual page in this state is mapped to one of two *static* physical pages. One of these static pages is black, representing regions outside of the level-set surface. The other static page is white and represents regions inside the level-set surface. The page table contains these many-to-one mappings, but the inverse page table does not store a valid entry for the static pages. Note that we could have alternatively solved this boundary problem using single pixels instead of entire pages. We also could have solved the problem by creating substreams for the active elements on the boundary of the active set.

B. Distance Transform on the GPU

In order to take advantage of the sparse nature of level-set solutions, algorithms must maintain a somewhat consistent *level-set density*, which is defined as the number of level sets per unit volume. If the level-set density becomes too low (spread out) it can become difficult to efficiently isolate the computation to the desired interface. Alternatively, a level-set density that becomes too high (close together) can cause aliasing and numerical problems. The most common way of maintaining a desired level-set density is to keep the embedding, ϕ, resembling a distance transform [6], [9], [34].

The new streaming level-set solver maintains the distance transform by introducing an additional speed term, G_r, to the level-set PDE (1) that controls the surface motion. This speed term *pushes* the level sets of ϕ, either closer together or farther apart, so that they resemble a clamped distance transform (CDT). The CDT has a constant level-set density within a predefined band and ensures that voxels near the isosurface have finite derivatives while those farther away have gradient magnitudes of zero. As described in Sections IV-A and IV-D, the identification of zero-derivative regions is critical for an efficient solver implementation. This *rescaling* speed term, G_r, is computed as

$$G_r = \phi g_\phi - \phi|\nabla\phi|, \tag{5}$$

where g_ϕ is the target gradient magnitude within the computational domain, and $|\nabla\phi|$ is the gradient magnitude in the direction of the level-set model isosurface. The target parameter, g_ϕ, can be set based on the numerical precision of the level-set data. By setting g_ϕ sufficiently high, numerical errors caused by underflow can easily be avoided. It is important to note that G_r is strictly a numerical construct; it does not affect the movement of the zero level set, i.e. the surface model. Also note that the solver can be used to compute *only* the distance transform (i.e. no surface movement) by setting g_ϕ to one and making G_r the only speed term.

C. Level-Set Computation

The GPU next performs the level-set computation (Step 2 of the sparse algorithm in Figure 2). The details of the level-set discretization used by our solver are given in Lefohn et al. [33]. This section gives a high-level overview of the computation. The level-set update proceeds in the following steps:

A. Compute 1st and 2nd partial derivatives.
B. Compute N level-set speed terms.
C. Update level-set PDE.

The derivative computation in Step A above uses the substream-based, virtual-to-physical address scheme described in Section III-C. The derivatives are computed in nine sub-stream render passes, each of which outputs to the same four, 4-tuple buffers. The speed function computations in Step B are application-dependent. Example speed terms include the curvature computation described in (3), the rescaling term described in (5), and the thresholding term described in (7). There will be zero or more render passes for each speed function. The level-set update (Step C) is the up-wind scheme described in Lefohn et al. [33]. This is computed in a single pass. Note that additional GPU memory must be allocated to store the intermediate results accumulated in Steps A and B before they are consumed in Step C. Our solver performs register allocation of temporary buffers to minimize GPU memory usage.

D. Update of Computational Domain

After each level-set update, the solver determines which virtual pages need to be added-to or removed-from the active domain. The solver accomplishes this by aggregating gradient information from all elements in each active page. In our solver, the GPU must compute this information because the level-set solution exists only in physical memory. The active set must be updated by the CPU, however, because the page table and geometry engine exist in CPU main memory. In addition, the amount of information passed from the GPU to the CPU must be kept to a minimum because of the limited bandwidth between the two processors. This section gives an overview of an algorithm that works within these constraints. Lefohn et al. [33] explains the full details of the algorithm.

The GPU creates a memory allocation/deallocation request by producing a small image (of size $S[\mathrm{G_P}]$) with a single-byte pixel per physical page. The value of each pixel is a bit code that encapsulates the activation or deactivation state of each page and its six adjacent neighbors (in $\mathrm{V_P}$). The CPU reads this small ($< 64\mathrm{kB}$) message, decodes it, and submits the allocation/deallocation requests to the virtual memory system (Figure 8).

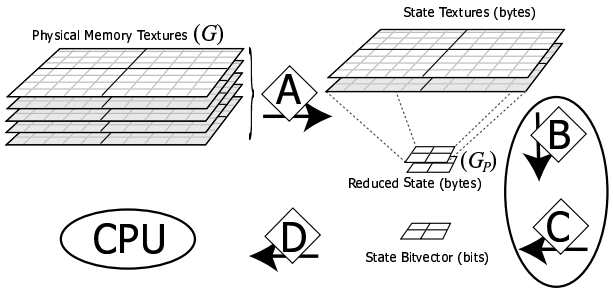


Fig. 8. The GPU's creation of a memory allocation/deallocation request. Step A uses solver-specific data to create two buffers containing the active state of each data element and its adjacent neighbors. Step B uses automatic mipmapping to reduce the buffers from size $S[\mathrm{G}]$ to the physical page space size, $S[\mathrm{G_P}]$. Step C combines the information from the two down-sampled state buffers into an eight-bit code for each pixel. This code encapsulates whether or not each active virtual memory page and its adjacent neighbors should be enabled. In step D, the CPU reads the bit-code buffer, decodes it, and allocates/deallocates pages as requested.

The GPU creates the bit-code image by first computing two, four-component neighbor information buffers of size $S[\mathrm{G}]$ (Step A of Figure 8). This computation uses the previously-computed, one-sided derivatives of ϕ to identify the required active pages. A page must be activated if it contains elements with non-zero gradient magnitudes. The automatic mipmapping GPU feature is then used to down-sample the resulting buffers (i.e. aggregate data samples) to the page-space image (Step B in Figure 8). The final GPU operation combines the active page information into the bit code (Step C in Figure 8). A fragment program performs this step by emulating a bit-wise OR operation via conditional addition of powers of two. Finally, in step D of Figure 8, the CPU reads this message from the GPU.

Note that the use of automatic mipmapping places some restrictions on the maximum memory page size due to quantization rounding errors that arise when down-sampling 8-bit values. This limitation can be relaxed by using a 16-bit fixed-point data type. Alternatively, floating-point values can be used if the down-sampling is performed with fragment program passes instead of automatic mipmapping.

E. GPU Implementation Details

The level-set solver and volume renderer are implemented in programmable graphics hardware using vertex and fragment programs on the ATI Radeon 9800 GPU. The programs are written in the OpenGL ARB_vertex_program and ARB_fragment_program assembly languages.

There are several details related to render pass output buffers that are critical to the performance of the level-set solver. First is the ability to output multiple, high-precision 4-tuple results from a fragment program. Writing sixteen scalar outputs from a single render pass enables us to perform the expensive 3D neighborhood reconstruction only once and use the gathered data to compute the derivatives in a single pass. Second, we avoid the expensive change between render targets [35] (i.e. pixel buffers) by allocating a single pixel buffer with many *render surfaces* (front, back, aux0, etc.) and using each surface as a separate output buffer.

Lastly, there is a subtle speed-versus-memory trade-off that must be carefully considered. Because the physical-memory texture can be as large as 2048^2, storing intermediate results (e.g. derivatives, speed values, etc.) during the computation can require a large amount of GPU memory. This memory requirement can be minimized by performing the level-set computation in sub-regions. The intermediate buffers must then be only the size of the sub-region. This partitioning does reduce computational efficiency, however, and so the sub-regions are made as large as possible. We currently use 512^2 sub-regions when the level-set texture is 2048^2 and use a single region when it is smaller.

V. Volume Rendering of Packed Data

The direct visualization of the level-set evolution is important for a variety of level-set applications. For instance, in the context of segmentation, direct visualization allows a user to immediately assess the quality and accuracy of the pending segmentation and steer the evolution toward the desired result. Volume rendering is a natural choice for visualizing the level-set surface model, because it does not require an intermediate geometric extraction, which would severely limit interactivity. If one were to use marching cubes, for instance, a distinct triangle mesh would need to be created (and rendered) for each iteration of the level-set solver. The proposed solver, therefore, includes a volume renderer, which produces a full 3D (transfer-function based) volume rendering of the evolving level set on the GPU [28].

For rendering the evolving level-set model, we use a variant of traditional 2D texture based volume rendering [25]. We modify the conventional approach to render the level-set solution directly from the packed physical memory layout, which is physically stored in a single 2D texture. Because the level-set data and physical page configuration are dynamic, it would be inefficient to pre-compute and store three separate versions of the data, sliced along cardinal views, as is typically done with 2D texture approaches. Instead we reconstruct these views each time the volume is rendered. This new technique is thus both applicable to rendering compressed data as well as traditional texture-based volume rendering from a single set of 2D slices.

The volume rendering algorithm utilizes a two pass approach for reconstruction and rendering. Figure 9 illustrates the steps involved. An additional off-screen buffer caches two reconstructed neighboring slices containing the level-set solution and its gradient (Figure 9 A). During the rendering phase arbitrary slices along the preferred slice direction are interpolated from these neighboring slices (Figure 9 B). Once all interpolated slices between slice i and $i-1$ are rendered and composited, the next slice $(i+1)$ is reconstructed. This newly reconstructed slice replaces the cached slice, $i-1$. The GPU then renders and composites the next set of interpolated slices (i.e. those between slice $i+1$ and i). This pattern continues until all slices have been reconstructed and rendered.

When the preferred slice axis, based on the viewing angle, is orthogonal to the virtual memory page layout, we reconstruct 2D slices of the level-set solution and its gradient using a

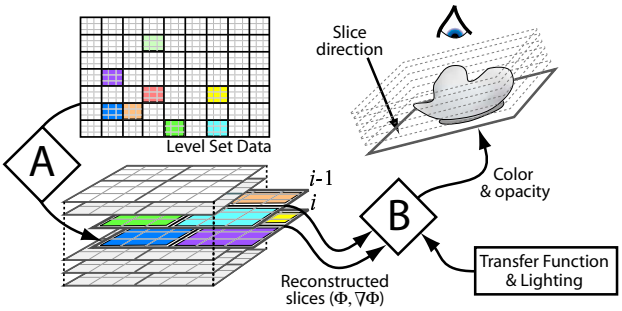


Fig. 9. Two pass rendering of packed volume data. In step A, a 2D slice (i) is reconstructed from the physical page (packed) layout, G_P. In step B, one or more intermediate slices between i and $i-1$ are interpolated, transformed into optical properties (via the transfer function), lit, and rendered for the current view. The next iteration begins by reconstructing slice $i+1$, replacing $i-1$, and so on.

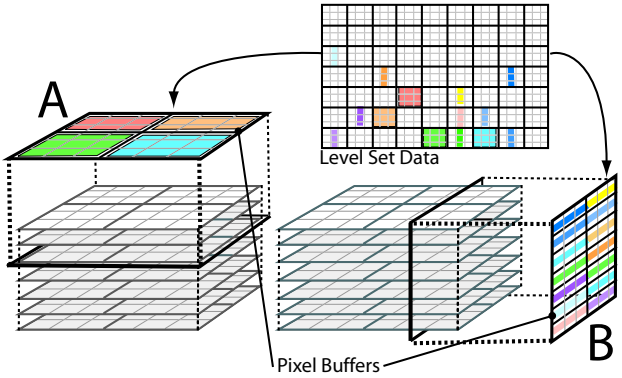


Fig. 10. Reconstruction of a slice for volume rendering the packed level-set model: (a) When the preferred slicing direction is orthogonal to the virtual memory page layout, the pages (shown in alternating colors) are draw into a pixel buffer as quadrilaterals. (b) For slicing directions parallel to the virtual page layout, the pages are drawn onto a pixel buffer as either vertical or horizontal lines.

textured quadrilateral for each page, as shown in Fig. 10 A. On the other hand, if the preferred slice direction is parallel to the virtual page layout, we render a row or column from each page using textured line primitives, as in Fig. 10 B. In both cases, slices are reconstructed into a pixel buffer which is bound as a texture in the rendering pass. These slices are reconstructed at the same resolution as level-set solution.

In the rendering phase, we leverage the hardware's bilinear filtering for in-plane interpolation of the reconstructed level-set slice. Trilinear interpolation of an arbitrary slice between two adjacent reconstructed slices is accomplished by combining them, *i.e.* performing linear interpolation along the preferred slice direction, in the fragment program. This same fragment program also evaluates the transfer function and lighting for the interpolated data. For efficiency, we also reuse data wherever possible. For instance, lighting for the level-set surface, evaluated in the rendering phase, uses gradient vectors computed during the level-set update stage.

VI. Application and Results

This section describes an application for interactive volume segmentation and visualization, which uses the level-set solver and volume renderer described previously. We show pictures from the system and present timing results relative to our current benchmark for level-set deformations, which is a highly optimized CPU solution [36].

A. Volume Segmentation With Level-Sets

For segmenting volume data with level sets, the speed functions usually consists of a combination of two terms [4], [37]

$$\frac{\partial \phi}{\partial t} = |\nabla \phi| \left[\alpha D(\bar{x}) + (1-\alpha) \nabla \cdot \frac{\nabla \phi}{|\nabla \phi|} \right], \qquad (6)$$

where D is a data term that forces the model to expand or contract toward desirable features in the input data (which we also call the *source* data), the term $\nabla \cdot (\nabla \phi / |\nabla \phi|)$ is the mean curvature H of the surface, which forces the surface to have less area (and remain smooth), and $\alpha \in [0,1]$ is a free parameter that controls the degree of smoothness in the solution.

This combination of a data-fitting speed function with the curvature term is critical to the application of level sets to volume segmentation. Most level-set data terms D from the segmentation literature are equivalent to well-known algorithms such as isosurfaces, flood fill, or edge detection when used without the smoothing term (i.e. $\alpha = 1$). The smoothing term alleviates the effects of noise and small imperfections in the data, and can prevent the model from leaking into unwanted areas. Thus, the level-set surface models provide several capabilities that complement volume rendering: local, user-defined control; smooth surface normals for better rendering of noisy data; and a closed surface model, which can be used in subsequent processing or for quantitative shape analysis.

For the work in this paper we have chosen a simple speed function to demonstrate the effectiveness of *interactivity* and *real-time visualization* in level-set solvers. The speed function we use in this work depends solely on the greyscale value input data I at the point $\bar{x}$:

$$D(I) = \epsilon - |I - T|, \qquad (7)$$

where T controls the brightness of the region to be segmented and ϵ controls the range of greyscale values around T that could be considered inside the object. In this way a model situated on voxels with greyscale values in the interval $T \pm \epsilon$ will expand to enclose that voxel, whereas a model situated on greyscale values outside that interval will contract to exclude that voxel. The speed term is gradual, as shown in Fig. 11, and thus the effects of D diminish as the model approaches the boundaries of regions with greyscale levels within the $T \pm \epsilon$ range. This makes the effects of the curvature term relatively larger. This choice of D corresponds to a simple, one-dimensional statistical classifier on the volume intensity [38].

To control the model a user specifies three free parameters, T, ϵ, and α, as well as an initialization. The user generally draws a spherical initialization inside the region to be segmented. Note that the user can alternatively initialize the solver with a pre-processed (thresholded, flood filled, etc.) version of the source data.

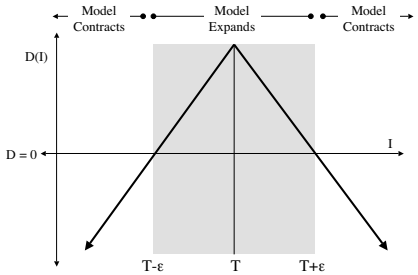


Fig. 11. A speed function based on image intensity causes the model to expand over regions with greyscale values within the specified (positive) range and contract otherwise.

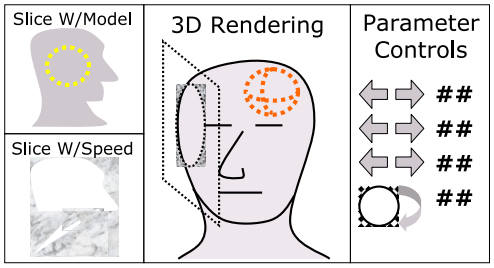


Fig. 12. A depiction of the user interface for the volume analysis application. Users interact via slice views, a 3D rendering, and a control panel.

B. Interface and Usage

The application in this paper consists of a graphical user interface that presents the user with two slice viewing windows, a volume renderer, and a control panel. (Fig. 12). Many of the controls are duplicated throughout the windows to allow the user to interact with the data and solver through these various views. Two and three dimensional representations of the level-set surface are displayed in real time as it evolves.

The first 2D window displays the current segmentation as a yellow line overlaid on top of the source data. The second 2D window displays a visualization of the level-set speed function that clearly delineates the positive and negative regions. The first window can be probed with the mouse to accomplish three tasks: set the level-set speed function, set the volume rendering transfer function, and draw 3D spherical initializations for the level-set solver. The first two tasks are accomplished by accumulating an average and variance for values probed with the cursor. In the case of the speed function, the T is set to the average and ϵ is set to the standard deviation. Users can modify these values, via the GUI, while the level set deforms. The spherical drawing tool is used to initialize and/or edit the level-set surface. The user can add-to or subtract-from the model by drawing white or black spheres, respectively. This feature gives the user "3D paint" and "3D eraser" tools with which to interactively edit the level-set solution.

The volume renderer displays a 3D reconstruction of the current level set isosurface (see Section V) as well as the input data. In addition, an arbitrary clipping plane, with texture-mapped source data, can be enabled via the GUI (Figure 1). Just as in the slice viewer, the speed function, transfer function, and level-set initialization can be set through probing on this clipping plane. The crossing of the level-set isosurface with the clipping plane is also shown in bright yellow.

The volume renderer uses a 2D transfer function to render the level set surface and a 3D transfer function to render the source data. The level-set transfer function axes are intensity and distance from the clipping plane (if enabled). The transfer function for rendering the original data is based on the source data value, gradient magnitude, and the level-set data value. The latter is included so that the level set model can function as a region-of-interest specifier. All of the transfer functions are evaluated on-the-fly in fragment programs rather than in lookup tables. This approach permits the use of arbitrarily high dimensional transfer functions, allows run-time flexibility, and reduces memory requirements [39].

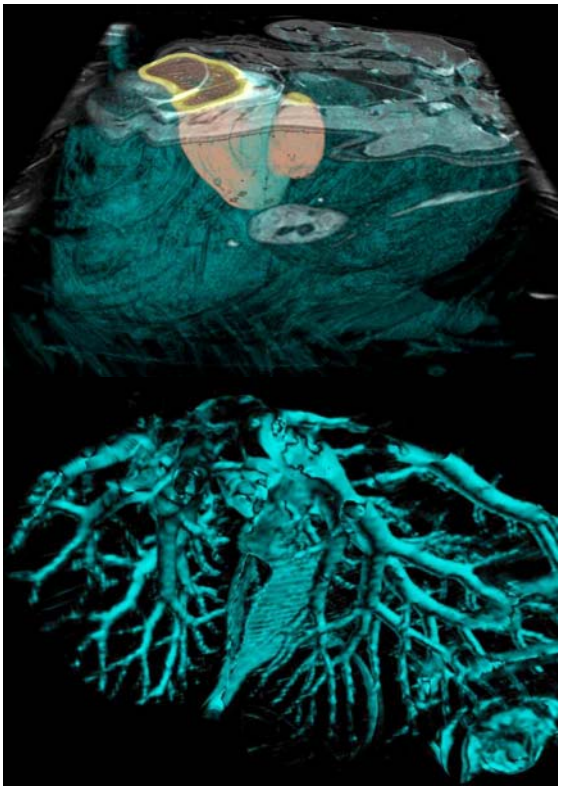


Fig. 13. (top) Volume rendering of a 256^3 MRI scan of a mouse thorax. Note the level-set surface which is deformed to segment the liver. (bottom) Volume rendering of the vasculature inside the liver using the same transfer function as in (top) with the level-set surface is being used as a region-of-interest specifier.

We demonstrate our interactive level-set solver and volume rendering system with the following three data sets: a brain tumor MRI (Fig. 1), an MRI scan of a mouse (Fig. 13) and transmission electron tomography data of a gap junction (Fig. 14). In all of these examples a user interactively controls the level-set surface evolution and volume rendering via the multi-view interface. The initializations for the tumor and mouse were drawn via the user interface. The initialization for Figure 14 was seeded with a thresholded version of the source data.

C. Performance Analysis

Our GPU-based level-set solver achieves a speedup of ten to fifteen times over a highly-optimized, sparse-field, CPU-based implementation [36]. All benchmarks were run on an Intel Xeon 1.7 GHz processor with 1 GB of RAM and an

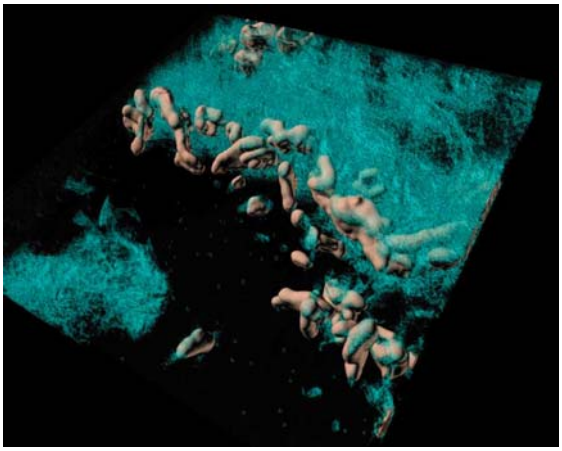


Fig. 14. Segmentation and volume rendering of $512 \times 512 \times 61$ 3D transmission electron tomography data. The picture shows cytoskeletal membrane extensions and connexins (pink surfaces extracted with the level-set models) near the gap junction between two cells (volume rendered in cyan).

ATI Radeon 9800 Pro GPU. All timings include the complete computation, i.e. both the virtual memory system update and the level-set computation are included. For a $256 \times 256 \times 175$ volume, the level-set solver runs at rates varying from 70 steps per second for the tumor segmentation (Fig. 1) to 3.5 steps per second for the final stages of the cortex segmentation from the same data set. In contrast, the CPU-based, sparse field implementation ran at 7 steps per second for the tumor and 0.25 steps per second for the cortex segmentation.

The speed of our solver is bound almost entirely by the fragment stage of the GPU. In addition, the speed of our solver scales linearly with the number of active voxels in the computation. Creation of the bit vector message consumes approximately 15% of the GPU arithmetic and texture instructions, but for most applications the speedup over a dense GPU-based implementation far eclipses this additional overhead.

The amount of texture memory required for the level-set computation is proportional to the surface area of the level-set surface—i.e. the number of active pages. Our tests have shown that for many applications, only 10%-30% of the volume is active. To take full advantage of this savings, the total size of physical memory, $S[\mathrm{G}]$, must increase when the number of allocated pages grows beyond the physical memory's capacity. Our current implementation performs only static allocation of the maximum physical memory space, but future versions could easily realize the above memory savings. Section VII discusses changes to GPU display drivers that will facilitate the implementation of this feature.

In comparison to the depth-culling-based sparse volume computation presented by Sherbondy et al. [21], our packing scheme guarantees that very few wasted fragments are generated by the rasterization stage. This is especially important for sparse computations on large volumes—where the rasterization and culling of unused fragments could consume a significant portion of the execution time. In addition, the packing strategy allows us to process the entire active data set simultaneously, rather than slice-by-slice. This improves the computationally efficiency by taking advantage of the GPU's deep pipelines and parallel execution. Our algorithm should also be able to process larger volumes, due to the memory savings discussed above. Our algorithm, however, does incur overhead associated with maintaining the packed tiles, and more experimentation is necessary to understand the circumstances under which each approach is advantageous. Furthermore, they are not mutually exclusive, and Sect. VII discusses the possibility of using depth culling in combination with our packed representation.

As with any sparse algorithm, it will be advantageous to simply compute the entire (original) domain if the active domain becomes sufficiently large. Our experience with segmentation thus far, however, has shown that the the computation remains sufficiently sparse even for large structures such as a cerebral cortex segmentation. The sparseness is due to the fact that only the surface needs to represented, and the interior regions need not be represented or computed.

VII. Conclusions and Future Work

This papers demonstrates a new tool for interactive volume exploration and analysis that combines the quantitative capabilities of deformable isosurfaces with the qualitative power of volume rendering. By relying on graphics hardware, the level-set solver operates at interactive rates (approximately 15 times faster than previous solutions). This mapping relies on an efficient multi-dimensional virtual memory system to implement a time-dependent, sparse computation scheme. The memory mappings are updated via a novel GPU-to-CPU message passing algorithm. The GPU renders the level-set surface model directly from a sparse, compressed texture format. Future extensions and applications of the level-set solver include the processing of multivariate data as well as surface reconstruction and surface processing. Most of these only involve changing only the speed functions.

There are a couple ways in which the memory and computational efficiency of our solver can be improved. First, it may be worth achieving an even narrower band of computation around the level-set model. This is possible by using depth culling to avoid computation on inactive elements within each active page [21]. Implementing this depth culling requires a memory model in which an arbitrary number of data buffers can access a single depth buffer. The second optimization is to allow the total amount of physical memory to change at run time and grow to the limits of GPU memory. This requires spreading physical memory across multiple 2D textures (i.e. creating a 3D physical memory space). The proposed *super buffer* [40] OpenGL extension supports both of these proposed optimizations.

The GPU virtual memory abstraction also indicate promising future research. We are currently beginning work on a more general virtual memory implementation that fully abstracts N-dimensional GPU memory. The goal is to provide an API that allows a GPU application programmer to specify an optimal physical and virtual memory layout for their problem, then write the computational kernels irrespective of the physical layout. The kernels will specify memory accesses via abstract memory access interfaces, and an operating-system-like layer will replace these memory access calls with the appropriate address translation code.

ACKNOWLEDGMENTS

Thanks to Evan Hart, Mark Segal, Jeff Royal and Jason Mitchell at ATI for donating technical advice and hardware to this project. Gordon Kindlmann's *nrrd* toolkit was used for data set manipulation (`http://teem.sourceforge.net`). Milan Ikits' *GLEW* library was used for OpenGL extension management (`http://glew.sourceforge.net`). Steve Lamont and Gina Sosinsky at the National Center for Microscopy and Imaging Research at UCSD provided the tomography data. Simon Warfield, Michael Kaus, Ron Kikinis, Peter Black and Ferenc Jolesz provided the MRI head data. The mouse data was supplied by the Center for In Vivo Microscopy at Duke University. This work was supported by grants from NSF, ACI0089915 and CCR0092065, and ONR N000140110033. We also thank John Owens and the anonymous reviewers for their input on the manuscript.

REFERENCES

[1] S. Osher and J. Sethian, "Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations," *Journal of Computational Physics*, vol. 79, pp. 12–49, 1988.

[2] R. Fedkiw and S. Osher, *Level Set Methods and Dynamic Implicit Surfaces*. Springer, 2002.

[3] J. A. Sethian, *Level Set Methods and Fast Marching Methods Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge University Press, 1999.

[4] R. T. Whitaker, "Volumetric deformable models: Active blobs," in *Visualization In Biomedical Computing 1994* (R. A. Robb, ed.), (Mayo Clinic, Rochester, Minnesota), pp. 122–134, SPIE, 1994.

[5] T. Tasdizen, R. Whitaker, P. Burchard, and S. Osher, "Geometric surface smoothing via anisotropic diffusion of normals," in *IEEE Visualization*, pp. 125–132, October 2002.

[6] R. Whitaker, "A level-set approach to 3D reconstruction from range data," *International Journal of Computer Vision*, vol. October, pp. 203–231, 1998.

[7] T. Yoo, U. Neumann, H. Fuchs, S. Pizer, T. Cullip, J. Rhoades, and R. Whitaker, "Direct visualization of volume data," *IEEE Computer Graphics and Applications*, vol. 12, pp. 63–71, 1992.

[8] M. Droske, B. Meyer, M. Rumpf, and C. Schaller, "An adaptive level set method for medical image segmentation," in *Proc. of the Annual Symposium on Information Processing in Medical Imaging* (R. Leahy and M. Insana, eds.), Springer, Lecture Notes Computer Science, 2001.

[9] D. Adalsteinson and J. A. Sethian, "A fast level set method for propogating interfaces," *Journal of Computational Physics*, pp. 269–277, 1995.

[10] D. Peng, B. Merriman, S. Osher, H. Zhao, and M. Kang, "A PDE based fast local level set method," *Journal of Computational Physics*, vol. 155, pp. 410–438, 1999.

[11] J. Owens, *Computer Graphics on a Stream Architecture*. PhD thesis, Stanford University, Nov. 2002.

[12] N. Goodnight, C. Woolley, G. Lewin, D. Luebke, and G. Humphreys, "A multigrid solver for boundary value problems using programmable graphics hardware," in *Graphics Hardware 2003*, pp. 102–111, July 2003.

[13] E. S. Larsen and D. McAllister, "Fast matrix multiplies using graphics hardware," in *Super Computing 2001*, ACM SIGARCH/IEEE, Nov. 2001.

[14] R. Strzodka and M. Rumpf, "Using graphics cards for quantized FEM computations," in *Proceedings VIIP Conference on Visualization and Image Processing*, 2001.

[15] M. Rumpf and R. Strzodka, "Level set segmentation in graphics hardware," in *International Conference on Image Processing*, pp. 1103–1106, 2001.

[16] A. E. Lefohn and R. T. Whitaker, "A GPU-based, three-dimensional level set solver with curvature flow." University of Utah tech report UUCS-02-017, December 2002.

[17] J. Bolz, I. Farmer, E. Grinspun, and P. Schröder, "Sparse matrix solvers on the GPU: Conjugate gradients and multigrid," in *ACM Transactions on Graphics*, vol. 22, pp. 917–924, July 2003.

[18] J. Krüger and R. Westermann, "Linear algebra operators for GPU implementation of numerical algorithms," in *ACM Transactions on Graphics*, vol. 22, pp. 908–916, July 2003.

[19] A. C. Beers, M. Agrawala, and N. Chaddha, "Rendering from compressed textures," in *Proceedings of SIGGRAPH 96*, Computer Graphics Proceedings, Annual Conference Series, pp. 373–378, Aug. 1996.

[20] M. Kraus and T. Ertl, "Adaptive texture maps," in *Graphics Hardware 2002*, pp. 7–16, Sept. 2002.

[21] A. Sherbondy, M. Houston, and S. Nepal, "Fast volume segmentation with simultaneous visualization using programmable graphics hardware," in *IEEE Visualization*, pp. 171–176, October 2003.

[22] R. A. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering," in *Computer Graphics (Proceedings of SIGGRAPH 88)*, vol. 22, pp. 65–74, Aug. 1988.

[23] M. Levoy, "Display of surfaces from volume data," *IEEE Computer Graphics & Applications*, vol. 8, pp. 29–37, 1988.

[24] P. Sabella, "A rendering algorithm for visualizing 3D scalar fields," in *Computer Graphics (Proceedings of SIGGRAPH 88)*, vol. 22, pp. 51–58, Aug. 1988.

[25] B. Cabral, N. Cam, and J. Foran, "Accelerated volume rendering and tomographic reconstruction using texture mapping hardware," in *ACM Symposium On Volume Visualization*, pp. 91–98, Oct. 1994.

[26] O. Wilson, A. V. Gelder, and J. Wilhelms, "Direct Volume Rendering via 3D Textures," Tech. Rep. UCSC-CRL-94-19, University of California at Santa Cruz, June 1994.

[27] K. Engel, M. Kraus, and T. Ertl, "High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading," in *Graphics Hardware 2001*, 2001.

[28] J. Kniss, G. Kindlmann, and C. Hansen, "Multi-Dimensional Transfer Functions for Interactive Volume Rendering," *Transactions on Visualization and Computer Graphics*, vol. 8, pp. 270–285, July-September 2002.

[29] T. J. Purcell, I. Buck, W. R. Mark, and P. Hanrahan, "Ray tracing on programmable graphics hardware," *ACM Transactions on Graphics*, vol. 21, pp. 703–712, July 2002. ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).

[30] A. Silberschatz and P. Galvin, *Operating System Concepts*. Addison-Wesley, 1998.

[31] U. Kapasi, W. Dally, S. Rixner, P. Mattson, J. Owens, and B. Khailany, "Efficient conditional operations for data-parallel architectures," in *Proceedings of the 33rd Annual International Symposium on Microarchitecture*, pp. 159–170, 2000.

[32] A. E. Lefohn, J. Kniss, C. Hansen, and R. Whitaker, "Interactive deformation and visualization of level set surfaces using graphics hardware," in *IEEE Visualization*, pp. 75–82, October 2003.

[33] A. E. Lefohn, J. Kniss, C. Hansen, and R. Whitaker, "A streaming narrow-band algorithm: Supplemental information." IEEE Digital Library.

[34] R. Fedkiw, T. Aslam, B. Merriman, and S. Osher, "A non-oscillatory Eulerian approach to interfaces in multimaterial flows (the ghost fluid method)," *Journal of Computational Physics*, vol. 152, pp. 457–492, 1999.

[35] J. Kniss, S. Premoze, C. Hansen, P. Shirley, and A. McPherson, "A model for volume lighting and modeling," *Transactions on Visualization and Computer Graphics*, vol. 9, pp. 150–162, April-June 2003.

[36] The Insight Toolkit `http://www.itk.org`, 2003.

[37] R. Malladi, J. A. Sethian, and B. C. Vemuri, "Shape modeling with front propagation: A level set approach," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 17, pp. 158–175, 1995.

[38] A. E. Lefohn, J. Cates, and R. Whitaker, "Interactive, GPU-based level sets for 3D brain tumor segmentation," in *Medical Image Computing and Computer Assisted Intervention*, pp. 564–572, 2003.

[39] J. Kniss, S. Premoze, M. Ikits, A. E. Lefohn, and C. Hansen, "Gaussian transfer functions for multi-field volume visualization," in *IEEE Visualization*, pp. 497–504, October 2003.

[40] J. Percy and R. Mace, "OpenGL extensions: Siggraph 2003." `http://mirror.ati.com/developer/techpapers.html`, 2003.

[41] R. Whitaker and X. Xue, "Variable-conductance, level-set curvature for image denoising," in *IEEE International Conference on Image Processing*, pp. 142–145, October 2001.

APPENDIX A: DISCRETIZATION OF LEVEL-SET EQUATIONS

This appendix describes the discretization of equation 1 and the curvature computation 3. We discretize equation 1 using the *up-wind* scheme [1] and compute the curvature of the level-set surface using the *difference of normals* method [41].

We begin by describing the finite difference derivatives required for the level-set update and curvature computation. The neighborhood, u, from which these derivatives are computed is specified with the numbering scheme

6	7	8
3	4	5
0	1	2

. (8)

Note that u_4 denotes the center pixel, and $u_i^{\pm z}$ represents the i^{th} sample on the slice above or below the current one. The derivatives of the level-set embedding, ϕ, are then defined as

$$\begin{array}{lcllcl} D_x & = & (u_5-u_3)/2 & D_x^{+z} & = & (u_5^{+z}-u_3^{+z})/2 \\ D_y & = & (u_7-u_1)/2 & D_x^{-z} & = & (u_5^{-z}-u_3^{-z})/2 \\ D_z & = & (u_4^{+z}-u_4^{-z})/2 & D_y^{+x} & = & (u_8-u_2)/2 \\ D_x^+ & = & u_5-u_4 & D_y^{-x} & = & (u_6-u_0)/2 \\ D_y^+ & = & u_7-u_4 & D_y^{+z} & = & (u_7^{+z}-u_1^{+z})/2 \\ D_z^+ & = & u_4^{+z}-u_4 & D_y^{-z} & = & (u_7^{-z}-u_1^{-z})/2 \\ D_x^- & = & u_4-u_3 & D_z^{+x} & = & (u_5^{+z}-u_5^{-z})/2 \\ D_y^- & = & u_4-u_1 & D_z^{-x} & = & (u_3^{+z}-u_3^{-z})/2 \\ D_z^- & = & u_4-u_4^{-z} & D_z^{+y} & = & (u_7^{+z}-u_7^{-z})/2 \\ D_x^{+y} & = & (u_8-u_6)/2 & D_z^{-y} & = & (u_1^{+z}-u_1^{-z})/2. \\ D_x^{-y} & = & (u_2-u_0)/2 & & & \end{array} \quad (9)$$

Curvature is then computed using the above derivatives. The two normals, $\mathbf{n}^+$ and $\mathbf{n}^-$, are computed by

$$\mathbf{n}^+ = \begin{bmatrix} \frac{D_x^+}{\sqrt{(D_x^+)^2+\left(\frac{D_y^{+x}+D_y}{2}\right)^2+\left(\frac{D_z^{+x}+D_z}{2}\right)^2}} \\ \frac{D_y^+}{\sqrt{(D_y^+)^2+\left(\frac{D_x^{+y}+D_x}{2}\right)^2+\left(\frac{D_z^{+y}+D_z}{2}\right)^2}} \\ \frac{D_z^+}{\sqrt{(D_z^+)^2+\left(\frac{D_x^{+z}+D_x}{2}\right)^2+\left(\frac{D_y^{+z}+D_y}{2}\right)^2}} \end{bmatrix} \quad (10)$$

and

$$\mathbf{n}^- = \begin{bmatrix} \frac{D_x^-}{\sqrt{(D_x^-)^2+\left(\frac{D_y^{-x}+D_y}{2}\right)^2+\left(\frac{D_z^{-x}+D_z}{2}\right)^2}} \\ \frac{D_y^-}{\sqrt{(D_y^-)^2+\left(\frac{D_x^{-y}+D_x}{2}\right)^2+\left(\frac{D_z^{-y}+D_z}{2}\right)^2}} \\ \frac{D_z^-}{\sqrt{(D_z^-)^2+\left(\frac{D_x^{-z}+D_x}{2}\right)^2+\left(\frac{D_y^{-z}+D_y}{2}\right)^2}} \end{bmatrix} \quad (11)$$

respectively. The components of the divergence from equation 3 are then computed as

$$\frac{\partial \mathbf{n}_x}{\partial x} = \mathbf{n}_x^+ - \mathbf{n}_x^-, \quad (12)$$

$$\frac{\partial \mathbf{n}_y}{\partial y} = \mathbf{n}_y^+ - \mathbf{n}_y^-, \quad (13)$$

and

$$\frac{\partial \mathbf{n}_z}{\partial z} = \mathbf{n}_z^+ - \mathbf{n}_z^-, \quad (14)$$

Finally, we estimate H with

$$H = \frac{1}{2}\left(\frac{\partial \mathbf{n}_x}{\partial x} + \frac{\partial \mathbf{n}_y}{\partial y} + \frac{\partial \mathbf{n}_z}{\partial z}\right). \quad (15)$$

The upwind approximation to $\nabla\phi$ is then computed using D_x^+, D_y^+, D_z^+, D_x^-, D_y^-, and D_z^-. To begin,

$$\nabla\phi_{\text{max}} = \begin{bmatrix} \sqrt{max(D_x^+,0)^2 + max(-D_x^-,0)^2} \\ \sqrt{max(D_y^+,0)^2 + max(-D_y^-,0)^2} \\ \sqrt{max(D_z^+,0)^2 + max(-D_z^-,0)^2} \end{bmatrix} \quad (16)$$

is computed followed by

$$\nabla\phi_{\text{min}} = \begin{bmatrix} \sqrt{min(D_x^+,0)^2 + min(-D_x^-,0)^2} \\ \sqrt{min(D_y^+,0)^2 + min(-D_y^-,0)^2} \\ \sqrt{min(D_z^+,0)^2 + min(-D_z^-,0)^2} \end{bmatrix}. \quad (17)$$

The final choice of $\nabla\phi$ is defined by

$$\nabla\phi = \begin{cases} \|\nabla\phi_{\text{max}}\|_2 & \text{if } F > 0 \\ \|\nabla\phi_{\text{min}}\|_2 & \text{otherwise} \end{cases}, \quad (18)$$

where F is the linear combination of all speed functions (e.g. mean curvature, the rescaling term G_r, etc). Section VI-A describes the speed terms used in our segmentation application.

The last step in the upwind scheme computes $\phi(t+\triangle t)$ by

$$\phi(t+\triangle t) = \phi(t) + \triangle t F|\nabla\phi|. \quad (19)$$

APPENDIX B: GPU MEMORY ALLOCATION REQUEST GENERATION

This appendix describes the details of the GPU memory allocation/deallocation request scheme used by the GPU virtual memory system. The algorithm is described first in terms of an abstract client solver. Section IV and the B subsection of this appendix describe the client-specific details.

A. General Allocation Request Algorithm

The allocation request algorithm consists of the following steps (see the corresponding steps in Figure 8):

A. GPU computes VPN of requested active pages.
B,C. GPU compresses active-page request.
D. CPU processes memory request.
  1. Reads compressed request image from GPU.
  2. Decodes memory allocation/deallocation requests.
  3. Releases newly deactivated pages.
  4. Allocates/initializes newly activated pages.

Steps A, B, and C create the set of requested active virtual pages. This set serves as the memory allocation/deallocation request to the CPU. The CPU then calls the client's `ReleasePage` function for each newly deallocated page before deallocating the page. Similarly, the CPU calls the client's `InitNewPage` function for each newly activated page.

In Step A, the GPU uses client-specific data to create two RGBA (i.e. 4-tuple) buffers that hold eight *true* or *false* (e.g. 255 or 0) values for each active data element (Figure 8). The first six values represent whether or not the virtual page in each of the six cardinal directions should be active for the next pass. The seventh value indicates if the active page itself should be active, and the eighth value is free to be used by the client. This eight-dimensional, active-page information vector, J, is thus $\mathrm{J} = (+\mathrm{x}, -\mathrm{x}, +\mathrm{y}, -\mathrm{y}, +\mathrm{z}, -\mathrm{z}, \mathrm{self}, \mathrm{clientSpecific})$, where the first six elements refer to relative neighbor offsets in the virtual page space, $\mathrm{V_P}$.

The eight-value code, J, is computed in eight substream passes followed by a single standard (i.e. entire memory page) pass. The substream passes compute whether the in-plane adjacent memory pages need to be active (i.e. the edge-adjacent pages $(+x, -x, +y, -y)$). Each substream pass computes a client-specified function, `IsNeighborActive`, across the page boundary orthogonal to the page edge being rendered and writes the boolean result to the corresponding output component of J. The second computation calls `IsNeighborActive` for the pages above and below the active one. Note, however, that because the neighboring pages are *face-adjacent*, this computation is performed at all data elements in the page instead of just the edges. The computation also writes a *true* value to the J component representing the active page itself if the client's `IsSelfActive` function returns true. The value of the eighth bit is filled by the result of the client's `IsEighthBitTrue` function.

Steps B and C of the allocation-request algorithm compress the two, J buffers into a small (≤ 64kB) active-page message. This compressed message serves as the memory allocation/deallocation request that is sent to the CPU. The compression is accomplished by rendering a quadrilateral of size $\mathrm{S}[\mathrm{G_P}]$ with the *automatic mipmapping* option enabled on the neighbor-information buffers (Step B). The render pass also uses a fragment program designed to create a bit code at each pixel value (Step C). Each pixel in the resulting small image corresponds to a physical memory page. The value of each pixel contains an eight-bit code of the same form as the eight-value code produced in step A (i.e. the J vector). This eight-bit code completely determines if the memory page and/or any of its six cardinal neighbors in virtual page space are to be active on the next pass.

The automatic mipmapping performs a box-filter averaging of the values written in Step A. The result is that if *any* data element in the memory page set a value to *true* in Step A, the down-sampled value will also be true. The fragment program inspects these down-sampled values. It sets the corresponding bit in the output value to true for each non-zero input. The bits are set via an emulated bitwise OR operation. Current fragment processors do not support bitwise operations, but an OR is emulated by conditionally adding power-of-two values to the output value.

In Step D.1, the CPU reads the bit-code message from the GPU. Step D.2 begins by the CPU wrapping the message buffer with a bit-vector accessor. The resulting bit vector is a linear representation of the physical page space, $\mathrm{G_P}$, where each byte represents the information for a page. Two auxiliary bit-vectors are allocated–each a bit-addressed, linear representation of the virtual memory page space, $\mathrm{V_P}$. The first is the `newActiveSet` bit vector, and the second is the client-specific `eighthBitSet` bit vector. After the allocation message is decoded, a true bit in the `newActiveSet` bit vector will denote an active virtual page.

In the next stage of Step D.2, the CPU decodes the bit-vector message. For each 8-bit sequence, the current linear index is converted to a physical page number (PPN). The inverse page table then converts the PPN to a VPN. Because each bit in the bit-code message represents an offset direction from the current virtual page, the decoder can easily reconstruct the VPN for each neighbor of each active page. The decoder then reads the seven spatial page bits. It then computes the VPN for the page represented by each true bit and sets the corresponding bit in the `newActiveSet` bit vector to true. If the eighth bit is true, the `eighthBitSet` is set to true for the corresponding virtual page.

The virtual memory system next determines which virtual memory pages to deallocate and which to allocate (Steps D.3 and D.4). The set of newly deactivated pages is constructed by performing a set-subtraction of the `newActiveSet` from the `oldActiveSet`. The set of pages that need to be allocated for the next pass is created by computing the opposite set difference. Each deallocated memory page is pushed onto a stack of free memory pages. The page table are updated based on the client's implementation of `ReleasePage` function. Each newly activated page is mapped to a physical memory location by popping a page from the free page stack. The physical page is mapped in the page tables and the geometry engine is appropriately updated. The new physical memory is then initialized via the client's `InitNewPage` implementation.

B. Level-Set Solver Implementation Details

For Step A of the update algorithm described in Section IV-D and the preceding subsection, the level-set solver defines the functions `IsNeighborActive` and `IsSelfActive`. The `IsNeighborActive` reads the previously computed, one-side derivative that crosses a page boundary onto a specific neighbor. The function returns true if the derivative is non-zero. The `IsSelfActive` function returns true if *any* of the six, cardinal, one-sided derivatives are non-zero. The level-set solver simply writes the value of the level-set embedding to the eighth data value. This is used to determine if a newly deactivated page is inside or outside of the level-set surface. The `IsEigthBitTrue` function used by the fragment program in Step B returns true if the eighth data value is greater than zero. If a page becomes inactive, it is guaranteed to be either all black or all white. The down-sampled level-set embedding for the page will thus be either pure black or pure white.

The `eighthBitSet` used in the bit-code message decoding stage (Step D.2) is used to determine if a newly deactivated memory page is inside or outside the level-set surface. If the bit for the page is true, then the page is inside the surface. Otherwise it is outside. This information is used by the solver's `ReleasePage` function to map deactivated pages to the correct *static* physical page (white or black). As described in Section IV-A, these static mappings ensure that derivatives across boundaries of the active domain are correct.

The solver's `InitNewPage` function initializes newly allocated physical memory. The memory is initialized to either white or black depending on the inside/outside setting in the page table entry. Note that no level-set data is transferred to accomplish the update. The entire level-set solution resides *only* on the GPU for the duration of the computation. Our current implementation also has to send pre-computed speed pages to the GPU when new pages are added. This could be optimized for many speed functions, however, by computing the function on the GPU.

Session 3

PDE/Level Set Applications

Image Inpainting: An Overview

Guillermo Sapiro
Electrical and Computer Engineering
University of Minnesota
guille@ece.umn.edu
mountains.ece.umn.edu/~guille/inpainting.htm

Overview

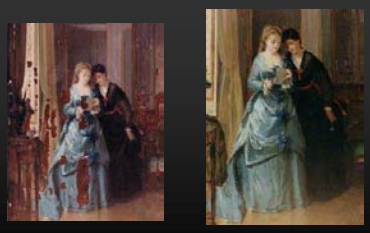
- Goal and background
  - Art, biology, math, and engineering come together
- Related work
- Inpainting
- Filling-in
- Inpainting and image decomposition
- 3D surface filling-in

What is inpainting?

- Modifying an image in a non-detectable form

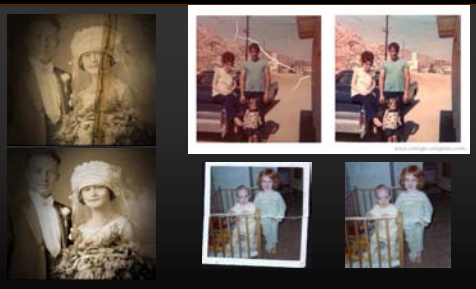
"Cornelia, Mother of the Gracchi" by J. Suvee (Louvre). Emile-Male "The Restorer's Handbook of easel painting".

Another example



From Geary Gallery

Real world example: Photo restoration



- Restorations courtesy of Photo Imaging Studio, Image Enigma, Alleycat Designs

Real world example: Object removal

- From D. King, "The Commissar vanishes".

Real world example: Object removal

- From D. King, “The Commissar vanishes”.

Real world example: Object removal

Lenin and friend Trotsky

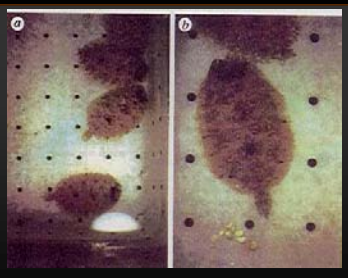
Where is Trotsky?

- From www.newseum.org

Biological inpainting

Biological inpainting

Biological inpainting



From Ramachandran et el.

Human Blind Spot

http://info.med.yale.edu/neurobio/mccormick/fill_in_seminar/figure16.html

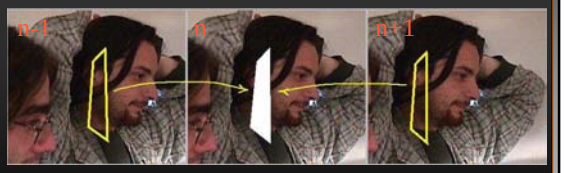
Real world example: Object removal and missing information

- From ProSpec-UK.

The goal

Related work: Films

- e.g. Kokaram et al., Geman et al.



- Doesn't work for stills or static objects

Related work: Texture synthesis

- Hirani, Efros, Heeger, DeBonet, Simoncelli, Zhu, etc.
- Not practical for rich regions
- Not (originally) designed for structured regions
- "Copy" information instead of "see and interpolate"

Related work: Disocclusion

- Masnou-Morel, Nitzberg-Mumford, etc.

- Limitations: Topology, angles

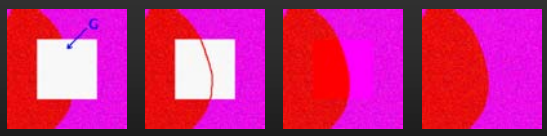
See also Jacobs, Basri, Zucker, etc, and Chan-Shen '00, Zhu-Mumford

Our Contribution

- User only selects region to inpaint
- Rich background and topology not an issue
- Less than 1 minute on a PC

How conservators inpaint

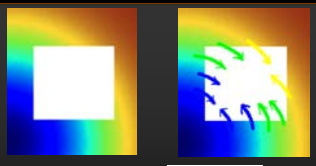
- Minneapolis Institute of Art



Approach 1

Bertalmio, Sapiro, Caselles, Ballester, SIGGRAPH 2000

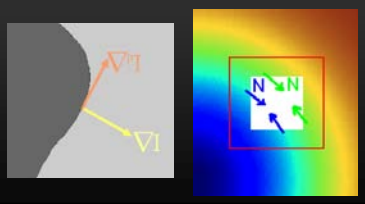
Automatic digital inpainting



- Propagate information $\nabla L \bullet \vec{N} = 0$
- Evolutionary form $\frac{\partial I}{\partial t} = \nabla L \bullet \vec{N}$

Digital inpainting (cont'd)

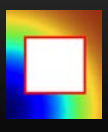
- L = smoothness estimator (Laplacian)
- N = isophote direction (time variant)



The equation

$$\frac{\partial I}{\partial t} = \nabla(\Delta I) \bullet \nabla^{\perp} I$$

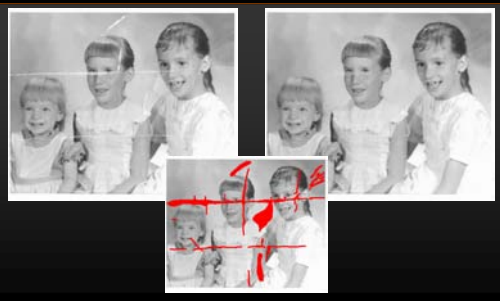
- Plus numerical schemes (Osher-Marquina)
- Boundary conditions
  - Gray values (in a band)
  - Directions (in a band)



Example

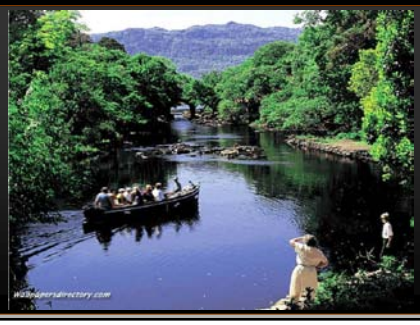
Example: Text removal

Example: Photo restoration

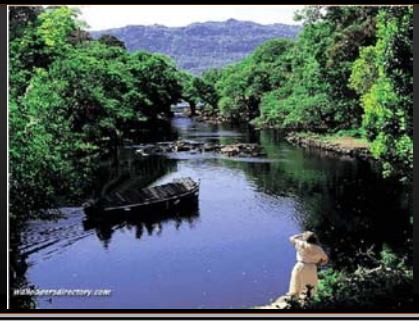


Example: Special effects

Example: Special effects

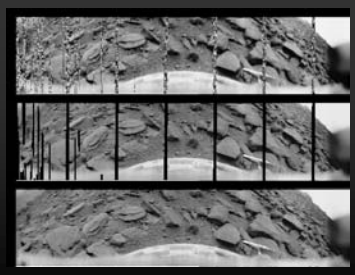


Example: Special effects



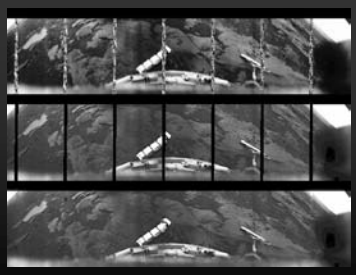
Example: Scratch removal

Russian Venus Mission
Venera 9



From Don Mitchell

Russian Venus Mission
Venera 10

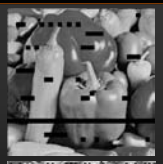


From Don Mitchell

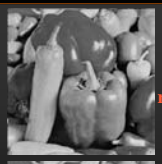
Automatic image inpainting/interpolation for compression and wireless transmission

(Rane-Sapiro-Bertalmio) JPEG and/or JPEG-2000 compatible

Transmitted



Automatic reconstruction



Transmitted

Automatic reconstruction

Automatic image inpainting/interpolation for compression and wireless transmission

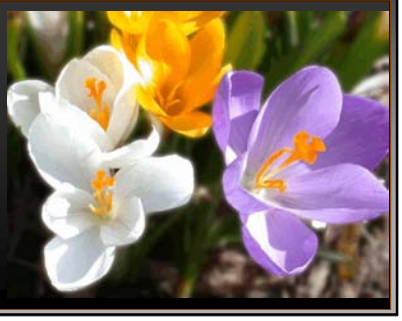
(Rane-Sapiro-Bertalmio) JPEG and/or JPEG-2000 compatible

Transmitted

Automatic image inpainting/interpolation for compression and wireless transmission

(Rane-Sapiro-Bertalmio) JPEG and/or JPEG-2000 compatible

Automatic reconstruction



Approach 1: Concluding remarks

- Technique imitates professionals
- Key concepts
  - Information propagation
  - Both gray values and directions are needed
  - Use a band surrounding the region
- Sharp results
- Low complexity
- Texture is not (yet) reproduced

Concluding remarks (cont.)

- Connected to fluid dynamics (see Bertalmio-Bertozzi-Sapiro CVPR 2001)

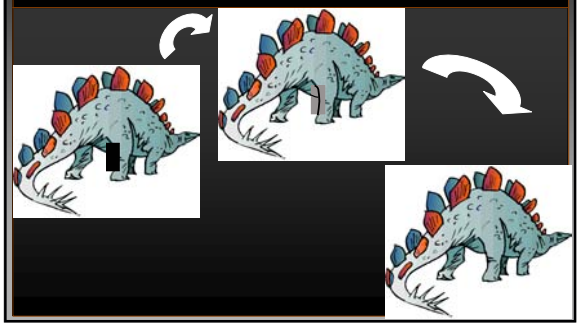
$$\frac{\partial(\Delta I)}{\partial t} = \nabla(\Delta I)\cdot\nabla I^{\perp}$$

- Opens then door to high order PDE's
- Extended to a variational formulation: Approach 2...

Approach 2

C. Ballester, M. Bertalmio, V. Caselles, G. Sapiro, and J. Verdera, IMA Report 2000, IEEE Trans. IP 2001

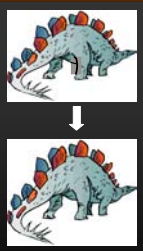
How conservators fill-in
(Minneapolis Institute of Art)



Our approach

- Jointly continue/interpolate level-lines (geometry) and gray values (photometry) in a smooth fashion

Interpolate the gray values given the edges



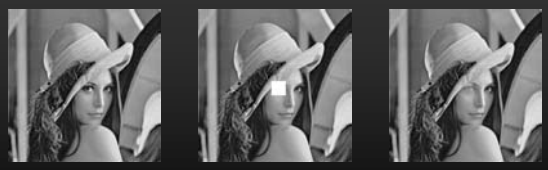
$$\eta = \text{normalized gradient} \Rightarrow \eta \bullet \nabla I = \|\nabla I\|$$

$$\min(I) \int_{\Omega Y Band} |\,\|\nabla I\| - \eta \bullet \nabla I\,|\,d\Omega$$

$$\frac{\partial I}{\partial t} = div\left(\frac{\nabla I}{\|\nabla I\|}\right) - div(\eta)$$

Theorem: The minimizer exists in BV space

Example



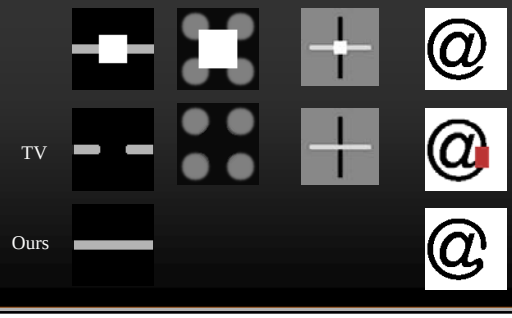
The full functional

$$\min_{I,\vec{n}} \int_{\Omega \in Band} div(\vec{n})^p (a + b\|\nabla G * I\|) + c(\|\nabla I\| - \vec{n} \bullet \nabla I)$$

- Solved via E-L: Coupled 2nd order PDE's
- Implicit discretization used
- Connected to Euler's elastica (Mumford)

- **Theorem:** For p>1 the minimizer exists

Examples



Examples

Examples

Examples

Approach 2: Concluding remarks

- Technique imitates professionals
- Key concepts
  - Information propagation
  - Both gray values and directions are needed
  - Use a band surrounding the region
- Sharp results
- Low complexity
- Texture is not (yet) reproduced

Inpainting and Image Decomposition

Bertalmio, Vese, Sapiro, Osher, July 2002

IEEE Trans. IP, 2003

Basic Idea

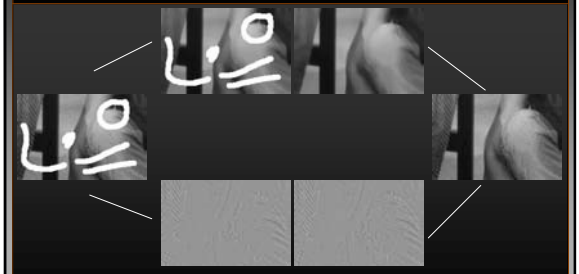


Image decomposition

Definition (Meyer) G = Banach space of generalized functions $v(x,y)$:

$$v(x,y) = \partial_x g_1(x,y) + \partial_y g_2(x,y),\ g_1, g_2 \in L^\infty(R^2),$$

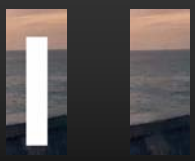
$$g := (g_1, g_2), |g(x,y)| = \sqrt{g_1(x,y)^2 + g_2(x,y)^2},$$

and use the infimum over all possible decompositions.

Definition (Vese-Osher) The image I is decomposed in *structure* (u) and *texture* (v) solving via gradient descent the variational problem

$$\inf_{u,g_1,g_2} \left\{ G_p(u,g_1,g_2) = \int |\nabla u| + \lambda \int |I - u - \partial_x g_1 - \partial_y g_2|^2 dxdy + \mu \left[\int \left(\sqrt{g_1^2 + g_2^2} \right)^p dxdy \right]^{\frac{1}{p}} \right\}$$

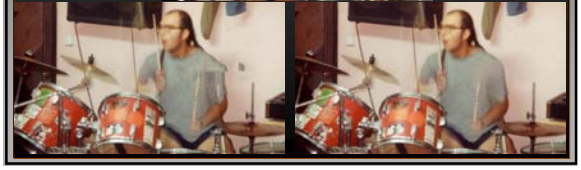
Example

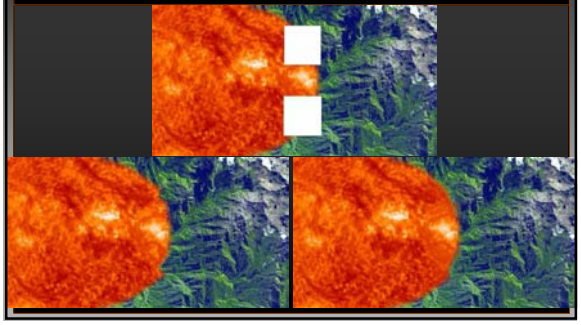


Example

Texture only

Ours

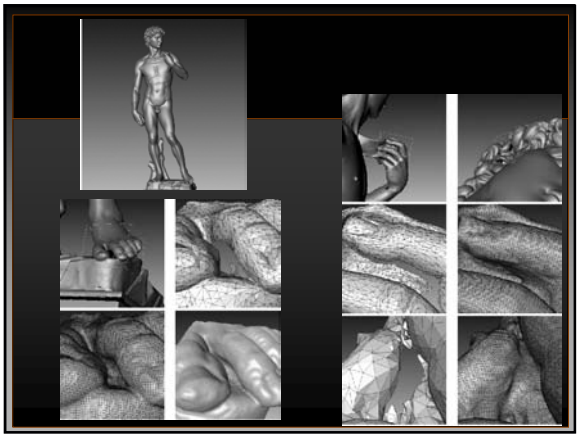




Filling surface holes

Verdera, Bertalmio, Caselles, Sapiro,
IEEE ICIP 2003

Data and inspiration from Levoy and the Michelangelo Project



Inpainting from Sensor Arrays

Yatziv, Sapiro, Levoy

\

Conclusion

- Inpainting 2D and 3D via PDEs (flows)
- Inpainting in a decomposition space
- Connections with biology and art

- See also recent works such as Tensor Voting (CVPR'03), Edge directed Efros (CVPR'03), Global inpainting (ICCV'03).

Acknowledgments

- Institute Henri Poincare in Paris, France.
- T. Robbins, E. Buschor, S. Betelú, S. Osher, E. Simoncelli, A. Bertozzi, T. Chan, C. Kenney, P. L. Lions, J. M. Morel, J. Shen.
- Supported by ONR-Math, ONR Young Investigator Award, Presidential Early Career Awards for Scientists and Engineers, NSF CAREER Award, NSF-LIS, Europen project on viscosity solutions, and IIE-Uruguay.

The end

Thank you

The Art of Distance and Geodesic Computations

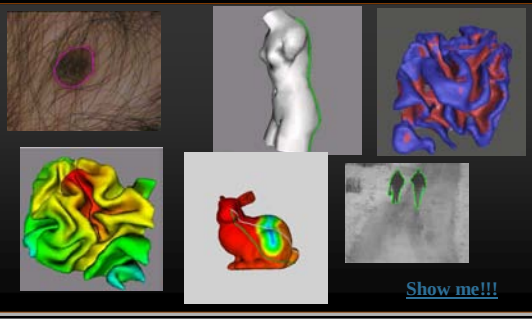
Guillermo Sapiro
Electrical and Computer Engineering
University of Minnesota
guille@ece.umn.edu

Supported by NSF, ONR, PECASE, CAREER, NIH

Overview

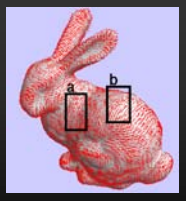
- Motivation
- Background on fast/accurate geodesic computations
- Distance functions and geodesics on implicit hyper-surfaces
- Unorganized points
- Generalized geodesics
- The future and concluding remarks

Motivation: A Few Examples

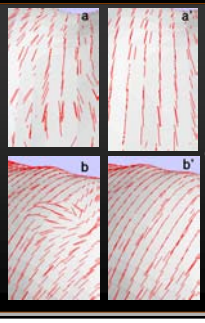


Show me!!!

Motivation: A Few Examples (cont.)



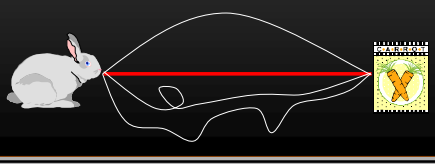
noisy



cleaned

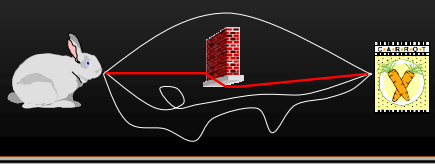
Motivation: What is a Geodesic?

$$d_S^g(p,x) = \inf_C \int_p^x g(C)ds$$

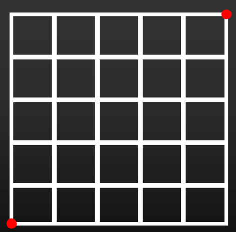


Motivation: What is a Geodesic?

$$d_S^g(p,x) = \inf_C \int_p^x g(C)ds$$

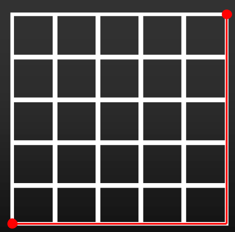


Background: Distance and Geodesic Computation via Dijkstra



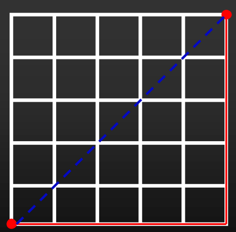
- **Complexity: O(n log n)**
- **Advantage: Works in any dimension and with any geometry (graphs)**
- **Problems:**
  - Not consistent
  - Unorganized points?
  - Noise?
  - Implicit surfaces?

Background: Distance and Geodesic Computation via Dijkstra



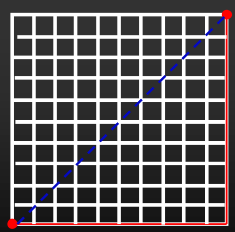
- **Complexity: O(n log n)**
- **Advantage: Works in any dimension and with any geometry (graphs)**
- **Problems:**
  - Not consistent
  - Unorganized points?
  - Noise?
  - Implicit surfaces?

Background: Distance and Geodesic Computation via Dijkstra



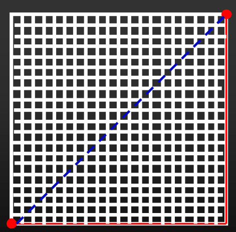
- **Complexity: O(n log n)**
- **Advantage: Works in any dimension and with any geometry (graphs)**
- **Problems:**
  - Not consistent
  - Unorganized points?
  - Noise?
  - Implicit surfaces?

Background: Distance and Geodesic Computation via Dijkstra



- **Complexity: O(n log n)**
- **Advantage: Works in any dimension and with any geometry (graphs)**
- **Problems:**
  - Not consistent
  - Unorganized points?
  - Noise?
  - Implicit surfaces?

Background: Distance and Geodesic Computation via Dijkstra



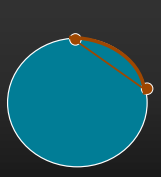
- **Complexity: O(n log n)**
- **Advantage: Works in any dimension and with any geometry (graphs)**
- **Problems:**
  - Not consistent
  - Unorganized points?
  - Noise?
  - Implicit surfaces?

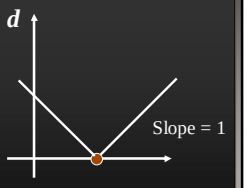
Background: Distance Functions as Hamilton-Jacobi Equations

- ***g* = weight on the hyper-surface**
- **The *g*-weighted distance function between two points *p* and *x* on the hyper-surface *S* is:**

$$\left\| \nabla_S d_S^g(p,x) \right\| = g$$

$$\left\|\nabla_S d_S^g(p,x)\right\| = g$$



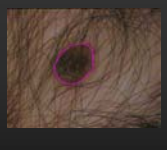


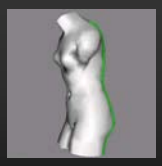
Background: Computing Distance Functions as Hamilton-Jacobi Equations

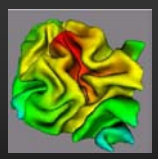
- Solved in O(n log n) by Tsitsiklis, by Sethian, and by Helmsen, only for Euclidean spaces and Cartesian grids

$$\left\|\nabla d^g(p,x)\right\| = g$$

- Solved only for acute 3D triangulations by Kimmel and Sethian





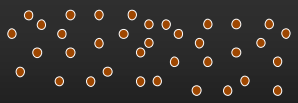


A real time example

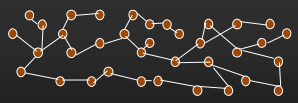
The Problem

- How to compute intrinsic distances and geodesics for
  - General dimensions
  - Implicit surfaces
  - Unorganized noisy points (hyper-surfaces just given by examples)

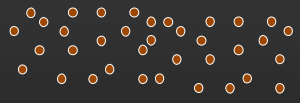
Intermezzo: Tenenbaum, de Silva, et al...



Intermezzo: Tenenbaum, de Silva, et al...



Intermezzo: Tenenbaum, de Silva, et al...



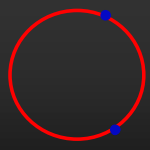
- **Problems:**
  - Doesn't address noisy examples/measurements
  - Restriction on sampling density and manifolds
  - Uses Dijkestra (back to non consistency)
  - Doesn't work for implicit surface representations

Our Approach

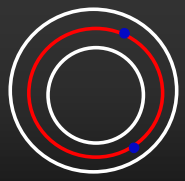
- We have to solve

$$\left\| \nabla_S d_S^g(p,x) \right\| = g$$

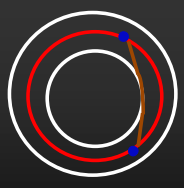
Basic Idea



Basic Idea



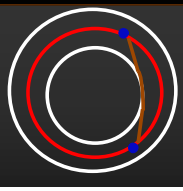
Basic Idea



Theorem (Memoli-Sapiro):

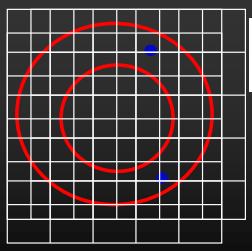
$$\left| d^g - d_S^g \right| \to 0$$

Basic idea



$$\left| d^g - d_S^g \right| \to \begin{cases} h^{1/2} & \text{general} \\ h & \text{local analytic} \\ h^\gamma, \gamma > 1 & \text{"smart" metric} \end{cases}$$

Why is this good?



$$\left\|\nabla_S d_S^g(p,x)\right\| = g$$

$$\Downarrow$$

$$\left\|\nabla d^{\,g}(p,x)\right\| = g$$

Implicit Form Representation

$S = \text{level} - \text{set of } \Psi : R^n \rightarrow R = \{x : \Psi(x) = 0\}$

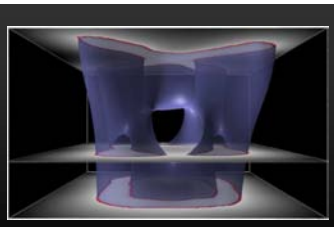


Figure from G. Turk

Data extension

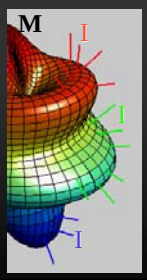
$I : M \rightarrow R$

- Embed M:

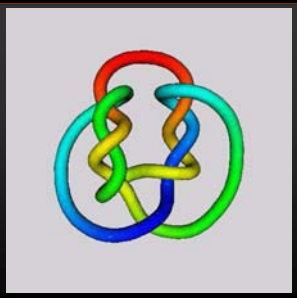
$$M = \{x : \Psi(x) = 0\}$$

- Extend I outside M:

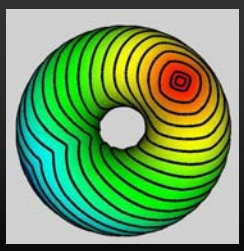
$$\frac{\partial I}{\partial t} + sign(\Psi)\,(\nabla I \cdot \nabla \Psi) = 0$$



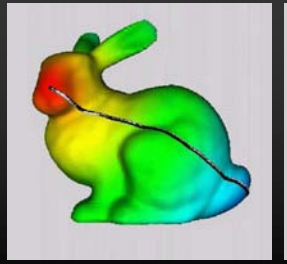
Examples

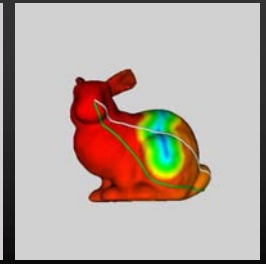


Examples

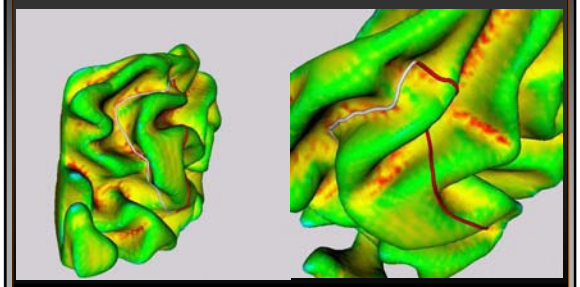


Examples



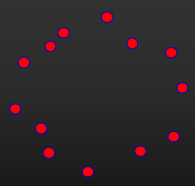


Examples

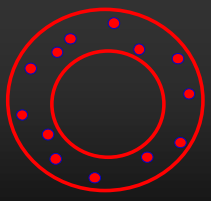


Examples

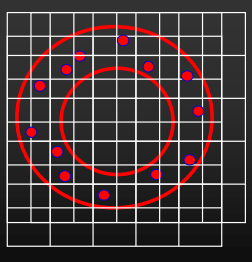
Unorganized points

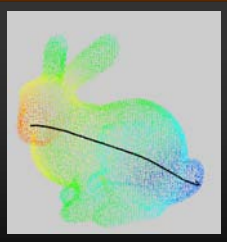


Unorganized points (cont.)

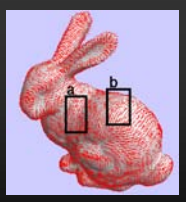


Unorganized points

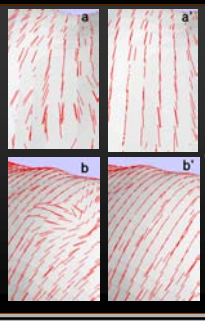




Is this a geodesic?



noisy



cleaned

Generalized geodesics: Harmonic maps

- Find a smooth map from two manifolds *(M,g)* and *(N,h)* such that

$$\min_{C:M\to N} \int_{\Omega} \left\| \nabla_M C \right\|^p \, dvol_M$$

$$\left(\frac{\partial C}{\partial t} = \right) \qquad \Delta_M C + A_N(C) < \nabla_M C, \nabla_M C >= 0$$

Examples

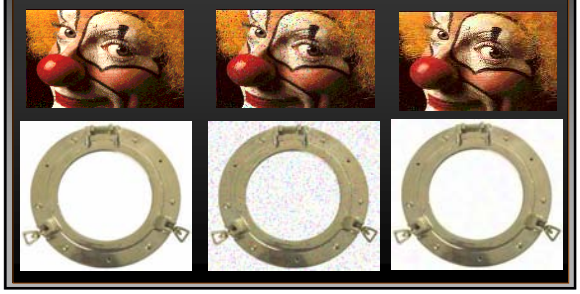
- M is an Euclidean space and N the real line

$$\Delta C = 0$$

- M = [0,1], geodesics!

$$\frac{\partial^2 C}{\partial t^2} + A_N(C) < \nabla_M C, \nabla_M C >= 0$$

Color Image Enhancement

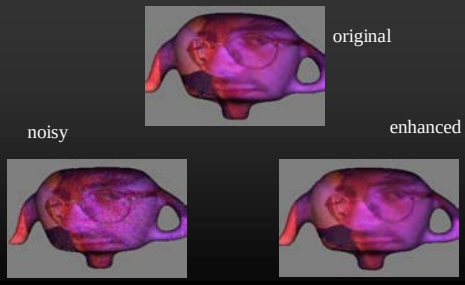


Implicit surfaces

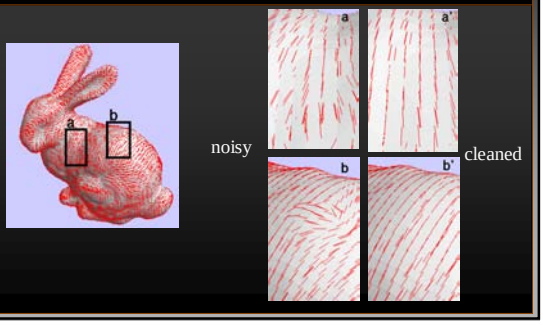
- Domain and target are implicitly represented: Simple Cartesian numerics

$$\frac{\partial C}{\partial t} = \mathrm{div}(P_{\nabla\Phi} \nabla C) + \left(\sum_k H_\Phi \left\langle \frac{\partial C}{\partial x_k}, \frac{\partial C}{\partial x_k} \right\rangle \right) \|\nabla \Phi\|$$

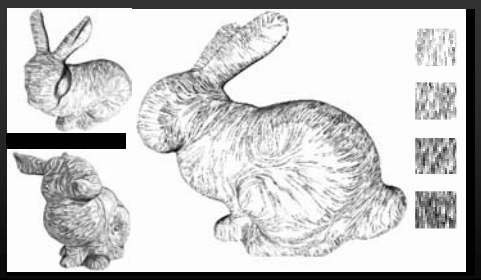
Example: Chroma denoising on a surface



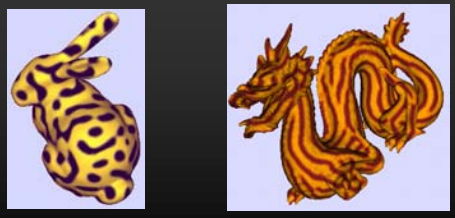
Example: Direction denoising



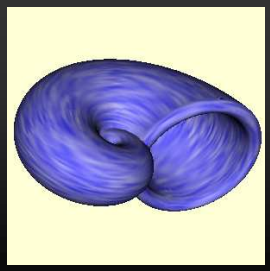
Application (with G. Gorla and V. Interrante)



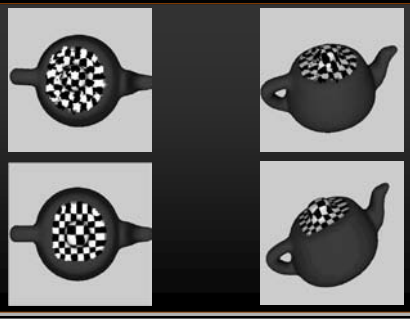
Examples



Vector field visualization (e.g., principal directions)



Texture mapping denoising



Texture mapping denoising

Concluding remarks

- A general computational framework for distance functions, geodesics, and generalized geodesics
- Implicit hyper-surfaces and un-organized points

Thanks

F. Memoli and G. Sapiro, "Fast computation of weighted distance functions and geodesics on implicit hyper-surfaces," *Journal of Computational Physics* 173:2, pp. 730-764, November 2001.

B. Tang, G. Sapiro, and V. Caselles, "Color image enhancement via chromaticity diffusion," *IEEE Trans. Image Processing* 10, pp. 701-707, May 2001.

B. Tang, G. Sapiro, and V. Caselles, "Diffusion of general data on non-flat manifolds via harmonic maps theory: The direction diffusion case," *Int. Journal Computer Vision* 36:2, pp. 149-161, February 2000.

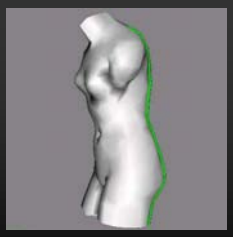
M. Bertalmio, L. T. Cheng, S. Osher, and G. Sapiro, "Variational problems and partial differential equations on implicit surfaces," *Journal of Computational Physics* 174:2, pp. 759-780, 2001.

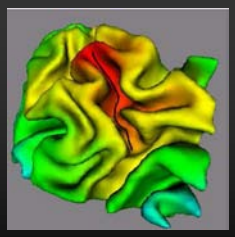
A. Bartesaghi and G. Sapiro, "A system for the generation of curves on 3D brain images," *Human Brain Mapping* 14:1, pp. 1-15, 2001.

F. Memoli, G. Sapiro, and S. Osher, "PDEs and variational problems into general manifolds," *Journal of Computational Physics,* 2004.

Sulcii extraction on meshes

(with A. Bartesaghi)





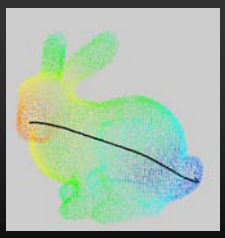
Follows Kimmel-Sethian

Distance Functions and Geodesics on Point Clouds

Guillermo Sapiro
Electrical and Computer Engineering
University of Minnesota
guille@ece.umn.edu

Joint work with F. Memoli (PhD Thesis)

Goal and motivation

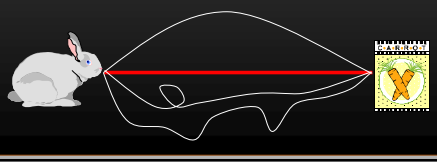


Outline

- Background on fast/accurate geodesic computations
- Distance functions and geodesics on implicit hyper-surfaces
- Point clouds
- The future and concluding remarks

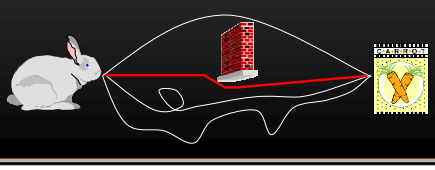
Motivation: What is a Geodesic?

$$d_S^g(p,x) = \inf_C \int_p^x g(C)ds$$

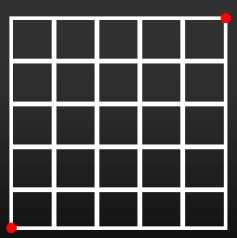


Motivation: What is a Geodesic?

$$d_S^g(p,x) = \inf_C \int_p^x g(C)ds$$

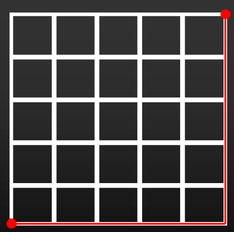


Background: Distance and Geodesic Computation via Dijkstra



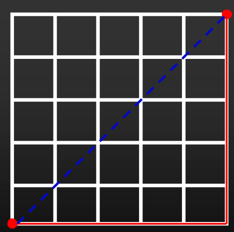
- Complexity: O(n log n)
- Advantage: Works in any dimension and with any geometry (graphs)
- Problems:
  - Not consistent
  - Unorganized point clouds?
  - Noise?
  - Implicit surfaces?

Background: Distance and Geodesic Computation via Dijkstra



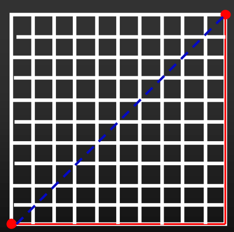
- **Complexity:** O(n log n)
- **Advantage:** Works in any dimension and with any geometry (graphs)
- **Problems:**
  - Not consistent
  - Unorganized point clouds?
  - Noise?
  - Implicit surfaces?

Background: Distance and Geodesic Computation via Dijkstra



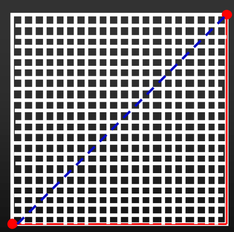
- **Complexity:** O(n log n)
- **Advantage:** Works in any dimension and with any geometry (graphs)
- **Problems:**
  - Not consistent
  - Unorganized point clouds?
  - Noise?
  - Implicit surfaces?

Background: Distance and Geodesic Computation via Dijkstra



- **Complexity:** O(n log n)
- **Advantage:** Works in any dimension and with any geometry (graphs)
- **Problems:**
  - Not consistent
  - Unorganized point clouds?
  - Noise?
  - Implicit surfaces?

Background: Distance and Geodesic Computation via Dijkstra



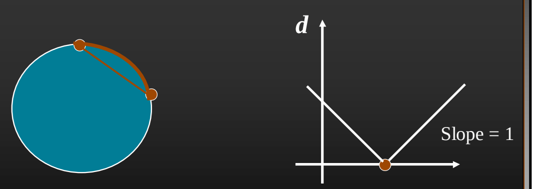
- **Complexity:** O(n log n)
- **Advantage:** Works in any dimension and with any geometry (graphs)
- **Problems:**
  - Not consistent
  - Unorganized point clouds?
  - Noise?
  - Implicit surfaces?

Background: Distance Functions as Hamilton-Jacobi Equations

- *g* = weight on the hyper-surface
- The *g*-weighted distance function between two points *p* and *x* on the hyper-surface *S* is:

$$\left\| \nabla_S d_S^g(p,x) \right\| = g$$

$$\left\| \nabla_S d_S^g(p,x) \right\| = g$$



Background: Computing Distance Functions as Hamilton-Jacobi Equations

- Solved in O(n log n) by Tsitsiklis, by Sethian, and by Helmsen, only for Euclidean spaces and Cartesian grids

$$\left\|\nabla d^{g}(p,x)\right\| = g$$

- Solved for acute 3D triangulations by Kimmel and Sethian

The Problem

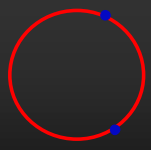
- How to compute intrinsic distances and geodesics for
  - General dimensions
  - Implicit surfaces
  - Unorganized noisy points (hyper-surfaces just given by samples/examples)

Our Approach: The Geometry

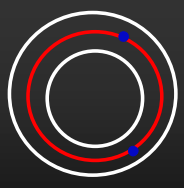
- We have to solve

$$\left\|\nabla_S d_S^{g}(p,x)\right\| = g$$

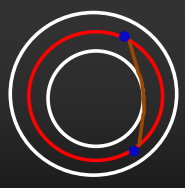
Basic Idea



Basic Idea



Basic Idea



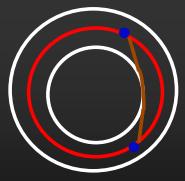
The result

Theorem (Memoli - S. 2001):

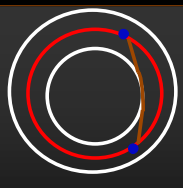
(open/closed -- any co-dimension)

$$\left| d^g - d_S^g \right| \to 0$$

$$\left| \gamma^g - \gamma_S^g \right| \to 0$$

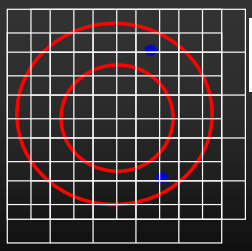


Rate of convergence



$$\left| d^g - d_S^g \right| \to \begin{cases} h^{1/2} & \text{general} \\ h & \text{local analytic} \\ h^\gamma, \gamma > 1 & \text{"smart" metric} \end{cases}$$

Why is this good?

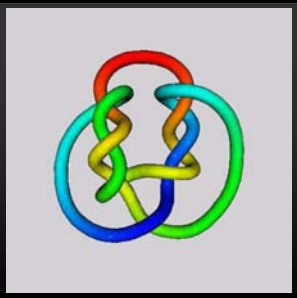


$$\left\| \nabla_S d_S^g(p,x) \right\| = g$$

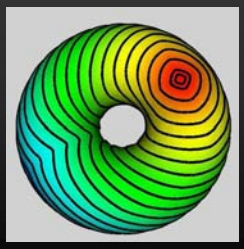
$$\Downarrow$$

$$\left\| \nabla d^g(p,x) \right\| = g$$

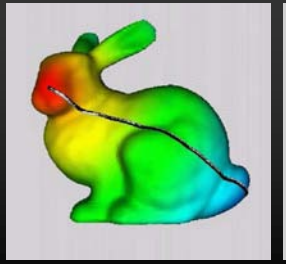
Examples

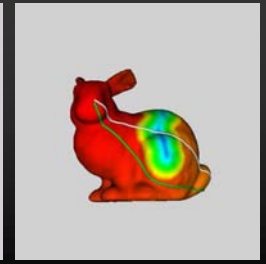


Examples

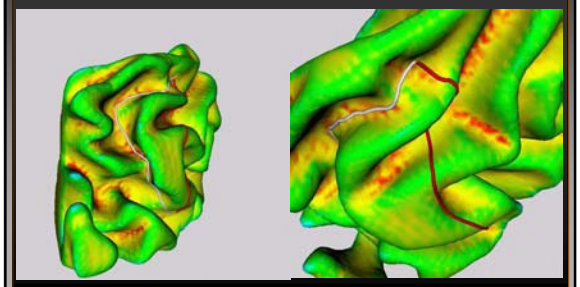


Examples

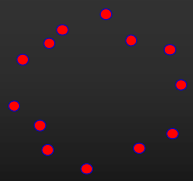




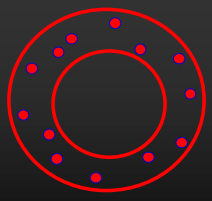
Examples



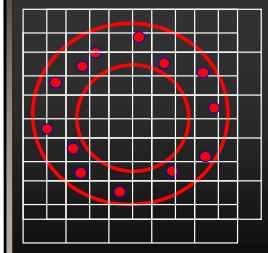
Unorganized point clouds

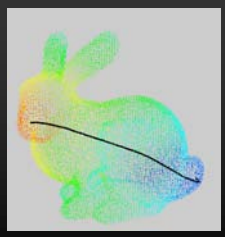


Unorganized points (cont.)



Unorganized points





Randomly sampled manifolds (with noise)

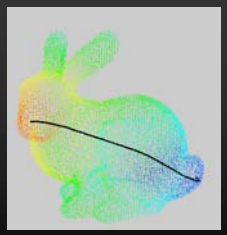
Theorem (Memoli - S. 2002):

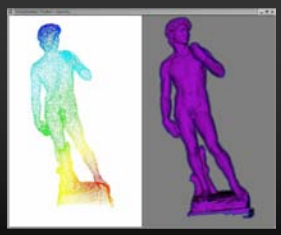
$$\max_{p,q\in\mathcal{S}}\left(d_{\mathcal{S}}(p,q)-d_{\Omega^h_{\mathcal{P}_{n(h)}(h)}}(p,q)\right)\leq C_{\mathcal{S}}\sqrt{h}$$

$$P(\max_{p,q\in calS}\left(d_{\mathcal{S}}(p,q)-d_{\Omega^h_{\mathcal{P}_n}}(p,q)\right)>\varepsilon)\overset{n\uparrow\infty}{\longrightarrow}0$$

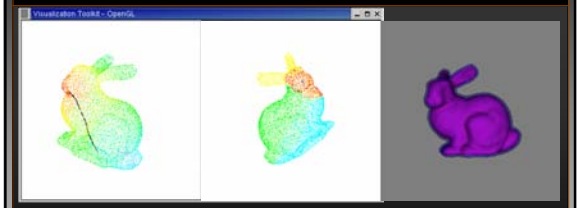
$$\lim_{h,n}P(d_{\mathcal{H}}(\mathcal{S},\Omega^h_{\mathcal{P}_n})>\varepsilon)=0$$

Examples (VRML)

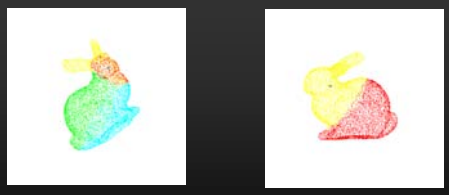




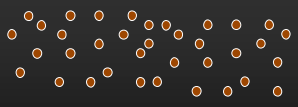
Examples

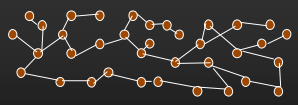


Intrinsic Voronoi of Point Clouds



Intermezzo: de Silva, Tenenbaum, et al...

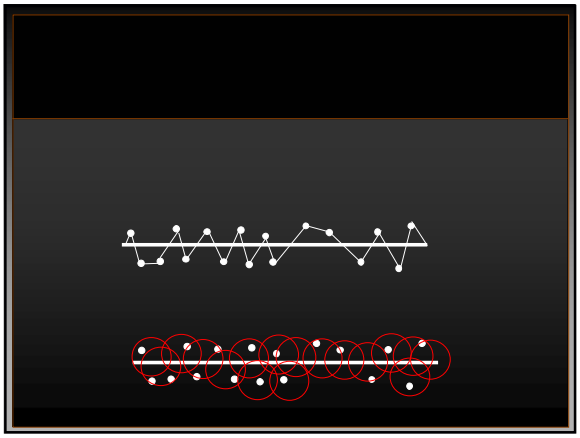




Intermezzo: Tenenbaum, de Silva, et al...

- **Main Problem:**
  - Doesn't address noisy examples/measurements: Much less robust to noise!

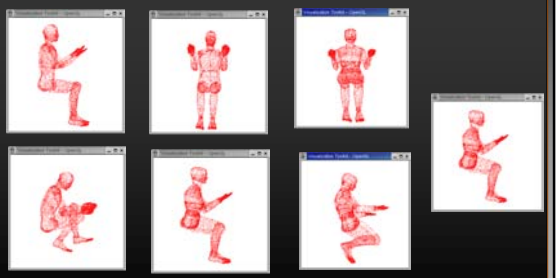
Error increases with the number of samples!



Intermezzo: de Silva, Tenenbaum, et al...

- **Problems:**
  - Doesn't address noisy examples/measurements: Much less robust to noise!
  - Only convex surfaces
  - Uses Dijkestra (back to non consistency)
  - Doesn't work for implicit surface representations

Comparing Point Clouds



Comparing Point Cloud

MODEL	Man2	Man3	Man5	Woman2	Woman3
Man2	*	0.0514	0.0570	0.4690	0.4853
Man3	*	*	0.0206	0.4701	0.4859
Man5	*	*	*	0.4702	0.4862
Woman2	*	*	*	*	0.2639
Woman3	*	*	*	*	*

Concluding remarks

- A general computational framework for distance functions and geodesics
- Implicit hyper-surfaces and points clouds
- See also …

Thanks

F. Memoli and G. Sapiro, "Fast computation of weighted distance functions and geodesics on implicit hyper-surfaces," *Journal of Computational Physics* 173:2, pp. 730-764, November 2001.

F. Memoli and G. Sapiro, "Distance functions and geodesics on point clouds," *IMA TR (www.ima.umn.edu)*, Dec. 2002; May 2003.

Level Set Surface Editing Operators

Ken Museth* David E. Breen* Ross T. Whitaker† Alan H. Barr*

*Computer Science Department
California Institute of Technology

†School of Computing
University of Utah

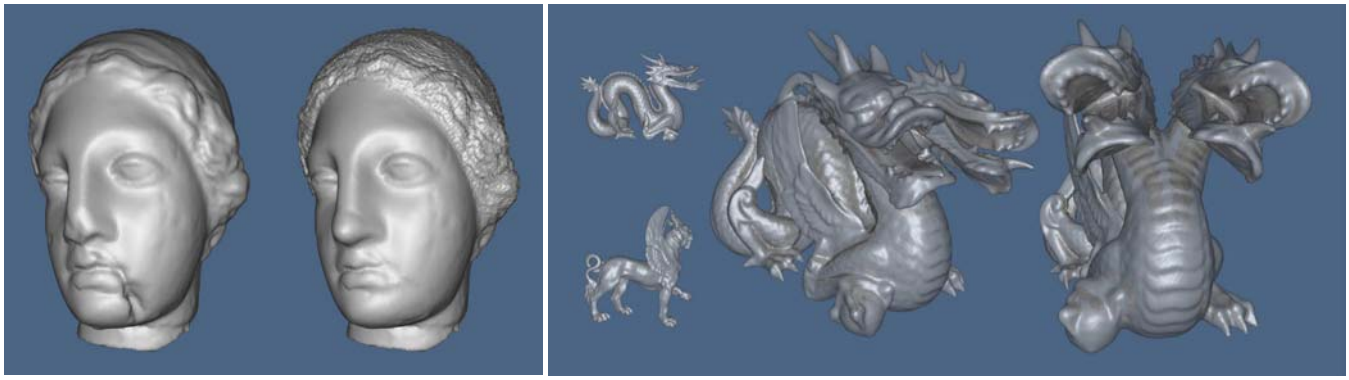


Figure 1: Surfaces edited with level set operators. Left: A damaged Greek bust model is repaired with a new nose, chin and sharpened hair. Right: An edited model is constructed from models of a griffin and dragon (small figures), producing a two-headed, winged dragon.

Abstract

We present a level set framework for implementing editing operators for surfaces. Level set models are deformable implicit surfaces where the deformation of the surface is controlled by a speed function in the level set partial differential equation. In this paper we define a collection of speed functions that produce a set of surface editing operators. The speed functions describe the velocity at each point on the evolving surface in the direction of the surface normal. All of the information needed to deform a surface is encapsulated in the speed function, providing a simple, unified computational framework. The user combines pre-defined building blocks to create the desired speed function. The surface editing operators are quickly computed and may be applied both regionally and globally. The level set framework offers several advantages. 1) By construction, self-intersection cannot occur, which guarantees the generation of physically-realizable, simple, closed surfaces. 2) Level set models easily change topological genus, and 3) are free of the edge connectivity and mesh quality problems associated with mesh models. We present five examples of surface editing operators: blending, smoothing, sharpening, openings/closings and embossing. We demonstrate their effectiveness on several scanned objects and scan-converted models.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Surface and object representations; I.3.4 [Computer Graphics]: Graphics Utilities—Graphics Editors;

Keywords: Deformations, geometric modeling, implicit surfaces, shape blending.

1 Introduction

The creation of complex models for such applications as movie special effects, graphic arts, and computer-aided design can be a time-consuming, tedious, and error-prone process. One of the solutions to the model creation problem is 3D photography [Bouguet and Perona 1999], i.e. scanning a 3D object directly into a digital representation. However, the scanned model is rarely in a final desired form. The scanning process is imperfect and introduces errors and artifacts, or the object itself may be flawed.

3D scans can be converted to polygonal and parametric surface meshes [Edelsbrunner and Mücke 1994; Bajaj et al. 1995; Amenta et al. 1998]. Many algorithms and systems for editing these polygonal and parametric surfaces have been developed [Cohen et al. 2001], but surface mesh editing has its limitations and must address several difficult issues. For example, it is difficult to guarantee that a mesh model will not self-intersect when performing a local editing operation based on the movement of vertices or control points, producing non-physical, invalid results. See Figure 2. If self-intersection occurs, it must be fixed as a post-process. Also, when merging two mesh models the process of clipping individual polygons and patches may produce errors when the elements are small and/or thin, or if the elements are almost parallel. In addition while it is not impossible to change the genus of a surface mesh model [Biermann et al. 2001], it is certainly difficult and requires significant effort to maintain the consistency/validity of the underlying vertex/edge connectivity structure.

1.1 New Surface Editing Operators

In order to overcome these difficulties we present a level set approach to implementing operators for locally and globally editing closed surfaces. Level set models are deformable implicit surfaces that have a volumetric representation [Osher and Sethian 1988]. They are defined as an iso-surface, i.e. a level set, of some implicit function ϕ. The surface is deformed by solving a partial differential equation (PDE) on a regular sampling of ϕ, i.e. a volume dataset. To date level set methods have not been developed for adaptive grids, a limitation of current implementations, but not of the mathematics. It should be emphasized that level set methods do

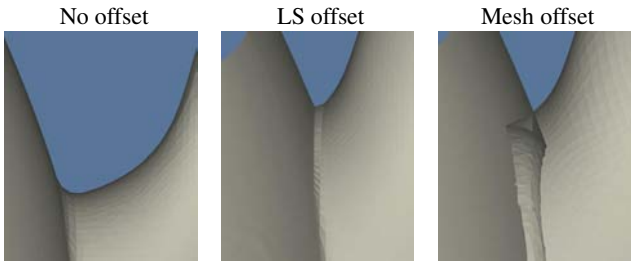


Figure 2: (left) A cross-section of the teapot model near the spout. (middle) No self-intersection occurs, by construction, when performing a level set (LS) offset, *i.e.* dilation, of the surface. (right) Self-intersections may occur when offsetting a mesh model.

not manipulate an explicit closed form representation of ϕ, but only a sampling of it. Level set methods provide the techniques needed to change the voxel values of the volume in a way that deforms the embedded iso-surface to meet a user-defined goal. The user controls the deformation of the level set surface by defining a speed function $\mathcal{F}(\boldsymbol{x}, \ldots)$, the speed of the level set at point $\boldsymbol{x}$ in the direction of the normal to the surface at $\boldsymbol{x}$. Therefore all the information needed to deform a level set model may be encapsulated in a single speed function $\mathcal{F}()$, providing a simple, unified computational framework.

We have developed a number of surface editing operators within the level set framework by defining a collection of new level set speed functions. The cut-and-paste operator (Section 5.1) gives the user the ability to copy, remove and merge level set models (using volumetric CSG operations) and automatically blends the intersection regions (See Section 5.2). Our smoothing operator allows a user to define a region of interest and smooths the enclosed surface to a user-defined curvature value. See Section 5.3. We have also developed a point-attraction operator. See Section 5.4. Here, a regionally constrained portion of a level set surface is attracted to a single point. By defining line segments, curves, polygons, patches and 3D objects as densely sampled point sets, the single point attraction operator may be combined to produce a more general surface embossing operator. As noted by others, the opening and closing morphological operators may be implemented in a level set framework [Sapiro et al. 1993; Maragos 1996]. We have also found them useful for performing global blending (closing) and smoothing (opening) on level set models. Since all of the operators accept and produce the same volumetric representation of closed surfaces, the operators may be applied repeatedly to produce a series of surface editing operations. See Figure 11.

1.2 Benefits and Issues

Performing surface editing operations within a level set framework provides several advantages and benefits. Many types of surfaces may be imported into the framework as a distance volume, a volume dataset that stores the signed shortest distance to the surface at each voxel. This allows a number of different types of surfaces to be modified with a single, powerful procedure. By construction, the framework always produces non-self-intersecting surfaces that represent physically-realizable objects, an important issue in computer-aided design. Level set models easily change topological genus, and are free of the edge connectivity and mesh quality problems associated with deforming and modifying mesh models. Additionally, some reconstruction algorithms produce volumetric models [Curless and Levoy 1996; Whitaker 1998; Zhao et al. 2001] and volumetric scanning systems are increasingly being employed in a number of diverse fields. Therefore volumetric models are becoming more prevalent and there is a need to develop powerful editing operators that act on these types of models directly.

There are implementation issues to be addressed when using level set models. Given their volumetric representation, one may be concerned about the amount of computation time and memory needed to process level set models. Techniques have been developed to limit level set computations to only a narrow band around the level set of interest [Adalsteinsson and Sethian 1995; Whitaker 1998; Peng et al. 1999] making the computational complexity proportional to the surface area of the model. We have also developed computational techniques that allow us to perform the narrow band calculations only in a portion of the volume where the level set is actually moving. Additionally, fast marching methods have been developed to rapidly evaluate the level set equation under certain circumstances [Tsitsiklis 1995; Sethian 1996]. Memory usage has not been an issue when generating the results in this paper. The memory needed for our results (512 MB) is available on standard workstations and PCs. We have implemented our operators in an interactive environment that allows us to easily edit a number of complex surfaces. Additionally, concerns have been raised that volume-based models cannot represent fine or sharp features. Recent advances [Frisken et al. 2000; Kobbelt et al. 2001] have shown that is is possible to model these kinds of structures with volume datasets, without excessively sampling the whole volume. These advances will also be available for our operators once adaptive level set methods, an active research area, are developed.

1.3 Contributions

The major contributions of our work are the following.

- The introduction of a unified approach to surface editing within a level set framework.
  - Editing operators defined by speed functions.
  - Results produced by solving a PDE.
- The definition of level set speed functions that implement blending, smoothing and embossing surface editing operators.
  - Blending is automatic and is constrained to only occur within a user-specified distance to an arbitrarily complex intersection curve.
  - Smoothing and embossing are constrained to occur within a user-specified region.
  - The user specifies the local geometric properties of the resulting surface modifications.
  - The user specifies if material should be added and/or removed during editing operations.
- The new techniques used to localize level set calculations.
- In Appendix B we present a new, numerically-stable curvature measure for level set surfaces.

2 Previous Work

Three areas of research are closely related to our level set surface editing work; volumetric sculpting, mesh-based surface editing/fairing and implicit modeling. Volumetric sculpting provides methods for directly manipulating the voxels of a volumetric model. CSG Boolean operations [Hoffmann 1989; Wang and Kaufman 1994] are commonly found in volume sculpting systems, providing a straightforward way to create complex solid objects by combining simpler primitives. One of the first volume sculpting systems is presented in [Galyean and Hughes 1991]. [Wang and Kaufman 1995] improved on this work by introducing tools for carving and sawing. More recently [Perry and Frisken 2001] implemented a volumetric sculpting system based on the Adaptive Distance Fields (ADF) [Frisken et al. 2000], allowing for models with adaptive resolution.

Performing CSG operations on mesh models is a long-standing area of research [Requicha and Voelcker 1985; Laidlaw et al. 1986]. Recently CSG operations were developed for multi-resolution subdivision surfaces by [Biermann et al. 2001], but this work did not address the problem of blending or smoothing the sharp features often produced by the operations. However, the smoothing of meshes has been studied on several occasions [Welch and Witkin 1994; Taubin 1995; Kobbelt et al. 1998]. [Desbrun et al. 1999] have developed a method for fairing irregular meshes using diffusion and curvature flow, demonstrating that mean-curvature based flow produces the best results for smoothing.

There exists a large body of surface editing work based on implicit models [Bloomenthal et al. 1997]. This approach uses implicit surface representations of analytic primitives or skeletal offsets. The implicit modeling work most closely related to ours is found in [Wyvill et al. 1999]. They describe techniques for performing blending, warping and boolean operations on skeletal implicit surfaces. [Desbrun and Gascuel 1995] address the converse problem of preventing unwanted blending between implicit primitives, as well as maintaining a constant volume during deformation.

Level set methods have been successfully applied in computer graphics, computer vision and visualization [Sethian 1999; Sapiro 2001], for example medical image segmentation [Malladi et al. 1995; Whitaker et al. 2001], shape morphing [Breen and Whitaker 2001], 3D reconstruction [Whitaker 1998; Zhao et al. 2001], and recently for the animation of liquids [Foster and Fedkiw 2001]

Our work stands apart from previous work in several ways. We have not developed volumetric modeling tools. Our editing operators act on surfaces that happen to have an underlying volumetric representation, but are based on the mathematics of deforming implicit surfaces. Our editing operators share several of the capabilities of mesh-based tools, but are not hampered by the difficulties of maintaining vertex/edge information. Since level set models are not tied to any specific implicit basis functions, they easily represent complex models to within the resolution of the sampling. Our work is the first to utilize level set methods to perform user-controlled editing of complex geometric models.

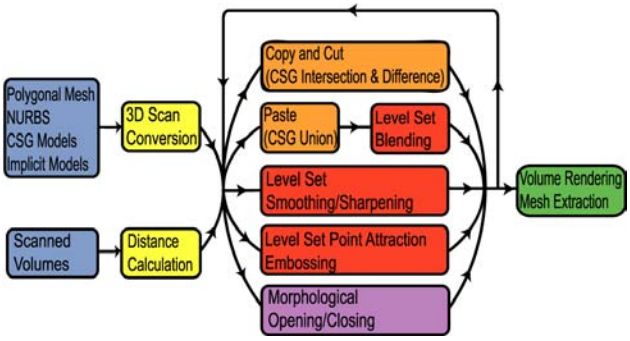


Figure 3: Our level set surface editing operators (red) fit into a larger editing framework. The pipeline consists of: input models (blue), pre-processing (yellow), CSG operations (orange), local LS operators (red), global LS operators (purple) and rendering (green).

3 Overview of the Editing Pipeline

The level set surface editing operators should be viewed as components of a larger modeling framework. The pipeline for this framework is presented in Figure 3. The red components contain the level set speed functions that we have developed for localized surface editing. The remaining components contain the data and operations needed for level set modeling, input models (blue), preprocessing (yellow), CSG operations (orange), global LS operators (purple) and rendering (green). The pipeline provides the context for the details of our speed functions.

3.1 Input and Output Models

We are able to import a wide variety of closed geometric models into the level set environment. We represent a level set model as an iso-surfaces embedded in a distance volume. Frequently we only store distance information in a narrow band of voxels surrounding the level set surface. As illustrated in Figure 3 we have developed and collected a suite of scan conversion methods for converting polygonal meshes, CSG models [Breen et al. 2000], implicit primitives, and NURBS surfaces into distance volumes. Additionally many types of scanning processes produce volumetric models directly, e.g. MRI, CT and laser range scan reconstruction. These models may be brought into our level set environment as is or with minimal pre-processing. We frequently utilize Sethian's Fast Marching Method [Sethian 1996] to convert volume datasets into distance volumes. The models utilized in this paper and their original form are listed in Table 1.

Table 1: Native representations of the input models and dimensions of the corresponding scan converted distance volumes.

Model	Representation	Dimensions
Dragon	volumetric reconstruction	$356 \times 161 \times 251$
Griffin	volumetric reconstruction	$312 \times 148 \times 294$
Greek bust	polygonal reconstruction	$221 \times 221 \times 161$
Human head	polygonal reconstruction	$256 \times 246 \times 193$
Utah teapot	NURBS surface	$156 \times 232 \times 124$
Supertoroid	implicit primitive	$91 \times 91 \times 31$

In the final stage of the pipeline we can either volume render the surface directly or render a polygonal mesh extracted from the volume. While there are numerous techniques available for both approaches, we found extracting and rendering Marching Cubes meshes [Lorensen and Cline 1987] to be satisfactory.

4 Level Set Surface Modeling

The Level Set Method, first presented in [Osher and Sethian 1988], is a mathematical tool for modeling surface deformations. A deformable (*i.e.* time-dependent) surface is implicitly represented as an iso-surface of a time-varying scalar function, $\phi(\boldsymbol{x}, t)$. A detailed description of level set models is presented in Appendix A.

4.1 LS Speed Function Building Blocks

Given the definition

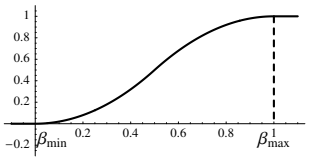
$$\mathcal{F}(\boldsymbol{x}, \boldsymbol{n}, \phi, \dots) \equiv \boldsymbol{n} \cdot \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t}, \tag{1}$$

the fundamental level set equation, Eq. (12), can be rewritten as

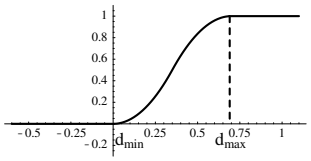
$$\frac{\partial \phi}{\partial t} = |\nabla \phi| \, \mathcal{F}(\boldsymbol{x}, \boldsymbol{n}, \phi, \dots) \tag{2}$$

where $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$ and $\boldsymbol{n} \equiv -\nabla\phi/|\nabla\phi|$ are the velocity and normal vectors at $\boldsymbol{x}$ on the surface. We assume a positive-inside/negative-outside sign convention for $\phi(\boldsymbol{x}, t)$, *i.e.* $\boldsymbol{n}$ *points outward*. Eq. (1) introduces the speed function $\mathcal{F}$, which is a user-defined scalar function that can depend on any number of variables including $\boldsymbol{x}$, $\boldsymbol{n}$, ϕ and its derivatives evaluated at $\boldsymbol{x}$, as well as a variety of external data inputs. $\mathcal{F}()$ is a *signed* scalar function that defines the motion (*i.e.* speed) of the level set surface in the direction of the local normal $\boldsymbol{n}$ at $\boldsymbol{x}$.

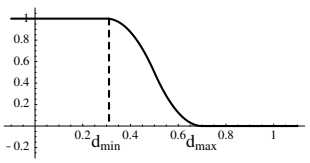
The speed function is usually based on a set of geometric measures of the implicit level set surface and data inputs. The challenge when working with level set methods is determining how to combine the building blocks to produce a local motion that creates a desired global or regional behavior of the surface. The general



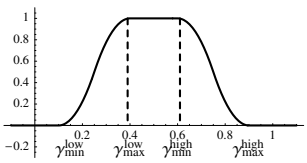
(a) Building-block, $P(\beta)$, for the ROI functions, *cf.* Eq. (4).



(b) Function, $\mathcal{D}_s(d)$, of distance to a surface, *cf.* Eq. (6b).



(c) Function, $\mathcal{D}_p(d)$, of distance to point-set, *cf.* Eq. (6a).



(d) Function, $\mathcal{C}(\gamma)$, of geometric measure, γ, *cf.* Eq. (7).

Figure 4: Graph of region-of-influence (ROI) functions used to define the speed functions for our local level set operations, *cf.* Eq. (3).

structure for the speed functions used in our surface editing operators is

$$\mathcal{F}(\boldsymbol{x}, \boldsymbol{n}, \phi) = \mathcal{D}_q(d)\mathcal{C}(\gamma)\mathcal{G}(\gamma) \tag{3}$$

where $\mathcal{D}_q(d)$ is a distance-based cut-off function which depends on a distance measure d to a geometric structure q. $\mathcal{C}(\gamma)$ is a cut-off function that controls the contribution of $\mathcal{G}(\gamma)$ to the speed function. $\mathcal{G}(\gamma)$ is a function that depends on geometric measures γ derived from the level set surface, e.g. curvature. Thus, $\mathcal{D}_q(d)$ acts as a region-of-influence function that regionally constrains the LS calculation. $\mathcal{C}(\gamma)$ is a filter of the geometric measure and $\mathcal{G}(\gamma)$ provides the geometric contribution of the level set surface. In general γ is defined as zero, first, or second order measures of the LS surface.

4.2 Regionally Constraining LS Deformations

Most of our surface operators may be applied locally in a small user-defined region on the edited surface. In order to regionally restrict the deformation during the level set computation, a technique is needed for driving the value of $\mathcal{F}()$ to zero outside of the region. This is accomplished in three steps. The first step involves defining the region of influence (ROI), *i.e.* the region where $\mathcal{F}()$ should be non-zero. This is done by either the user interactively placing a 3D object around the region, or by automatically calculating a region from properties of the surface. Both cases involve defining a geometric structure that we refer to as a "region-of-influence (ROI) primitive". The nature of these primitives will vary for the different LS operations and will explicitly be defined in Section 5. The second step consists of calculating a distance measure to the ROI primitive. The final step involves defining a function that smoothly approaches zero at the boundary of the ROI.

We define a region-of-influence function $\mathcal{D}_q(d)$ in Eq. (3), where d is a distance measure from a point on the level set surface to the ROI primitive q. The functional behavior of $\mathcal{D}_q(d)$ clearly depends on the specific ROI primitive, q, but we found the following piecewise polynomial function to be useful as a common speed function building block:

$$P(\beta) = \begin{cases} 0 & \text{for } \beta \leq 0 \\ 2\beta^2 & \text{for } 0 < \beta \leq 0.5 \\ 1 - 2(\beta - 1)^2 & \text{for } 0.5 < \beta < 1 \\ 1 & \text{for } \beta \geq 1. \end{cases} \tag{4}$$

$P(\beta)$ and its derivatives are continuous and relatively inexpensive to compute. See Figure 4(a). Other continuous equations with the same basic shape would also be satisfactory. We then define

$$\mathcal{P}(d; d_{min}, d_{max}) \equiv P\left(\frac{d - d_{min}}{d_{max} - d_{min}}\right) \tag{5}$$

where d_{min} and d_{max} are user-defined parameters that define the limits and sharpness of the cut-off. Let us finally define the following region-of-influence functions

$$\mathcal{D}_p(d) = 1 - \mathcal{P}(d; d_{min}, d_{max}) \tag{6a}$$
$$\mathcal{D}_s(d) = \mathcal{P}(d; 0, d_{max}) \tag{6b}$$

for a point-set, p, and a closed surface, s.

In Eq. (6a) d denotes the distance from a point on the level set surface to the closet point in the point-set p. In Eq. (6b) d denotes a signed distance measure from a point on the level set surface to the implicit surface s. The signed distance measure does not necessarily have to be Euclidean distance - just a monotonic distance measure following the positive-inside/negative-outside convention. Note that $\mathcal{D}_p(d)$ is one when the shortest distance, d, to the point-set is smaller than d_{min}, and decays smoothly to zero as d increases to d_{max}, after which it is zero. $\mathcal{D}_s(d)$, on the other hand, is zero everywhere outside, as well as on, the surface s ($d \leq 0$), but one inside when the distance measure d is larger than d_{max}.

An additional benefit of the region-of-influence functions is that they define the portion of the volume where the surface cannot move. We use this information to determine what voxels should be updated during the level set deformation, significantly lowering the amount of computation needed when performing editing operations. This technique allows our operators to be rapidly computed when modifying large models.

4.3 Limiting Geometric Property Values

We calculate a number of geometric properties from the level set surface. The zero order geometric property that we utilize is shortest distance from the level set surface to some ROI primitive. The first order property is the surface normal, $\boldsymbol{n} \equiv -\nabla\phi/|\nabla\phi|$. Second order information includes a variety of curvature measures of the LS surface. In Appendix B we outline a new numerical approach to deriving the mean, Gaussian and principle curvatures of a level set surface. Our scheme has numerical advantages relative to traditional central finite difference schemes for computing the second order derivatives. We found the mean curvature to be a useful second order measure [Evans and Spruck 1991].

Another desirable feature of our operators is that they allow the user to control the geometric properties of surface in the region being edited. This feature is implemented with another cut-off function, $\mathcal{C}()$, within the level set speed function. $\mathcal{C}()$ allows the user to slow and then stop the level set deformation as a particular surface property approaches a user-specified value. We reuse the cut-off function, Eq. (5), defined in the previous section, as a building block for $\mathcal{C}()$. We define

$$\mathcal{C}(\gamma) = \begin{cases} \mathcal{P}(\gamma; \gamma_{min}^{low}, \gamma_{max}^{low}) & \text{for } \gamma \leq \overline{\gamma} \\ 1 - \mathcal{P}(\gamma; \gamma_{min}^{high}, \gamma_{max}^{high}) & \text{for } \gamma > \overline{\gamma} \end{cases} \tag{7}$$

where $\overline{\gamma} \equiv (\gamma_{max}^{low} + \gamma_{min}^{high})/2$. The four parameters γ_{min}^{low}, γ_{max}^{low}, γ_{min}^{high}, and γ_{max}^{high} define respectively the upper and lower support of the filter, see Figure 4(d).

4.4 Constraining the Direction of LS Motions

Another important feature of the level set framework is its ability to control the direction of the level set deformation. We are able

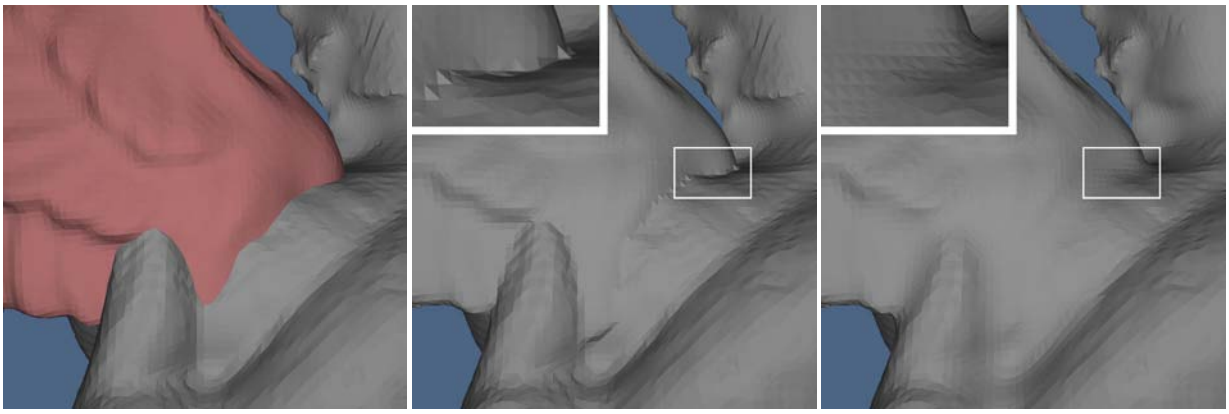


Figure 5: Left: Positioning the (red) wing model on the dragon model. Middle: The models are pasted together (CSG union operation), producing sharp, undesirable creases, a portion of which is expanded in the box. Right: Same region after automatic blending based on mean curvature. The blending is constrained to only move outwards. The models are rendered with flat-shading to highlight the details of the surface structure.

to restrict the motion of the surface to only add or remove material during the level set editing operations. At any point the level set surface can only move in the direction of the local surface normal. Hence, we can simply redefine the speed function as $\min(\mathcal{G}, 0)$ to remove material (inward motion only) and $\max(\mathcal{G}, 0)$ to add material (outward motion only). In the case of curvature driven speed functions this produces min/max flows [Sethian 1999]. Of course no restriction on the direction of the motion need be imposed.

5 Definition of Surface Editing Operators

Given the building blocks described in the previous section, the level set surface editing operators outlined in Figure 3 may be defined. We begin by defining the well-known CSG operations that are essential to most editing systems. We then define the new level set speed functions that implement our surface editing operators by combining the geometric measures with the region-of-influence and cut-off functions.

5.1 CSG Operations

Since level set models are volumetric, the constructive solid geometry (CSG) [Hoffmann 1989] operations of union, difference and intersection may be applied to them. This provides a straightforward approach to copy, cut and paste operations on the level set surfaces. In our level set framework with a positive-inside/negative-outside sign convention for the distance volumes these are implemented as min/max operations [Wang and Kaufman 1994] on the voxel values as summarized in Table 2. Any two closed surfaces represented as signed distance volumes can be used as either the main edited model or the cut/copy primitive. In our editing system the user is able to arbitrarily scale, translate and rotate the models before a CSG operation is performed.

Table 2: Implementation of CSG operations on two level set models, A and B, represented by distance volumes V_A and V_B with positive inside and negative outside values.

Action	CSG Operation	Implementation
Copy	Intersection, $A \cap B$	$\text{Min}(V_A, V_B)$
Paste	Union, $A \cup B$	$\text{Max}(V_A, V_B)$
Cut	Difference, $A - B$	$\text{Min}(V_A, -V_B)$

5.2 Automatic Localized LS Blending

The surface models produced by the CSG paste operation typically produces sharp and sometimes jagged creases at the intersection of the two surfaces. We can dramatically improve this region of the surface by applying an automatic localized blending. The method is automatic because it only requires the user to specify a few parameter values. It is localized because the blending operator is only applied near the surface intersection region. One possible solution to localizing the blending is to perform the deformation in regions near both of the input surfaces. However, this naive approach would result in blending the two surfaces in *all* regions of space where the surfaces come within a user-specified distance of each other, creating unwanted blends. A better solution, and the one we use, involves defining the region of influence based on the distance to the *intersection curve* shared by both input surfaces. A sampled representation of this curve is the set of voxels that contains a zero distance value (within some sub-voxel value ϵ) to both surfaces. We have found this approximate representation of the intersection curve as a point-set to be sufficient for defining a shortest distance d for the region-of-influence function, $\mathcal{D}_p(d)$, *cf.* Eq. (3). Representing the intersection curve by a point-set allows the curve to take an arbitrary form - it can even be composed of multiple curve segments without introducing any complication to the computational scheme.

The blending operator moves the surface in a direction that minimizes a curvature measure, $\mathcal{K}$, on the level set surface. This is obtained by making the speed function, $\mathcal{G}$, Eq. (3), proportional to $\mathcal{K}$, leading to the following blending speed function:

$$\mathcal{F}_{blend}(\boldsymbol{x}, \boldsymbol{n}, \phi) = \alpha\, \mathcal{D}_p(d)\, \mathcal{C}(\mathcal{K})\, \mathcal{K} \tag{8}$$

where α is a user-defined positive scalar that is related to the rate of convergence of the LS calculation, $\mathcal{D}_p(d)$ is defined in Eq. (6a) where d is the shortest distance from the level set surface to the intersection curve point set, and $\mathcal{C}(\mathcal{K})$ is given by Eq. (7) where $\mathcal{K}$ is one of the curvatures define in Appendix B. Through the functions $\mathcal{D}_p$ and $\mathcal{C}$ the user has full control over the region of influence of the blending (d_{min} and d_{max}) and the upper and lower curvature values of the blend (γ_{min}^{low}, γ_{max}^{low} and γ_{min}^{high}, γ_{max}^{high}). Furthermore we can control if the blend adds or removes material, or both as described in Section 4.4.

Automatic blending is demonstrated in Figure 5. A wing model is positioned relative to a dragon model. The two models are pasted together and automatic mean curvature-based blending is applied to smooth the creased intersection region.

5.3 Localized LS Smoothing/Sharpening

The smoothing operator smooths the level set surface in a user-specified region. This is accomplished by enclosing the region of interest by a geometric primitive. The "region-of-influence prim-

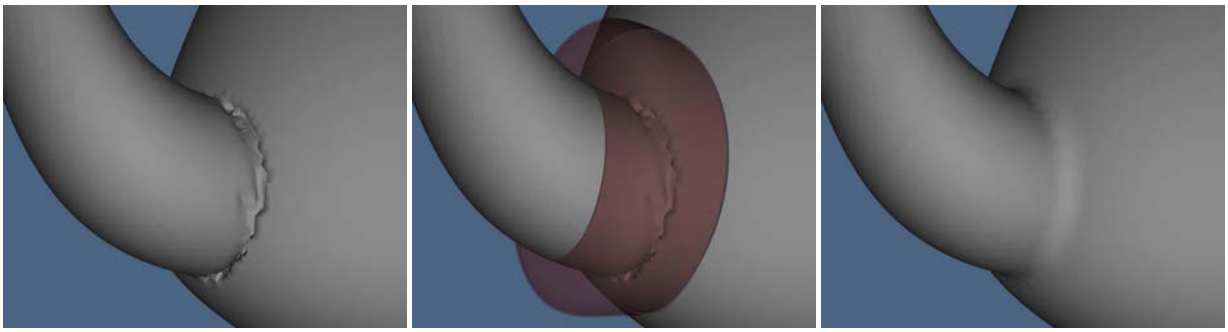


Figure 6: (left) Scan conversion errors near the teapot spout. (middle) Placing a (red) superellipsoid around the errors. (right) The errors are smoothed away in 15 seconds. The surface is constrained to only move outwards.

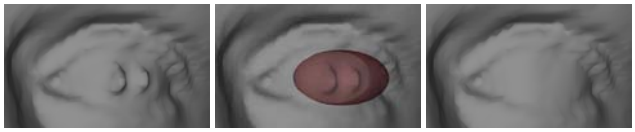


Figure 7: Regionally constrained smoothing. Left: Laser scan reconstruction with unwanted, pointed artifacts in the eye. Middle: Defining the region to be smoothed with a (red) superellipsoid. Right: Smoothing the surface within the superellipsoid. The surface is constrained to only move inwards.

itive" can be any closed surface for which we have signed inside/outside information, e.g. a level set surface or an implicit primitive. We use superellipsoids [Barr 1981] as a convenient ROI primitive, a flexible implicit primitive defined by two shape parameters. The surface is locally smoothed by applying motions in a direction that reduces the local curvature. This is accomplished by moving the level set surface in the direction of the local normal with a speed that is proportional to the curvature. Therefore the speed function for the smoothing operator is

$$\mathcal{F}_{smooth}(\boldsymbol{x}, \boldsymbol{n}, \phi) = \alpha \mathcal{D}_s(d)\mathcal{C}(\mathcal{K})\mathcal{K}. \tag{9}$$

Here d denotes the signed value of the monotonic inside/outside function of the ROI primitive s evaluated at $\boldsymbol{x}$. As before, $\mathcal{C}_s(d)$ ensures that the speed function smoothly goes to zero as $\boldsymbol{x}$ approaches the boundary of the ROI primitive.

Figure 7 demonstrates our smoothing operator applied to a laser scan reconstruction. Unwanted artifacts are removed from an eye by first placing a red superellipsoid around the region of interest. A smoothing operator constrained to only remove material is applied and the spiky artifacts are removed. Figure 6 demonstrates our smoothing operator applied to a preliminary 3D scan conversion of the Utah teapot. Unwanted artifacts are removed from the region where the spout meets the body of the teapot by first placing a superellipsoid around the region of interest. A smoothing operator constrained to only add material is applied and the unwanted artifacts are removed. In our final, artificial smoothing example in Figure 9 a complex structure is completely smoothed away. This examples illustrates that changes of topological genus and number of disconnected components are easily handled within a level set framework during smoothing.

We obtain a sharpening operator by simply inverting the sign of α in Eq. (9) and applying an upper cut-off to the curvature in $\mathcal{C}()$ in order to maintain numerical stability. The sharpening operator has been applied to the hair of the Greek bust in Figure 1.

5.4 Point-Set Attraction and Embossing

We have developed an operator that attracts and repels the surface towards and away from a point set. These point sets can be samples of lines, curves, planes, patches and other geometric shapes, *e.g.* text. By placing the point sets near the surface, we are able to emboss the surface with the shape of the point set. Similar to the smoothing operator, the user encloses the region to be embossed with a ROI primitive *e.g.* a superellipsoid. The region-of-interest function for this operator is $\mathcal{D}_s(d)$, Eq. (6b).

Figure 8: Left: Three types of single point attractions/repulsions using different ROI primitives and γ values. Right: Utah teapot embossed with 7862 points sampling the "SIGGRAPH 2002" logo.

First, assume that all of the attraction points are located outside the LS surface. $\boldsymbol{p}_i$ denotes the closest attraction point to $\boldsymbol{x}$, a point on the LS surface. Our operator only allows the LS surface to move towards $\boldsymbol{p}_i$ if the unit vector, $\boldsymbol{u}_i \equiv (\boldsymbol{p}_i - \boldsymbol{x})/|\boldsymbol{p}_i - \boldsymbol{x}|$, is pointing in the same direction as the local surface normal $\boldsymbol{n}$. Hence, the speed function should only be non-zero when $0 < \boldsymbol{n} \cdot \boldsymbol{u}_i \leq 1$. Since the sign of $\boldsymbol{n} \cdot \boldsymbol{u}_i$ is reversed if $\boldsymbol{p}_i$ is instead located inside the LS surface we simply require $\gamma = -\text{sign}[\phi(\boldsymbol{p}_i, t)]\boldsymbol{n} \cdot \boldsymbol{u}_i$ to be positive for any closest attraction point $\boldsymbol{p}_i$. This amounts to having only positive cut-off values for $\mathcal{C}(\gamma)$. Finally we let $\mathcal{G} = -\alpha\phi(\boldsymbol{p}_i, t)$ since this will guarantee that the LS surface, $\boldsymbol{x}$, actually stops once it reaches $\boldsymbol{p}_i$. The following speed function implements the point-set attraction operator:

$$\mathcal{F}_{point}(\boldsymbol{x}, \boldsymbol{n}, \phi) = -\alpha \mathcal{D}_s(d)\mathcal{C}(-\text{sign}[\phi(\boldsymbol{p}_i, t)]\boldsymbol{n} \cdot \boldsymbol{u}_i)\phi(\boldsymbol{p}_i, t), \tag{10}$$

where d is a signed distance measure to a ROI primitive evaluated at $\boldsymbol{x}$ on the LS surface, and p_i is the closest point in the set to $\boldsymbol{x}$. The shape of the primitive and the values of the four positive parameters in Eq. (7) define the footprint and sharpness of the embossing. See Figure 8, left. Point-repulsion is obtained by making α negative. Note that Eq. (10) is just one example of many possible point-set attraction speed functions.

In Figure 8, right, the Utah teapot is embossed with 7862 points that have been acquired by scanning an image of the SIGGRAPH 2002 logo and warping the points to fit the shape of the teapot.

5.5 Global Morphological Operators

The new level set operators presented above were designed to perform localized deformations of a level set surface. However, if the user wishes to perform a global smoothing of a level set surface, it is advantageous to use an operator other than $\mathcal{F}_{smooth}$. For

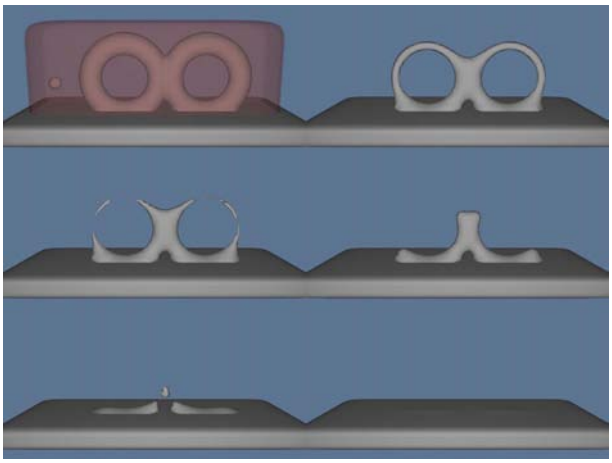


Figure 9: Changes in topological genus and the number of disconnected components are easily handled within a level set framework during smoothing. The superellipsoid defines the portion of the surface to be smoothed. The surface is constrained to move only inwards.

a global smoothing the level set propagation is computed on the whole volume, which can be slow for large volumes. However, in this case morphological opening and closing operators [Serra 1982] offer faster alternatives to global smoothing of level set surfaces. While we are not the first to explore morphological operators within a level set framework [Sapiro et al. 1993; Maragos 1996], we have implemented them and find them useful. Morphological openings and closings consist of two fundamental operators, dilations D_ω and erosions E_ω. Dilation creates an offset surface a distance ω outward from the original surface, and erosion creates an offset surface a distance ω inwards from the original surface. The morphological opening operator O_ω is an erosion followed by a dilation, i.e. $\mathrm{O}_\omega = \mathrm{D}_\omega \circ \mathrm{E}_\omega$, which removes small pieces or thin appendages. A closing is defined as $\mathrm{C}_\omega \phi = \mathrm{E}_\omega \circ \mathrm{D}_\omega \phi$, and closes small gaps or holes within objects. Morphological operators may be implemented by solving a special form of the level set equation, the Eikonal equation, $\partial\phi/\partial t = \pm|\nabla\phi|$, up to a certain time t, utilizing Sethian's Fast Marching Method [Sethian 1996]. The value of t controls the offset distance from the original surface of $\phi(t = 0)$. Figure 10 contains a model from a laser scan reconstruction that has been smoothed with an opening operator with ω equal to 3.

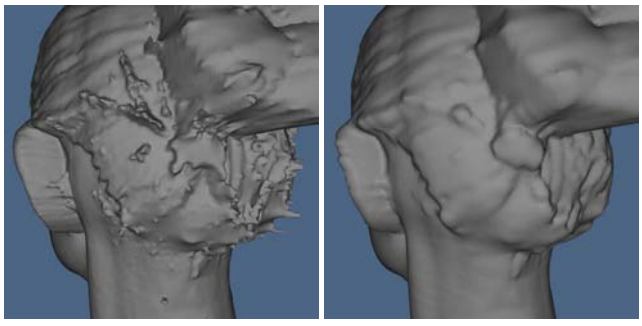


Figure 10: Applying a morphological opening to a laser scan reconstruction of a human head. The opening performs global smoothing by removing protruding structures smaller than a user-defined value.

5.6 Editing Session Details

Figure 11 contains a series of screen shots taken of our level set modeling program while constructing the two-headed winged dragon. The first shows the original dragon model loaded into the system. A cylindrical primitive is placed around its head and it is cut off. The model of the head is duplicated and the two heads are positioned relative to each other. Once the user is satisfied with their orientation, they are pasted together and an automatic blending is performed at the intersection seam. The combined double head model is positioned over the cropped neck of the dragon body. The double head is pasted and blended onto the body. The griffin model is loaded into the LS modeling system. A primitive is place around one of its wings. The portion of the model within the primitive is copied, being stored in a buffer. Several cutting operations are used to trim the wing model (not shown). The double-headed dragon model is loaded, and the wing is positioned, pasted and blended onto it. A mirror copy of the wing model is created. It is also positioned, pasted and blended onto the other side of the double-headed dragon. We then added a loop onto the dragon's back as if designing a bracelet charm. This is accomplished by positioning, pasting, and blending a scan-converted supertoroid, producing the final model seen in the bottom right.

The Greek bust model was repaired by copying the nose from the human head model of Figure 10, and pasting and blending the copied model onto the broken nose. A piece from the right side of the bust was copied, mirrored, pasted and blended onto the left side of her face. Local smoothing operators were applied to various portions of her cheeks to clean minor cracks. Finally, the sharpening operator was applied within a user-defined region around her hair.

Table 3: Typical operator execution times on a R10K 250MHz MIPS processor.

Operation	Objects	sub-volume	Time
Paste	wing on dragon	$316 \times 172 \times 215$	33 sec.
Blend	wing on dragon	$82 \times 48 \times 63$	98 sec.
Smooth	teapot spout	$60 \times 55 \times 31$	15 sec.
Opening	human head	$256 \times 246 \times 193$	22 sec.
Emboss	single point	$21 \times 29 \times 29$	1.5 sec.

Table 4: Parameters used in examples. γ_{min}^{high} and γ_{min}^{high} are only used during sharpening. Their values are 0.8 and 0.9. No upper limit is placed on γ in the other examples.

Example	d_{min}	d_{max}	γ_{min}^{low}	γ_{max}^{low}
Wing Blending	7	9	0.04	0.06
Eye Smoothing	0.9	1	0.04	0.07
Spout Smoothing	0.9	1	0.1	0.13
Hair Sharpening	0.9	1	0.01	0.013
Teapot Embossing	0.9	1	0.8	0.9

6 Conclusion and Future Work

We have presented an approach to implementing surface editing operators within a level set framework. By developing a new set of level set speed functions automatic blending, localized smoothing and embossing may be performed on level set models. Additionally we have implemented morphological and volumetric CSG operators to fill out our modeling environment. All of the information needed to deform a level set surface is encapsulated in the speed function, providing a simple, unified computational framework. The level set framework offers several advantages. By construction, self-intersection cannot occur, which guarantees the generation of physically-realizable, simple, closed surfaces. Additionally, level set models easily change topological genus, and are free of the edge connectivity and mesh quality problems associated with mesh models.

Several issues still must be addressed to improve our work. Currently level set implementations are based on uniform samplings of

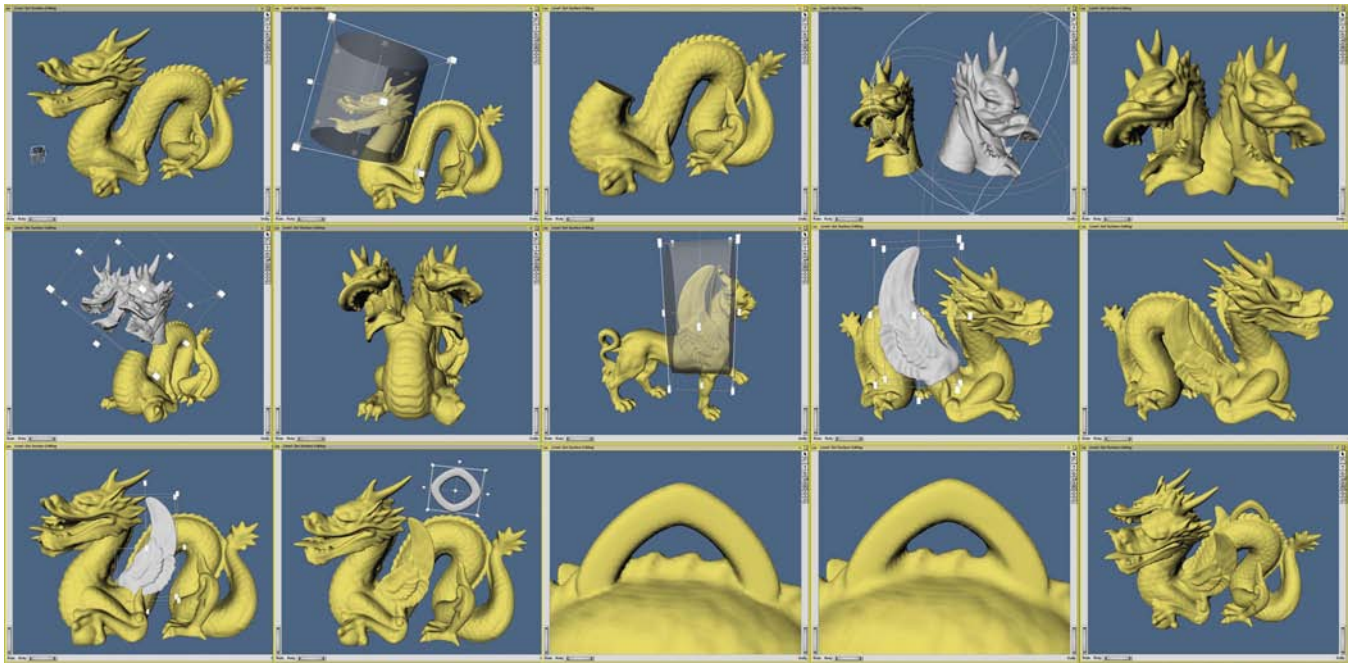


Figure 11: Series of operations used to create the winged two-headed dragon of Figure 1. First the head is cut off, pasted and blended back onto the body. Next a wing is copied from a different model and blended onto one side of the dragon. The same wing is then mirrored and blended onto the other side. Finally a scan converted supertoroid is blended onto the dragon's back to form the loop of a bracelet charm.

space, a fact that effectively limits the resolution of the objects that can be modeled. The development of adaptive level set methods would allow our operators to be applied to adaptive distance fields. It is possible to shorten the time needed to edit level set surfaces. Incrementally updating the mesh used to view the edited surface, utilizing direct volume rendering hardware, parallelizing the level set computations, and exploring multiresolution volumetric representations will lead to editing operations that require only a fraction of a second, instead of tens of seconds.

We have presented five example level set surface editing operators. Given the generality and flexibility of our framework many more can be developed. We intend to explore operators that utilize Gaussian and principal curvature, extend embossing to work directly with lines, curves and solid objects, and ones that may be utilized for general surface manipulations, such as dragging, warping, and sweeping.

7 Acknowledgements

We would like to thank Mathieu Desbrun for his helpful suggestions, and Cici Koenig and Katrine Museth for helping us with the figures. The Greek bust and human head models were provided by Cyberware Inc. The dragon and griffin models were provided the Stanford Computer Graphics Laboratory. The teapot model was provided by the University of Utah's Geometric Design and Computation Group. This work was financially supported by National Science Foundation grants ASC-89-20219, ACI-9982273 and ACI-0083287.

References

ADALSTEINSSON, D., AND SETHIAN, J. 1995. A fast level set method for propagating interfaces. *Journal of Computational Physics*, 269–277.

AMENTA, N., BERN, M., AND KAMVYSSELIS, M. 1998. A new voronoi-based surface reconstruction algorithm. In *Proc. SIGGRAPH '98*, 415–421.

BAJAJ, C., BERNARDINI, F., AND XU, G. 1995. Automatic reconstruction of surfaces and scalar fields from 3D scans. In *Proc. SIGGRAPH '95*, 109–118.

BARR, A. 1981. Superquadrics and angle-preserving transformations. *IEEE Computer Graphics and Applications 1*, 1, 11–23.

BIERMANN, H., KRISTJANSSON, D., AND ZORIN, D. 2001. Approximate Boolean operations on free-form solids. In *Proc. SIGGRAPH 2001*, 185–194.

BLOOMENTHAL, J., ET AL., Eds. 1997. *Introduction to Implicit Surfaces*. Morgan Kaufmann, San Francisco.

BOUGUET, J.-Y., AND PERONA, P. 1999. 3D photography using shadow in dual space geometry. *International Journal of Computer Vision 35*, 2 (Nov/Dev), 129–149.

BREEN, D., AND WHITAKER, R. 2001. A level set approach for the metamorphosis of solid models. *IEEE Trans. on Visualization and Computer Graphics 7*, 2, 173–192.

BREEN, D., MAUCH, S., AND WHITAKER, R. 2000. 3D scan conversion of CSG models into distance, closest-point and color volumes. In *Volume Graphics*, M. Chen, A. Kaufman, and R. Yagel, Eds. Springer, London, 135–158.

COHEN, E., RIESENFELD, R., AND ELBER, G. 2001. *Geometric Modeling with Splines*. AK Peters, Natick, MA.

CURLESS, B., AND LEVOY, M. 1996. A volumetric method for building complex models from range images. In *Proc. SIGGRAPH '96*, 303–312.

DESBRUN, M., AND GASCUEL, M. 1995. Animating soft substances with implicit surfaces. In *Proc. SIGGRAPH 95 Conference*, 287–290.

DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. 1999. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proc. SIGGRAPH '99*, 317–324.

DO CARMO, M. 1976. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, Englewood Cliffs, NJ.

EDELSBRUNNER, H., AND MÜCKE, E. 1994. Three-dimensional alpha shapes. *ACM Trans. on Graphics 13*, 1, 43–72.

EVANS, L., AND SPRUCK, J. 1991. Motion of level sets by mean curvature, I. *Journal of Differential Geometry*, 635–681.

FOSTER, N., AND FEDKIW, R. 2001. Practical animation of liquids. In *Proc. SIGGRAPH 2001*, 23–30.

FRISKEN, S., PERRY, R., ROCKWOOD, A., AND JONES, T. 2000. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *SIGGRAPH 2000 Proceedings*, 249–254.

GALYEAN, T., AND HUGHES, J. 1991. Sculpting: An interactive volumetric modeling technique. In *Proc. SIGGRAPH '91*, 267–274.

HOFFMANN, C. 1989. *Geometric and Solid Modeling*. Morgan Kaufmann, San Francisco.

KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P. 1998. Interactive multi-resolution modeling on arbitrary meshes. In *Proc. SIGGRAPH '98*, 105–114.

KOBBELT, L. P., BOTSCH, M., SCHWANECKE, U., AND SEIDEL, H.-P. 2001. Feature sensitive surface extraction from volume data. In *Proc. SIGGRAPH 2001*, 57–66.

LAIDLAW, D., TRUMBORE, W., AND HUGHES, J. 1986. Constructive solid geometry for polyhedral objects. In *Proc. SIGGRAPH '86*, vol. 20, 161–170.

LORENSEN, W., AND CLINE, H. 1987. Marching Cubes: A high resolution 3D surface construction algorithm. In *Proc. SIGGRAPH '87*, 163–169.

MALLADI, R., SETHIAN, J., AND VEMURI, B. 1995. Shape modeling with front propagation: A level set approach. *IEEE Trans. on Pattern Analysis and Machine Intelligence 17*, 2, 158–175.

MARAGOS, P. 1996. Differential morphology and image processing. *IEEE Trans. on Image Processing 5*, 6 (June), 922–937.

OSHER, S., AND FEDKIW, R. 2001. Level set methods: An overview and some recent results. *Journal of Computational Physics 169*, 475–502.

OSHER, S., AND SETHIAN, J. 1988. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics 79*, 12–49.

PENG, D., MERRIMAN, B., OSHER, S., ZHAO, H.-K., AND KANG, M. 1999. A PDE-based fast local level set method. *Journal of Computational Physics 155*, 410–438.

PERRY, R., AND FRISKEN, S. 2001. Kizamu: A system for sculpting digital characters. In *Proc. SIGGRAPH 2001*, 47–56.

REQUICHA, A., AND VOELCKER, H. 1985. Boolean operations in solid modeling: Boundary evaluation and merging algorithms. *Proceedings of the IEEE 73*, 1, 30–44.

RUDIN, L., OSHER, S., AND FATEMI, C. 1992. Nonlinear total variation based noise removal algorithms. *Physica D 60*, 259–268.

SAPIRO, G., KIMMEL, R., SHAKED, D., KIMIA, B., AND BRUCKSTEIN, A. 1993. Implementing continuous-scale morphology via curve evolution. *Pattern Recognition*, 9, 1363–1372.

SAPIRO, G. 2001. *Geometric Partial Differential Equations and Image Analysis*. Cambridge University Press, Cambridge, UK.

SERRA, J. 1982. *Image Analysis and Mathematical Morphology*. Academic Press, London.

SETHIAN, J. 1996. A fast marching level set method for monotonically advancing fronts. In *Proceedings of the National Academy of Science*, vol. 93 of 4, 1591–1595.

SETHIAN, J. 1999. *Level Set Methods and Fast Marching Methods*, second ed. Cambridge University Press, Cambridge, UK.

TAUBIN, G. 1995. A signal processing approach to fair surface design. In *Proc. SIGGRAPH '95*, 351–358.

TSITSIKLIS, J. 1995. Efficient algorithms for globally optimal trajectories. *IEEE Trans. on Automatic Control 40*, 9, 1528–1538.

WANG, S., AND KAUFMAN, A. 1994. Volume-sampled 3D modeling. *IEEE Computer Graphics and Applications 14*, 5 (September), 26–32.

WANG, S., AND KAUFMAN, A. 1995. Volume sculpting. In *1995 Symposium on Interactive 3D Graphics*, ACM SIGGRAPH, 151–156.

WELCH, W., AND WITKIN, A. 1994. Free-form shape design using triangulated surfaces. In *Proc. SIGGRAPH '94*, 247–256.

WHITAKER, R., AND XUE, X. 2001. Variable-conductance, level-set curvature for image denoising. In *Proc. IEEE International Conference on Image Processing*, 142–145.

WHITAKER, R., BREEN, D., MUSETH, K., AND SONI, N. 2001. Segmentation of biological datasets using a level-set framework. In *Volume Graphics 2001*, M. Chen and A. Kaufman, Eds. Springer, 249–263.

WHITAKER, R. 1998. A level-set approach to 3D reconstruction from range data. *Int. Jrnl. of Comp. Vision October*, 3, 203–231.

WYVILL, B., GALIN, E., AND GUY, A. 1999. Extending the CSG tree. warping, blending and Boolean operations in an implicit surface modeling system. *Computer Graphics Forum 18*, 2 (June), 149–158.

ZHAO, H.-K., OSHER, S., AND FEDKIW, R. 2001. Fast surface reconstruction using the level set method. In *Proc. 1st IEEE Workshop on Variational and Level Set Methods*, 194–202.

A Level Set Models

A deformable (*i.e.* time-dependent) surface, $\mathcal{S}(t)$, is implicitly represented as an iso-surface of a time-varying[1] scalar function, $\phi(\boldsymbol{x}, t)$, embedded in 3D, *i.e.*

$$\mathcal{S}(t) = \{\boldsymbol{x}(t) \mid \phi(\boldsymbol{x}(t), t) = k\}, \tag{11}$$

where $k \in \Re$ is the iso-value, $t \in \Re^+$ is time, and $\boldsymbol{x}(t) \in \Re^3$ is a point in space on the iso-surface. It might seem inefficient to implicitly represent a surface with a 3D scalar function; however the higher dimensionality of the representation provides one of the major advantages of the LS method: the flexible handling of changes in the topology of the deformable surface. This implies that LS surfaces can easily represent complicated surface shapes that can, form holes, split to form multiple objects, or merge with other objects to form a single structure.

The fundamental level set equation of motion for $\phi(\boldsymbol{x}(t), t)$ is derived by differentiating both sides of Eq. (11) with respect to time t, and applying the chain rule giving:

$$\frac{\partial \phi}{\partial t} = -\nabla \phi \cdot \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t}, \tag{12}$$

where $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$ denotes the speed vectors of the level set surface. A number of numerical techniques by [Osher and Sethian 1988; Adalsteinsson and Sethian 1995] make the initial value problem of Eq. (12) computationally feasible. A complete discussion of the details of the level set method is beyond the scope of this paper. We instead refer the interested reader to [Sethian 1999; Osher and Fedkiw 2001]. However, we will briefly mention two of the most important techniques: the first is the so called "up-wind scheme" which addresses the problem of overshooting when trying to solve Eq. (12) by a simple finite forward difference scheme. The second is related to the fact that one is typically only interested in a single solution to Eq. (12), say the $k = 0$ level set. This implies that the evaluation of ϕ is important only in the vicinity of that level set. This forms the basis for "narrow-band" schemes [Adalsteinsson and Sethian 1995; Whitaker 1998; Peng et al. 1999] that solve Eq. (12) in a narrow band of voxels containing the surface. The "up-wind scheme" makes the level set method numerically robust, and the "narrow-band scheme" makes its computational complexity proportional to the level set's surface area rather than the size of the volume in which it is embedded.

[1] Our work uses the dynamic level set equation, which is more flexible than the corresponding stationary equation, $\phi(\boldsymbol{x}) = k(t)$, see [Sethian 1999] for more details.

B Curvature of Level Set Surfaces

The principle curvatures and principle directions are the eigenvalues and eigenvectors of the *shape matrix* [do Carmo 1976]. For an implicit surface, the shape matrix is the derivative of the normalized gradient (surface normals) projected onto the tangent plane of the surface. If we let the normals be $\boldsymbol{n} = \nabla\phi/|\nabla\phi|$, the derivative of this is the 3×3 matrix

$$\boldsymbol{N} = \left(\frac{\partial \boldsymbol{n}}{\partial x} \;\; \frac{\partial \boldsymbol{n}}{\partial y} \;\; \frac{\partial \boldsymbol{n}}{\partial z} \right)^T. \tag{13}$$

The projection of this derivative matrix onto the tangent plane gives the shape matrix [do Carmo 1976] $\boldsymbol{B} = \boldsymbol{N}(I - \boldsymbol{n} \otimes \boldsymbol{n})$, where $\otimes$ is the exterior product. The eigenvalues of the matrix $\boldsymbol{B}$ are k_1, k_2 and zero, and the eigenvectors are the principle directions and the normal, respectively. Because the third eigenvalue is zero, we can compute k_1, k_2 and various differential invariants directly from the invariants of $\boldsymbol{B}$. Thus the weighted curvature flow is computing from $\boldsymbol{B}$ using the identities $D = ||\boldsymbol{B}|\!|$, $H = \mathrm{Tr}(\boldsymbol{B})/2$, and $K = 2H^2 - D^2/2$. The choice of numerical methods for computing $\boldsymbol{B}$ is discussed in the following section. The principle curvature are calculated by solving the quadratic

$$k_{1,2} = H \pm \sqrt{\frac{D^2}{2} - H^2}. \tag{14}$$

In many circumstances, the curvature term, which is a kind of directional diffusion, which does not suffer from overshooting, can be computed directly from first- and second-order derivatives of ϕ using central difference schemes. However, we have found that central differences do introduce instabilities when computing flows that rely on quantities other than the mean curvature. Therefore we use the method of *differences of normals* [Rudin et al. 1992; Whitaker and Xue 2001] in lieu of central differences. The strategy is to compute normalized gradients at staggered grid points and take the difference of these staggered normals to get centrally located approximations to $\boldsymbol{N}$. The shape matrix $\boldsymbol{B}$ is computed with gradient estimates from central differences. The resulting curvatures are treated as speed functions (motion in the normal direction), and the associated gradient magnitude is computed using the up-wind scheme.

Algorithms for Interactive Editing of Level Set Models

Ken Museth[1] David E. Breen[2] Ross T. Whitaker[3] Sean Mauch[4] David Johnson[3]

[1]Linköping Institute of Technology [2]Drexel University [3]University of Utah [4]California Institute of Technology

Abstract

Level set models combine a low-level volumetric representation, the mathematics of deformable implicit surfaces, and powerful, robust numerical techniques to produce a novel approach to shape design. While these models offer many benefits, their large-scale representation and their numerical requirements create significant challenges when developing an interactive system. This paper describes the collection of techniques and algorithms (some new, some pre-existing) needed to overcome these challenges and to create an interactive editing system for this new type of geometric model. We summarize the algorithms for producing level set input models and, more importantly, for localizing/minimizing computation during the editing process. These algorithms include distance calculations, scan conversion, closest point determination, fast marching methods, bounding box creation, fast and incremental mesh extraction, numerical integration, and narrow band techniques. Together these algorithms provide the capabilities required for interactive editing of level set models.

1 Introduction

Level set models are a new type of geometric model for creating complex, closed objects. They combine a low-level volumetric representation, the mathematics of deformable implicit surfaces, and powerful, robust numerical techniques to produce a novel approach to shape design. During an editing session a user focuses on and conceptually interacts with the shape of a level set surface, while the level set methods "under the hood" calculate the appropriate voxel values for a particular editing operation, completely hiding the volumetric representation of the surface from the user.

More specifically, level set models are defined as an iso-surface, i.e. a level set, of some implicit function ϕ. The surface is deformed by solving a partial differential equation (PDE) on a regular sampling of ϕ, i.e. a volume dataset [Osher and Sethian 1988]. Thus, it should be emphasized that level set methods do not manipulate an explicit closed form representation of ϕ, but only a sampling of it. Level set methods provide the techniques needed to change the voxel values of the volume in a way that moves the embedded iso-surface to meet a user-defined goal.

Defining a surface with a volume dataset may seem unusual and inefficient, but level set models do offer numerous benefits in comparison to other types of geometric surface representations. They are guaranteed to define simple (non-self-intersecting) and closed surfaces. Thus level set editing operations will always produce a physically-realizable (and therefore manufacturable) object. Level set models easily change topological genus, making them ideal for representing complex structures of unknown genus. They are free of the edge connectivity and mesh quality problems common in surface mesh models. Additionally, they provide the advantages of implicit models, e.g. supporting straightforward solid modeling operations and calculations, while offering a powerful surface modeling paradigm.

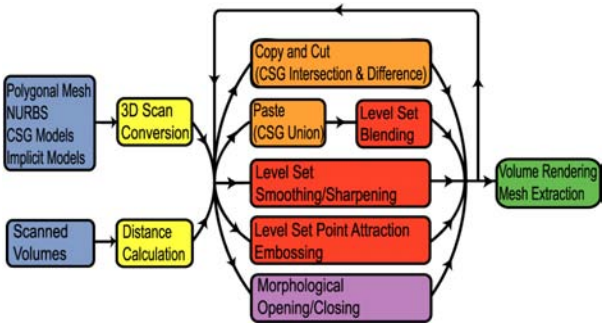


Figure 1: Level set modeling system modules. The system consists of: input models (blue), pre-processing (yellow), CSG operations (orange), local LS operators (red), global LS operators (purple) and rendering (green).

1.1 Level Set Model Editing System

We have developed an interactive system for editing level set models. The modules and the data flow of the system is diagramed in Figure 1. The blue modules contain the types of models that may be imported into the system. The yellow modules contain the algorithms for converting the input models into level set models. The orange, red and purple modules are the editing operations that can be performed on the models. The final (green) module renders the model for interactive viewing.

In a previous paper [Museth et al. 2002a] described the mathematical details of the editing operators, some of which were based on concepts proposed in [Whitaker and Breen 1998]. The cut-and-paste (orange) operators give the user the ability to copy, remove and merge level set models (using volumetric CSG operations) and automatically blend the intersection regions (1st red module). Our smoothing operator (2nd red module) allows a user to define a region of interest and smooths the enclosed surface to a user-defined curvature value. We have also developed a point-attraction operator. A regionally constrained portion of a level set surface may be attracted to a set of points to produce a surface embossing operator (3rd red operator). As noted by others, the opening and closing morphological (purple) operators [Serra 1982] may be implemented in a level set framework [Sapiro and Tannenbaum 1993; Maragos 1996]. We have also found them useful for global blending (closing) and smoothing (opening).

1.2 Challenges and Solutions

The volumetric representation and the mathematics of level set models create numerous challenges when developing an interactive level set editing system. Together they indicate the need for an enormous amount of computation on large-scale datasets, in order to numerically solve the level set equation at each voxel in the volume. In this paper, we address these issues, focusing on the implementation details of our level set editing work and describing the collection of algorithms (some new, some pre-existing) needed to create an interactive level set model editing system.

The first challenge encountered when editing level set models is converting conventional surface representations into the volumetric format needed for processing with level set methods. Our goal has been to connect level set editing with other forms of geometric modeling. The user may utilize pre-existing modeling tools to create a variety of models. Creating a suite of model conversion tools allows those models to be imported into our system for additional modifications using editing operations unique to level set models. We therefore have implemented several 3D scan conversion algorithms. The essential computation for most of these algorithms involves calculating a closest point (and therefore the shortest distance) from a point to the model.

The second major challenge of interactive level set model editing is minimizing the amount of computation needed to perform the individual operations. The mathematics of level set models is defined globally, but in practice most level set operators only modify a small portion of the model. We therefore employ a variety of techniques to localize the level set computations in order to make the editing system interactive. Since we are only interested in one level set (iso-surface) in the volume, narrow band techniques [Adalsteinsson and Sethian 1995; Whitaker 1998; Peng et al. 1999] may be used to make the computation proportional to the surface area of the model. Additionally, the extensive use of bounding boxes further limits the region of computation on the surface. Some of our operators require a closest-point-in-set calculation. Here K-D trees [de Berg et al. 1997; Arya et al. 1998] are utilized. Finally, interactive viewing is made possible by an incremental, optimized mesh extraction algorithm. Brought together, all of these techniques and data structures allow us to import a variety of models and interactively edit them with level set surface editing operators.

1.3 Related Work

Two areas of research are closely related to our level set surface editing work; volumetric sculpting, and implicit modeling. Volumetric sculpting provides methods for directly manipulating the voxels of a volumetric model. CSG Boolean operations [Hoffmann 1989; Wang and Kaufman 1994] are commonly found in volume sculpting systems, providing a straightforward way to create complex solid objects by combining simpler primitives. One of the first volume sculpting systems is presented in [Galyean and Hughes 1991]. Incremental improvements to the concept of volume sculpting soon followed. [Wang and Kaufman 1995] introduced tools for carving and sawing, [Avila and Sobierajski 1996] developed a haptic interface for sculpting, [Ferley et al. 2000] introduced new sculpting tools and improved interactive rendering, and [Cutler et al. 2002] provide procedural methods for defining volumetric models. Physical behavior has been added to the underlying volumetric model in order to produce virtual clay [Arata et al. 1999]. [McDonnell et al. 2001] improved upon this work by representing the virtual clay with subdivision solids [MacCracken and Roy 1996]. More recently sculpting systems [Perry and Frisken 2001; Ferley et al. 2001] have been based on octree representations [Meagher 1982; Frisken et al. 2000], allowing for volumetric models with adaptive resolution.

There exists a large body of surface editing work based on implicit models [Bloomenthal et al. 1997]. This approach uses implicit surface representations of analytic primitives or skeletal offsets. The implicit modeling work most closely related to ours is found in [Wyvill et al. 1999]. They describe techniques for performing blending, warping and boolean operations on skeletal implicit surfaces. An interesting variation of implicit modeling is presented by [Raviv and Elber 2000], who use a forest of trivariate functions [Casale and Stanton 1985] evaluated on an octree to create a multiresolution sculpting capability.

Level set methods have been successfully applied in computer graphics, computer vision and visualization [Sethian 1999; Sapiro 2001; Osher and Fedkiw 2002], for example medical image segmentation [Malladi et al. 1995; Whitaker et al. 2001], shape morphing [Desbrun and Cani 1998; Breen and Whitaker 2001], 3D reconstruction [Museth et al. 2002b; Whitaker 1998; Zhao et al. 2001], volume sculpting [Baerentzen and Christensen 2002], and the animation of liquids [Enright et al. 2002].

Our work stands apart from previous work in several ways. We have not developed volumetric modeling tools. Our editing system acts on surfaces that happen to have an underlying volumetric representation, but are based on the mathematics of deforming implicit surfaces. In our system voxels are not directly modified by the user, instead voxel values are determined numerically by solving the level set equation, based on user input. Since level set models are not tied to any specific implicit basis functions, they easily represent complex models to within the resolution of the sampling. Our work is the first to utilize level set methods to perform a variety of interactive editing operations on complex geometric models.

It should also be noted that several of the algorithms described in this paper have been implemented in graphics hardware, e.g. solving level set equations [Rumpf and Strzodka 2001a; Lefohn et al. 2003], evaluating other types of differential equations [Rumpf and Strzodka 2001b; Bolz et al. 2003; Sherbondy et al. 2003], morphological operators [Hopf and Ertl 2000; Yang and Welch 2002], and voxelization [Fang and Chen 2000; Sigg et al. 2003]. This work predominantly focuses on coping with the issues that arise from mapping general algorithms onto hardware-specific GPUs with restrictive memory sizes, data types and instruction sets in order to shorten computation times.

2 Level Set Models

Level set models implicitly represent a deforming surface as an iso-surface (or level set),

$$\mathcal{S} = \{\boldsymbol{x} \mid \phi(\boldsymbol{x}) = k\}\,, \tag{1}$$

where $k \in \mathbb{R}$ is the iso-value, $\boldsymbol{x} \in \mathbb{R}^3$ is a point in space on the iso-surface and $\phi : \mathbb{R}^3 \rightarrow \mathbb{R}$ is an arbitrary scalar function. To allow for deformations of the level set surface we assume that S can change over time. Introducing time dependence into the right-hand side of Eq. (1) produces two distinct types of level set surface representations. In the first, the iso-value k can be considered time-dependent and the level set function ϕ is only implicitly time-dependent, leading to the so called *static level set formulation*,

$$\mathcal{S}(t) = \{\boldsymbol{x}(t) \mid \phi(\boldsymbol{x}(t)) = k(t)\}\,. \tag{2}$$

In the second, the iso-value k is fixed in time and the level set function explicitly depends on time, leading to the *dynamic level set formulation*,

$$\mathcal{S}(t) = \{\boldsymbol{x}(t) \mid \phi(\boldsymbol{x}(t), t) = k\}\,. \tag{3}$$

These two level set formulations are *not* equivalent and offer very distinct advantages and disadvantages. A detailed discussion of level set methods is beyond the scope of this paper, but since both formulations play important roles in our work each is briefly described. For more details we refer the interested reader to [Osher and Fedkiw 2002; Sethian 1999].

2.1 Equation Formulations

The static formulation of Eq. (2) describes the deforming surface as a family of level sets, $\mathcal{S}(t)$, of a static function $\phi(\boldsymbol{x})$. The corresponding equation of motion for a point, $\boldsymbol{x}(t)$, on the surface is

easily derived by differentiating both sides of $\phi(\boldsymbol{x}(t)) = k(t)$ with respect to time t, and applying the chain rule giving:

$$\nabla\phi(\boldsymbol{x}(t)) \cdot \frac{\mathrm{d}\boldsymbol{x}(t)}{\mathrm{d}t} = \frac{\mathrm{d}k(t)}{\mathrm{d}t}. \tag{4}$$

Before interpreting this equation it is first necessary to define the term *level set speed function*. Throughout this paper we assume a *positive-inside/negative-outside* sign convention for ϕ, i.e. normal vectors, $\boldsymbol{n}$, of any level set of ϕ *point outwards* and are simply given by $\boldsymbol{n} \equiv -\nabla\phi/|\nabla\phi|$. This allows us to define the following *speed function*

$$\mathcal{F}(\boldsymbol{x}, \boldsymbol{n}, \phi, \dots) \equiv \boldsymbol{n} \cdot \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = -\frac{\nabla\phi}{|\nabla\phi|} \cdot \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t}, \tag{5}$$

which in general is a user-defined scalar function that can depend on any number of variables including $\boldsymbol{x}$, $\boldsymbol{n}$, ϕ and its derivatives evaluated at $\boldsymbol{x}$, as well as a variety of external data inputs. The geometric interpretation of this function is straightforward; since $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$ denotes the velocity vectors of a point $\boldsymbol{x}$ on a level set surface, the speed function $\mathcal{F}$ defines the projection of this vector onto the local surface normal. In other words, $\mathcal{F}$ is a *signed* scalar function that defines the motion (i.e. speed) of the level set surface in the direction of the local normal $\boldsymbol{n}$ at a point $\boldsymbol{x}$ on a level set $\mathcal{S}$.

Using the definition of a speed function in Eq. (5), Eq. (4) may be simplified to

$$|\nabla\phi|\mathcal{F} = -\frac{\mathrm{d}k}{\mathrm{d}t}. \tag{6}$$

First we note that since the right hand side only depends on time, so does the sign of the speed function, $\mathcal{F}$. Consequently the static level set formulation only allows for monotonic surface motions at any given time t. Secondly we note that Eq. (6) actually includes two unknown and explicitly time-dependent functions, $\mathcal{F}$ and k. Whereas the presence of a general functional expression for $k(t)$ adds a little extra flexibility to this formulation, it also makes the users task of defining a speed function less intuitive since it generally depends on the specific choice of $k(t)$. Fortunately we can easily address this issue by limiting ourselves to a simple special case of Eq. (2) where $\phi(\boldsymbol{x}) \equiv \pm t$, i.e $\phi(\boldsymbol{x})$ is defined to be the (forward or backward) time of arrival of the level set surface at a point $\boldsymbol{x}$. With this trivial choice of $k(t) = \pm t$, Eq. (6) simplifies to

$$|\nabla\phi|\mathcal{F} = \pm 1. \tag{7}$$

which only depends implicitly on time and therefore describes a simple boundary value problem. Eq. (7) is known as the *fundamental stationary level set equation* and can be efficiently solved using fast marching methods [Tsitsiklis 1995; Sethian 1996], which will be described in detail in Section 3.1.6. We point out that the so-called *Eikonal equation*

$$|\nabla\phi| = 1, \tag{8}$$

which produces a *signed distance field* to an initial surface $\mathcal{S}_0$, can be considered a special case of the stationary level set equation (Eq. (7)) with a unit speed function and the boundary condition $\{\boldsymbol{x} \in \mathcal{S} | \phi(\boldsymbol{x}) = 0\}$. Therefore fast marching methods may be used to efficiently compute the signed distance field to an arbitrary (closed and orientable) surface.

The stationary level set representation has a significant limitation, however. It follows directly from Eq. (7) that the speed functions, $\mathcal{F}$, has to be strictly positive or negative depending on the sign of the right-hand side. Consequently surface deformations are limited to strict monotonic motions - always inward or outward, similar to the layers of an onion. This limitation stems from the fact that $\phi(\boldsymbol{x})$ by definition has to be single-valued (time-of-arrival), i.e. the level set surface resulting from the stationary formulation cannot self-intersect over time. The inherit limitation of the static formulation can be overcome by adding an explicit time-dependence to ϕ, which leads to the dynamic level set Eq. (3).

Following the same steps as the stationary case, the dynamic equation of motion may be derived by differentiating the right-hand side of Eq. (3) with respect to time and applying the speed function definition (Eq. (5)), giving

$$\frac{\partial\phi}{\partial t} = -\nabla\phi \cdot \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} \tag{9a}$$

$$= |\nabla\phi|\, \mathcal{F}(\boldsymbol{x}, \boldsymbol{n}, \phi, \dots). \tag{9b}$$

These fundamental equations are referred to as "the level set equations" in the literature, even though they are strictly speaking the dynamic counterpart to the stationary level set Eq. (7). In contrast to the stationary form the dynamic surface representation of Eq. (9b) does not limit the sign of the speed function $\mathcal{F}$, and therefore allows for arbitrary surface deformations. The speed function is usually based on a set of geometric measures of the implicit level set surface and data inputs. The challenge when working with level set methods is determining how to combine these components to produce a local motion that creates a desired global or regional surface structure. [Museth et al. 2002a] defines several such speed functions that may be used to edit geometric objects.

2.2 Geometric Properties

The speed function, $\mathcal{F}$, introduced in the previous section typically depends on different geometric properties of the level set surface. These properties can conveniently be expressed as either zero, first or second order derivatives of ϕ. Examples include the shortest distance from an arbitrary point to the surface, the local surface normal and different curvature measures. Assuming ϕ is properly normalized, i.e. satisfies Eq. (8), the distance is simply the numerical value of ϕ, and as indicated above the normal vector is just a normalized gradient of ϕ. The latter is easily proved by noting that the directional derivative in the tangent plane of the level set function by definition vanishes, i.e.

$$\frac{\mathrm{d}\phi}{\mathrm{d}\boldsymbol{T}} \equiv \boldsymbol{T} \cdot \nabla\phi = 0 \tag{10}$$

where $\boldsymbol{T}$ is an arbitrary unit vector in the tangent plane of the level set surface. We have many different curvature measures for surfaces, but as has been noted by others geometric flow based on the mean curvature seems to be most useful. From the definition of the mean curvature in differential geometry [do Carmo 1976] we have

$$K \equiv K_1 + K_2 \equiv \mathrm{Div}_{\boldsymbol{e}_1}[\boldsymbol{n}] + \mathrm{Div}_{\boldsymbol{e}_2}[\boldsymbol{n}] \tag{11a}$$

$$= \boldsymbol{e}_1(\boldsymbol{e}_1 \cdot \nabla) \cdot \boldsymbol{n} + \boldsymbol{e}_2(\boldsymbol{e}_2 \cdot \nabla) \cdot \boldsymbol{n} \tag{11b}$$

where $\{K_1, K_2\}$ are the principle curvatures and $\mathrm{Div}_{\boldsymbol{e}_1}[\boldsymbol{n}]$ denotes the divergence of the normal vector $\boldsymbol{n}$ in the principle direction $\boldsymbol{e}_1$. Next, resolving the gradient operator in the orthonormal frame of the principle directions $\{\boldsymbol{e}_1, \boldsymbol{e}_2\}$ in the tangent plane and the normal vector $\boldsymbol{n}$ gives

$$\nabla = \boldsymbol{e}_1(\boldsymbol{e}_1 \cdot \nabla) + \boldsymbol{e}_2(\boldsymbol{e}_2 \cdot \nabla) + \boldsymbol{n}(\boldsymbol{n} \cdot \nabla). \tag{12}$$

Eq. (11b) simplifies to

$$K = \nabla \cdot \boldsymbol{n} - \boldsymbol{n}(\boldsymbol{n} \cdot \nabla) \cdot \boldsymbol{n} \tag{13a}$$

$$= \nabla \cdot \boldsymbol{n} = \nabla \cdot \frac{\nabla\phi}{|\nabla\phi|}, \tag{13b}$$

Figure 2: A slice through a narrow band distance volume.

where we have also made used of the following relations

$$\boldsymbol{n}(\boldsymbol{n}\cdot\nabla)\cdot\boldsymbol{n} = \sum_j \boldsymbol{n}_j \sum_i \boldsymbol{n}_i \nabla_i \boldsymbol{n}_j = \sum_{ij} \boldsymbol{n}_i \boldsymbol{n}_j \nabla_i[\boldsymbol{n}_j] \quad (14a)$$

$$= \sum_{ij} \boldsymbol{n}_i \frac{1}{2} \nabla_i[\boldsymbol{n}_j^2] = \frac{1}{2} \sum_i \boldsymbol{n}_i \nabla_i[\sum_j \boldsymbol{n}_j^2] \quad (14b)$$

$$= \frac{1}{2} \sum_i \boldsymbol{n}_i \nabla_i[1] = 0, \quad (14c)$$

since the normal vector is always normalized to one. Eq. (13b) can finally be expanded to obtain an expression directly in terms of derivatives of ϕ

$$K = \begin{array}{c} \left(\phi_x^2(\phi_{yy}+\phi_{zz}) - 2\phi_x\phi_y\phi_{xy} + \right. \\ \phi_y^2(\phi_{xx}+\phi_{zz}) - 2\phi_x\phi_z\phi_{xz} + \\ \left. \phi_z^2(\phi_{xx}+\phi_{yy}) - 2\phi_y\phi_z\phi_{yz}\right) / \\ \left(\phi_x^2+\phi_y^2+\phi_z^2\right)^{3/2}. \end{array} \quad (15)$$

Eq. (15) clearly reduces the problem of computing the mean curvature of a level set surfaces to the simple approximation of first and second order derivatives of the level set function itself. In section 3.2.2 we shall also discuss a convenient and fast way to compute the mean curvature, as well as other types of surface curvature, directly from Eq. (13b).

3 Types of Computation

The main algorithms employed by our level set modeling system may be placed in three categories: distance computations, level set evolutions and efficient mesh extractions.

3.1 Distance Computations

A level set model is represented by a distance volume, a volume dataset where each voxel stores the shortest distance to the surface of the object being represented by the volume. The inside-outside status of the point is defined by its sign, positive for inside and negative for outside. Since we are only interested in one level set (iso-surface) embedded in the volume, distance information is only maintained around one level set (usually of iso-value zero). Depending on the accuracy of the integrations scheme this "narrow band" is typically only a five voxels wide (two voxels on each side of the zero level set). See Figure 2.

Before an object can be edited in our system, it must first be converted into a narrow-band distance volume. Currently we are able to convert polygonal, NURBS, implicit and CSG models, as well as general volumetric models into the appropriate volumetric format. The fundamental operation performed in the conversion process is the calculation of the shortest distance from an arbitrary point to the geometric model being scan converted. Since the calculation is performed repeatedly, efficient computation is essential to minimizing the time needed for conversion.

3.1.1 Narrow Band Approximation

All of our level set editing operators assume that our models are represented as "narrow-band" distance volumes. Unfortunately, our operators do not necessarily produce this representation, signed distance in a narrow band and constant values outside of the band.[1] The level set equation (Eq. (9b)) contains no explicit constraints that maintain ϕ as a signed distance function as time evolves. In fact it can be shown that ϕ will only remain a distance field for certain restricted types of speed functions [Zhao et al. 1996; Sapiro 2001]. Additionally, the CSG operations used extensively in our editing system are known not to produce true distance values for all circumstances [Perry and Frisken 2001; Breen et al. 2000]. We must therefore reset the volumetric representation of our models after each editing operation in order to ensure that ϕ is approximately equal to the shortest distance to the zero level set in the narrow band.

The re-normalization after each editing operation can be implemented in a number of ways. One option is to directly solve the Eikonal equation, $|\nabla\phi| = 1$ using algorithms such as Sethian's Fast Marching Method[Sethian 1996]. Alternatively one can solve the following time-dependent Hamilton-Jacobi equation until it reaches a steady state,

$$\frac{\mathrm{d}\phi}{\mathrm{d}t} = S(\phi)(1 - |\nabla\phi|), \quad (16)$$

where $S(\phi)$ returns the sign of ϕ [Peng et al. 1999]. Both approaches are accurate but in the context of an interactive implementation they are too slow. Instead we use a faster but approximate solution where points on the zero level set (iso-surface) of the embedded surface are found by linearly interpolating the voxel values along grid edges that span the zero crossings. These "zero-crossing" edges have end-points (voxels) whose associated ϕ values have opposite signs. The first step in rebuilding ϕ in the narrow band after an editing operation consists of creating the list of "active" voxels, those adjacent to a zero crossing. The values at these voxels are then recalculated with a first-order Newton's approximation [Whitaker 1998], $\phi_{new}(\boldsymbol{x}) = \phi_{old}(\boldsymbol{x})/|\nabla\phi_{old}(\boldsymbol{x})|$, which is only valid near the zero level set.

The ϕ values of the next N layers of voxels that form a narrow band on either side of the active list voxels are approximated by a simple city block distance metric. First, all of the voxels that are adjacent to the active list voxels are found. They are assigned a ϕ value that is one plus the smallest ϕ value of their 6-connected neighbors in the active list. Next all of the voxels that are adjacent to the first layer, but not in the active list are identified and their ϕ values are set to be one plus the smallest value of their 6-connected neighbors. This process continues until a narrow band ℓ voxels thick has been created.

3.1.2 Scan Conversion of Polygonal Models

This section describes an algorithm for calculating a distance volume from a 3D closed, orientable polygonal mesh composed of

[1] They do properly produce the correct zero crossings in the resulting volumes.

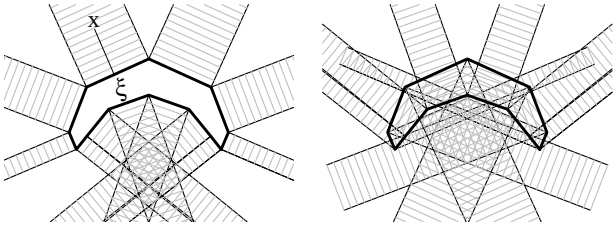


Figure 3: Strips containing points with negative (left) and positive (right) distance to edges.

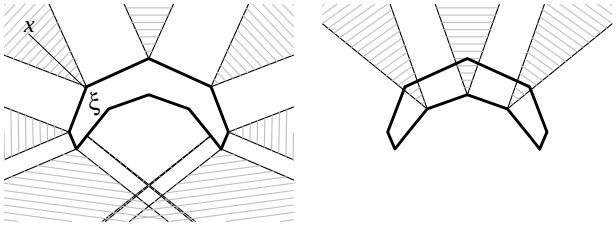


Figure 4: Wedges containing points with negative (left) and positive (right) distance to vertices.

triangular faces, edges, vertices, and normals pointing outwards. The algorithm computes the closest point on and shortest signed distance to the mesh by solving the Eikonal equation, $|\nabla\phi| = 1$, derived in Section 2, by the method of characteristics. The method of characteristics is implemented[2] efficiently with the aid of computational geometry and polyhedron scan conversion producing an algorithm with computational complexity that is linear in the number of faces, edges, vertices and voxels [Mauch 2003; Mauch 2004].

Let $\boldsymbol{\xi}$ be the closest point on a manifold to the point $\mathbf{x}$. The distance to the manifold is $|\mathbf{x} - \boldsymbol{\xi}|$. $\mathbf{x}$ and $\boldsymbol{\xi}$ are the endpoints of the line segment that is a characteristic of the solution of the Eikonal equation. If the manifold is smooth then the line connecting $\mathbf{x}$ to $\boldsymbol{\xi}$ is orthogonal to the manifold. If the manifold is not smooth at $\boldsymbol{\xi}$ then the line lies "between" the normals of the smooth parts of the manifold surrounding $\boldsymbol{\xi}$.

Based on this observation, a Voronoi diagram is built for the faces, edges and vertices of the mesh, with each Voronoi cell defined by a polyhedron. Scan conversion is then utilized to determine which voxels of the distance volume lie in each Voronoi cell. By definition the face, edge or vertex associated with the Voronoi cell is the closest element on the mesh to the voxels in the cell. The closest point/shortest distance to the element is then calculated for each voxel.

Suppose that the closest point $\boldsymbol{\xi}$ to a grid point $\mathbf{x}$ lies on a triangular face. The vector from $\boldsymbol{\xi}$ to $\mathbf{x}$ is orthogonal to the face. Thus the closest points to a given face must lie within a triangular prism defined by the edges and normal vector of the face. Faces produce prisms of both positive and negative distance depending on their relationship to the face's normal vector. The sign of the distance value in the prism in the direction of the normal (outside the mesh) is negative and is positive opposite the normal (inside the mesh). A 2D example is presented in Figure 3. In two dimensions the Voronoi cells are defined as strips with negative and positive distance.

Consider a grid point $\mathbf{x}$ whose closest point $\boldsymbol{\xi}$ is on an edge. Each

[2]The scan conversion algorithm presented here is available in the CPT library at http://www.acm.caltech.edu/∼seanm/software/cpt/cpt.html.

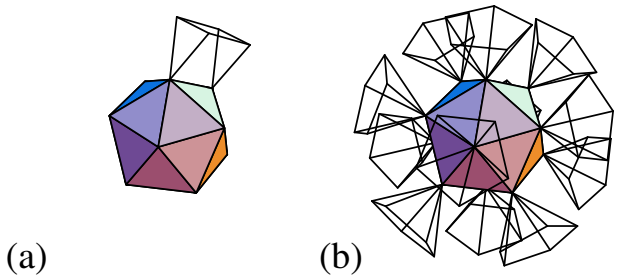


Figure 5: (a) The polyhedron for a single edge. (b) The polyhedra for the vertices.

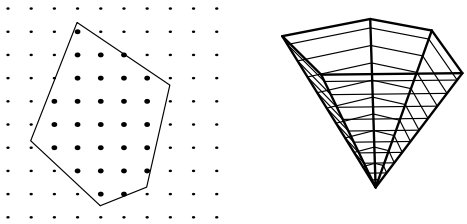


Figure 6: Scan conversion of a polygon in 2D. Slicing a polyhedron to form polygons.

edge in the mesh is shared by two faces. The closest points to an edge must lie in a wedge defined by the edge and the normals of the two adjacent faces. We define only one Voronoi cell for each edge in the direction where the angle between the faces is greater than π. Finally, consider a grid point $\mathbf{x}$ whose closest point $\boldsymbol{\xi}$ is on a vertex. Each vertex in the mesh is shared by three or more faces. The closest points to a vertex must lie in a faceted cone defined by the normals to the adjacent faces. Similar to the edge Voronoi cells, we only define one polyhedron for each vertex. The cone will point outwards and contain negative distance if the surface is convex at the vertex. The cone will point inwards and contain positive distance if the surface is concave at the vertex. Figure 4 may be thought of as a 2D cross-section of an edge or a vertex Voronoi cell and demonstrates the conditions for defining one positive or negative polyhedron. Figure 5a shows a Voronoi cell (polyhedron) for a single edge. Figure 5b shows all of the vertex polyhedra of an icosahedron.

Once the Voronoi diagram is constructed, the polyhedra associated with each cell is scan converted in order to associate the closest face, edge or vertex with each voxel for the shortest distance calculation. Each polyhedron is intersected with the planes that coincide with the grid rows to form polygons. This reduces the problem to polygon scan conversion. See Figure 6. For each grid row that intersects the resulting polygon we find the left and right intersection points and mark each grid point in between as being inside the polygon. The polyhedra that define the Voronoi cells must be enlarged slightly to make sure that grid points are not missed due to finite precision arithmetic. Therefore, some grid points may be scan converted more than once. In this case, the smaller distance and thus the closer point is chosen.

3.1.3 Superellipsoids

Superellipsoids are used as modeling primitives and region-of-influence (ROI) primitives for some of our operators. In both cases,

a scan-converted representation is needed. The parametric equation for a superellipsoid is

$$\mathbf{S}(\eta,\omega) = \begin{bmatrix} a_1 cos^{\epsilon 1}(\eta) cos^{\epsilon 2}(\omega) \\ a_2 cos^{\epsilon 1}(\eta) sin^{\epsilon 2}(\omega) \\ a_3 sin^{\epsilon 1}(\eta) \end{bmatrix} \quad (17)$$

where $\eta \in [-\pi/2, \pi/2]$ and $\omega \in [-\pi, \pi]$ are the longitudinal and latitudinal parameters of the surface, a_1, a_2, a_3 are the scaling factors in the X, Y, and Z directions, and $\epsilon 1$ and $\epsilon 2$ define the shape in the longitudinal and latitudinal directions [Barr 1981].

The distance to a point on the surface of a superellipsoid defined at $[\eta, \omega]$ from an arbitrary point $\mathbf{P}$ is

$$d(\eta,\omega) = ||\mathbf{S}(\eta,\omega) - \mathbf{P}||. \quad (18)$$

Squaring and expanding Eq. (18) gives

$$\begin{aligned} \hat{d}(\eta,\omega) = &(a_1 cos^{\epsilon 1}(\eta) cos^{\epsilon 2}(\omega) - P_x)^2 \\ &+ (a_2 cos^{\epsilon 1}(\eta) sin^{\epsilon 2}(\omega) - P_y)^2 \\ &+ (a_3 sin^{\epsilon 1}(\eta) - P_z)^2. \end{aligned} \quad (19)$$

The closest point to the superellipsoid from an arbitrary point $\mathbf{P}$ can then be calculated by determining the values of $[\eta, \omega]$ which minimize Eq. (19). In general Eq. (19) is minimized with a gradient descent technique utilizing variable step-sizes. The values of $[\eta, \omega]$ may then be plugged into Eq. (17) to give the closest point on the surface of the superellipsoid, which in turn may be used to calculate the shortest distance.

Finding the values of η and ω at the closest point with a gradient descent technique involves calculating the gradient of Eq. (19),

$$\nabla \hat{d} = [\partial \hat{d}/\partial\eta, \partial \hat{d}/\partial\omega]. \quad (20)$$

Unfortunately, superellipsoids have a tangent vector singularity near $[\eta, \omega]$ values that are multiples of $\pi/2$. To overcome this problem, we re-parameterize $\mathbf{S}$ by arc length [do Carmo 1976]. Once our steepest descent (on $\hat{d}$) is redefined so that it is steepest with respect to the normalized parameters (α, β) we can use the gradient of the re-parameterized $\hat{d}$,

$$\nabla d' = [\partial \hat{d}/\partial\alpha, \partial \hat{d}/\partial\beta], \quad (21)$$

to find the closest point with greater stability. For more details see [Breen et al. 2000].

The general formulation of Eq. (21) significantly simplifies for values of η and ω near multiples of $\pi/2$. Instead of deriving and implementing these simplifications for all regions of the superellipsoid the calculation is only performed in the first octant ($0 \le \eta \le \pi/2, 0 \le \omega \le \pi/2$). Since a superellipsoid is 8-way symmetric, point $\mathbf{P}$ may be reflected into the first octant, the minimization performed, and the solution point reflected back into $\mathbf{P}$'s original octant.

It should be noted that for certain values of $\epsilon 1$ and $\epsilon 2$ the normals of a superellipsoid become discontinuous, producing special degenerate primitives that must be dealt with separately. The most common cases are the cuboid ($\epsilon 1 = \epsilon 2 = 0$), and the cylinder ($\epsilon 1 = 0, \epsilon 2 = 1$). The shortest distance to these primitives may be determined by calculating the shortest to each individual face (6 for the cuboid, 3 for the cylinder), and choosing the smallest value.

A faster, but less accurate, alternative for scan-converting any implicit primitive involves utilizing the approximation from Section 3.1.1 at the voxels adjacent to the primitive's surface. Given these voxel values, the distance values at the remaining voxels may be calculated with a Fast Marching Method [Sethian 1996; Tsitsiklis 1995]. See Section 3.1.6. Also, once shortest distance can be calculated for any closed primitive, distance to a Constructive Solid Geometry (CSG) model consisting of combinations of the primitive may also be computed [Breen et al. 2000].

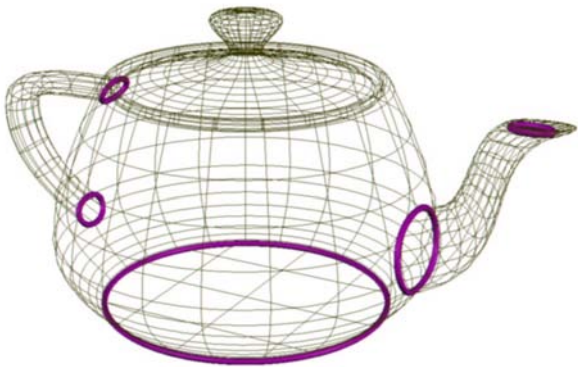


Figure 7: A trimmed NURBS teapot model. The trimming curves remove portions of each surface's domain and maintain topological connectivity between adjacent surfaces.

3.1.4 Trimmed NURBS Models

A trimmed NURBS model has portions of its domain, and thus portions of the surface, trimmed away [Shantz and Chang 1988; Piegl and Tiller 1998]. The trimming data structure is commonly a piecewise linear curve in the parameter space and a companion piecewise curve in the space of the surface. A set of trimmed surfaces may be joined together into a solid model with topological connectivity maintained by the trimming curves (Figure 7). Our approach to converting a trimmed NURBS model consists of three stages: 1) compute the minimum distance to the Euclidean trimming curves, 2) compute local distance minima to the NURBS surface patches, discarding solutions that lie outside the trimmed domain, and 3) perform an inside/outside test on the resulting closest point.

Distance to Trimming Curves Models typically contain thousands of trimming segments and computing the closest point on these segments is very similar to finding the minimum distance to polygonal models, except the primitives are line segments instead of triangles. We have modified the publically available PQP package [3], which computes swept sphere volume hierarchies around triangulated models, to use line segments. This reduces the query time for the distance to trimming loops from O(n) closer to O(logn).

Local Distance to NURBS Surfaces Similar to a superquadric the distance between a point and parametric surface is described by

$$\mathbf{D}^2(u,v) = ||\mathbf{S}(u,v) - \mathbf{P}||^2. \quad (22)$$

Minimizing Equation (22) corresponds to finding the parameter values of the local closest point on that surface and can be done by finding the simultaneous roots of the partial derivatives of $\mathbf{D}^2(u,v)$,

$$(\mathbf{S}(u,v) - \mathbf{P}) \cdot \mathbf{S_u} = 0 \quad (23)$$

$$(\mathbf{S}(u,v) - \mathbf{P}) \cdot \mathbf{S_v} = 0 \quad (24)$$

We search for local minima in distance until the closest local minimum inside the trimmed domain is found. Multi-dimensional Newton's method can quickly find a local minimum when given a reasonable starting point. However, Newton's method does not always converge. For robustness, we use Newton's method at multiple starting locations around a potential local minimum. As a preprocess, each original polynomial span of the model's original

[3] http://www.cs.unc.edu/∼geom/SSV/

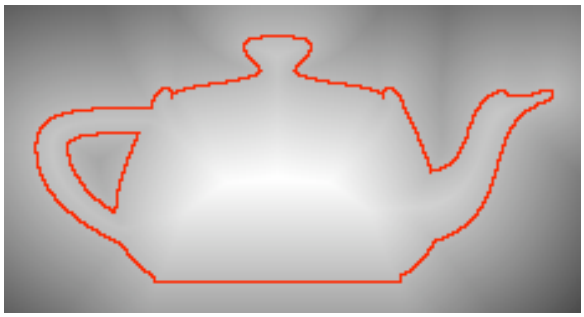


Figure 8: A slice through a 232x156x124 distance volume of the Utah teapot. The zero level set is highlighted in red.

Figure 9: An embossed level set teapot model.

surfaces are refined into a new sub-surfaces. These refined sub-surfaces provide multiple starting locations for each potential minimum.

The algorithm initializes Newton's method by projecting the query point onto the control polygon of the tested surface. This projection is used to compute a surface point using *nodal mapping*. Nodal mapping associates a parameter value (the node) to each control point of that surface piece, and then linearly interpolates between node values using the projection onto the control mesh. Evaluating the surface at this interpolated value produces a first order approximation to the closest point on the surface and a reasonable starting value for Newton's method to improve.

The closest point returned by Newton's method may be outside of the trimmed domain. Again, a modified PQP algorithm for planar line segments is used to find the closest point on the parametric trimming segments of that surface. Valid parametric solutions are to the right of the closest trimming line segment. Average normals are stored at the vertices of the trimming curve in order to correctly perform the inside/outside domain test.

The closest valid point on the surface is compared with the distance to the spatial representation of the trimming loops, and the closest is used as the closest point on the model. If the closest point is on a surface patch, the inside/outside status of the query is determined by dotting the vector from the query point to the closest point with the surface normal at the closest point. The sign of the dot product gives the sign of the distance. If the closest point is on a trimming loop, we use a classic ray shooting algorithm and count the number of crossing of the model to determine whether the query point is inside or outside.

Acceleration Techniques NURBS surfaces have a local convex hull property when the homogeneous coordinate of the control points have positive values. These convex hulls provide a natural means of computing a lower bound on the minimum distance from the query point to the contained portion of surface. We use the GJK package[4] as a robust implementation of Gilbert's algorithm [Gilbert et al. 1988; Cameron 1997] to efficiently compute a lower bound on distance. Refined surfaces with a lower distance bound larger than the current minimum distance cannot contribute a closer point, so they can be ignored.

We note that the minimum distance from one query to the next cannot vary more than the distance between the two query points. We therefore initialize the minimum distance for a new query as the last minimum distance plus the space between query points. This helps to quickly remove surfaces to be tested using the convex hull technique. Finally, the ray intersection inside/outside test can be accelerated by noting that if the last query point was outside, then the new query cannot be inside if the last minimum distance was larger than distance between query points, and vice versa when the last query was inside. This very efficiently removes the need to test the sign of many queries. Figure 8 presents a slice from a signed distance volume produced from a trimmed NURBS models containing 20 patches and four trimming curves. It was calculated in ## minutes.

3.1.5 Point Sets

Some level set editing operators need to determine the closest point in a set from another arbitrary point. This capability is used during level set blending (when calculating the distance to an intersection "curve") and embossing (moving a level set surface towards a point set). We utilize the ANN library of Mount and Arya.[5] The library calculates closest point queries of a point set in $\mathcal{O}(\log N)$ time by first storing the point set in a hierarchical data structure that partitions the space around the point set into non-overlapping cells. Given an input point, the hierarchical structure is traversed and candidate cells are identified and sorted [Arya et al. 1998]. A priority search technique is then utilized to find the closest point (within some tolerance ϵ) in the list of candidate cells [Arya and Mount 1993]. When the points are uniformly distributed, we have found that storing the point set in a K-D tree [de Berg et al. 1997] provides the best performance. For clustered points, storing the point set in the *balanced box decomposition (BBD) tree* described in [Arya et al. 1998] produces the fastest result.

3.1.6 Fast Marching Method

We utilize a Fast Marching Method (FMM) to generate distance volumes when given distance values only at voxels immediately adjacent to the zero level set. This can occur when scan-converting implicit primitives, and generating distance volumes from a level set segmentation [Whitaker et al. 2001]. The FMM is also used to calculate the distance values needed for our morphological operators.

The solution of the Eikonal Eq. (8) with the boundary condition $\phi|_S = 0$ (a zero level set) is the distance from the manifold S. The characteristics of the solution are straight lines which are orthogonal to S. We call the direction in which the characteristics propagate the *downwind* direction. More than one characteristic may reach a given point. In this case the solution is multi-valued. One

[4] URL: http://web.comlab.ox.ac.uk/oucl/work/stephen.cameron/distances/

[5] URL: http://www.cs.umd.edu/∼mount/ANN

can obtain a single-valued weak solution by choosing the smallest of the multi-valued solutions at each point. This is a weak solution because ϕ is continuous, but not everywhere differentiable. The equation may be efficiently and directly solved by ordering the grid points of the volume, so that information is always propagated in the direction of increasing distance. This is *the Fast Marching Method* [Sethian 1996]. It achieves a computational complexity of $\mathcal{O}(N \log N)$.

The Fast Marching Method is similar to Dijkstra's algorithm [Dijkstra 1959; Cormen et al. 2001] for computing the single-source shortest paths in a weighted, directed graph. In solving this problem, each vertex is assigned a distance, which is the sum of the edge weights along the minimum-weight path from the source vertex. As Dijkstra's algorithm progresses, the status of each vertex is either *known*, *labeled* or *unknown*. Initially, the source vertex in the graph has *known* status and zero distance. All other vertices have *unknown* status and infinite distance. The source vertex labels each of its adjacent neighbors. A *known* vertex labels an adjacent vertex by setting its status to *labeled* if it is *unknown* and setting its distance to be the minimum of its current distance and the sum of the *known* vertices' weight and the connecting edge weight. It can be shown that the labeled vertex with minimum distance has the correct value. Thus the status of this vertex is set to *known*, and it labels its neighbors. This process of freezing the value of the minimum labeled vertex and labeling its adjacent neighbors is repeated until no labeled vertices remain. At this point all the vertices that are reachable from the source have the correct shortest path distance. The performance of Dijkstra's algorithm depends on quickly determining the labeled vertex with minimum distance. One can efficiently implement the algorithm by storing the labeled vertices in a binary heap. Then the minimum labeled vertex can be determined in $\mathcal{O}(\log n)$ time where n is the number of labeled vertices.

Sethian's Fast Marching Method differs from Dijkstra's algorithm in that a finite difference scheme is used to label the adjacent neighbors when a grid point becomes known. If there are N grid points, the labeling operations have a computational cost of $\mathcal{O}(N)$. Since there may be at most N labeled grid points, maintaining the binary heap and choosing the minimum labeled vertices makes the total complexity $\mathcal{O}(N \log N)$.

3.2 Solving the Level Set Equation

Several editing operators modify geometric objects, represented by volume datasets (a 3D grid), by evolving the level set partial differential equation (PDE) (Eq. (9b)). As was first noted by Osher and Sethian [Osher and Sethian 1988] this PDE can be solved using finite difference (FD) schemes originally developed for Hamilton-Jacobi type equations. This corresponds to discretizing Eq. (9b) on a regular 3D spatial grid and a 1D temporal grid. The use of such grids raises a number of numerical and computational issues that are important to the accuracy and stability of the implementation. The two central issues are the proper choice of a numerical integration scheme with respect to time, and the development of an appropriate narrow band algorithm for localizing computation in the spatial dimensions. The details of these schemes/algorithms will ultimately affect the *stability, accuracy and efficiency* of the system.

There exists a large number of so-called implicit and explicit integrations schemes that can be used to propagate Eq. (9b) forward in time [Burden and Faires 2001]. The implicit schemes have the advantage of being unconditionally stable with respect to the time discretization, but typically at the cost of large truncation errors. They also require massive matrix manipulations which make them hard to implement and more importantly increase the computation time per time step. This is in strong contrast to the explicit methods that are relatively simple to set up and program. Unfortunately explicit schemes often have stability constraints on their time discretization given a certain space discretization. The exceptions to this rule are the so-called semi-Lagrangian integration schemes that can the considered unconditionally stable explicit schemes. However is it unclear how to extend semi-Lagrangian schemes to the general class of PDEs that we are dealing with here.

It is our experience that for the level set problems considered in this paper the stability constraints associated with a simple explicit integration scheme like the "forward Euler method"

$$u^{n+1}_{i,j,k} = u^n_{i,j,k} + \Delta t \Delta u^n_{i,j,k}, \tag{25}$$

offer a very good balance of speed, fast update times and simplicity. In this equation u^n denotes the approximation of $\phi(\boldsymbol{x}, t)$ at the nth discrete time step, Δt is a time-increment that is chosen to ensure stability, and $\Delta u^n_{i,j,k}$ is the discrete approximation to $\partial\phi/\partial t$ evaluated at grid point $\boldsymbol{x}_{i,j,k}$ and time-step t_n. We shall assume, without a loss in generality, that the grid spacing is unity. The initial conditions u^0 are established by the scan conversion algorithms discussed in the previous sections and the boundary conditions produce zero derivatives toward the outside of the grid (Neumann type).

The next step is to express the time-increment, $\Delta u^n_{i,j,k}$ of Eq. (25), in terms of the fundamental level set Eq. (9b)

$$\Delta u^n_{i,j,k} = \mathcal{F}(i,j,k) \left|\nabla u^n_{i,j,k}\right| \tag{26a}$$

$$\approx \mathcal{F}(i,j,k) \sqrt{\sum_{w \in x,y,z} \left(\delta_w u^n_{i,j,k}\right)^2} \tag{26b}$$

where $\delta_w u^n_{i,j,k}$ approximates $\partial u^n_{i,j,k}/\partial w$, i.e. the discretization of the partial derivative of u with respect to the spatial coordinate $w \in x, y, z$. The final step is to express these spatial derivatives as well as the speed function, $\mathcal{F}(i,j,k)$, in terms of finite differences (FD) on the spatial 3D grid. Many different FD schemes with varying stencil and truncation error exist, but to list the simplest we have

$$\frac{\partial u^n_{i,j,k}}{\partial w} = \delta^+_w u^n_{i,j,k} + O(\Delta w) \tag{27a}$$

$$= \delta^-_w u^n_{i,j,k} + O(\Delta w) \tag{27b}$$

$$= \delta^\pm_w u^n_{i,j,k} + O(\Delta w^2) \tag{27c}$$

where we have defined short-hand notations for the following FD expressions

$$\delta^+_x u^n_{i,j,k} = \frac{u^n_{i+1,j,k} - u^n_{i,j,k}}{\Delta x} \tag{28a}$$

$$\delta^-_x u^n_{i,j,k} = \frac{u^n_{i,j,k} - u^n_{i-1,j,k}}{\Delta x} \tag{28b}$$

$$\delta^\pm_x u^n_{i,j,k} = \frac{u^n_{i+1,j,k} - u^n_{i-1,j,k}}{2\Delta x} \tag{28c}$$

The explicit choice of an FD scheme for the first order derivatives in the term $\left|\nabla u^n_{i,j,k}\right|$ turns out to be closely related to the functional expression of the speed-term. This is a simple consequence of the fact that the the corresponding solutions to the level set PDE with different speed-functions can shown very different mathematical behavior. This is formulated more precisely by the so-called CFL condition which will be explained in more detail below. There are two important[6] classes of level set PDEs, namely hyperbolic and parabolic.

[6]The third type - the so-called elliptic PDEs - correspond to boundary value problems and are therefore not relevant to the initial-value problems that we are studying here.

3.2.1 Hyperbolic Speed Functions

For many level set deformations the speed function can, to some approximation, be assumed to be independent of the level set function itself. This is the case for the embossing operator described in [Museth et al. 2002a] or simple constant normal flow when performing surface dilation or erosion. This corresponds to the level set surface being *advected* in an external flow field generated by, for example, attraction forces to other geometry like a surface or a set of points. Such advection problems are common in computational fluid dynamics and the corresponding *hyperbolic* PDEs have the mathematical property of propagating information in certain *characteristic* directions. The explicit finite difference scheme used for solving the corresponding hyperbolic level set equations should be consistent with the information flow direction. Indeed, this is nothing more than requiring the numerical scheme to obey the underlying "physics" of the level set surface deformation.

This is formulated more precisely as the Courant-Friedrichs-Lewy (CFL) stability condition[Courant et al. 1928] that states that the domain of dependence of the discretized FD problem has to include the domain of dependence of the differential equation in the limit as the length of the FD steps goes to zero. Consequently the stencil used for the FD approximation of the spatial derivatives in Eq. (26a) should only include sample points (information) from the domain of dependence of the differential equation, i.e. from the side of the zero-crossing opposite to the direction in which it moves - or simply up-wind to the level set surface. This amounts to using an *up-wind scheme* that employs *single-sided* derivatives like those in Eqs. (28a) and (28b). The partial derivatives in the term $|\nabla\phi|$ of Eq. (26b) are computed using only those derivatives that are upwind relative to the movement of the level set. In our initial work we used the upwind scheme described in [Whitaker 1998], but we now use the more stable Godunov's method [Rouy and Tourin 1992]:

$$(\delta_w u^n_{i,j,k})^2 = \max(\min(\delta^+_w u^n_{i,j,k}, 0)^2, \max(\delta^-_w u^n_{i,j,k}, 0)^2) \quad (29)$$

$$\text{for } \mathcal{F}(i,j,k) \leq 0, \text{ and}$$

$$(\delta_w u^n_{i,j,k})^2 = \min(\max(\delta^+_w u^n_{i,j,k}, 0)^2, \min(\delta^-_w u^n_{i,j,k}, 0)^2) \quad (30)$$

$$\text{for } \mathcal{F}(i,j,k) > 0.$$

Another consequence of the CFL condition is that for the numerical FD scheme to be stable the corresponding numerical wave has to propagate at least as fast as the level set surface. Since the maximum surface motion is defined by the speed-function $\mathcal{F}(i,j,k)$ and the FD scheme (by definition) propagates the numerical information exactly one grid cell (defined by $\{\Delta x, \Delta y, \Delta z)\}$) per time iteration, an upper bound is effectively imposed on the numerical time steps, Δt in Eq. (25). This can be expressed in a conservative time step restriction[7]

$$\Delta t < \frac{\text{Min}(\Delta x, \Delta y, \Delta z)}{\sup_{i,j,k\in\mathbf{S}} |\mathcal{F}(i,j,k)|}. \quad (31)$$

As a closing remark we note that even when the speed function depends on zero or first order partial derivatives of the level set function will it typically show hyperbolic behavior[8] and should therefore be discretized using upwind-schemes and CFL time restrictions. This however, is not the case when the speed function depends on higher order partial derivatives of ϕ which is exactly the topic of the next sections.

[7] This expression can also be derived using Von Neumann stability analysis.

[8] The explicit classification of the PDE obviously depends on the actual functional dependence of the speed function on ϕ

3.2.2 Parabolic Speed Functions

Another important scenario occurs when the speed function depends on the local curvature of the level set surface. This is the case for the blending and smoothing/sharpening operators described in [Museth et al. 2002a]. In it simplest form the resulting level set equation resembles the geometric heat equation

$$\frac{\partial\phi}{\partial t} = \alpha K|\nabla\phi| \approx \alpha\nabla^2\phi, \quad (32)$$

where α is a scaling parameter and K is the mean curvature, which according to Eq. (13b) can be expressed as

$$\begin{aligned} K = \nabla\cdot\frac{\nabla\phi}{|\nabla\phi|} &= \frac{\phi_x^2(\phi_{yy}+\phi_{zz}) - 2\phi_y\phi_z\phi_{yz}}{|\nabla\phi|^3} \\ &+ \frac{\phi_y^2(\phi_{xx}+\phi_{zz}) - 2\phi_x\phi_z\phi_{xz}}{|\nabla\phi|^3} \\ &+ \frac{\phi_z^2(\phi_{xx}+\phi_{yy}) - 2\phi_x\phi_y\phi_{xy}}{|\nabla\phi|^3}, \end{aligned} \quad (33)$$

using the short-hand notation $\phi_{xy} \equiv \partial^2\phi/\partial x\partial y$. If the level set function is normalized to a signed distance function, i.e. $|\nabla\phi| = 1$, the geometric heat equation simplifies to the regular (thermodynamic) heat equation as indicated in Eq. (32). This type of PDE has a mathematical behavior which is very different from the hyperbolic PDE's behavior described in the previous section. As oppose to the latter, Eq. (32) does not propagate information in any particular direction. More specifically, the *parabolic* PDE has no real characteristics associated with it and hence the corresponding solution at a particular time and position depends (in principle) on the previous global solutions. Consequently parabolic PDEs have no domain of dependence defined from characteristics and one needs to use ordinary central finite difference schemes to discretize the spatial derivatives. So, for first order partial derivatives in Eq. (26b) we can use Eq. (28c) and we can use FD schemes of the type

$$\frac{\partial^2 u^n_{i,j,k}}{\partial x^2} = \frac{u^n_{i+1,j,k} - 2u^n_{i,j,k} + u^n_{i-1,j,k}}{\Delta x^2} + O(\Delta x^2) \quad (34a)$$

$$\frac{\partial^2 u^n_{i,j,k}}{\partial x\partial y} = \frac{u^n_{i+1,j+1,k} - u^n_{i+1,j-1,k}}{4\Delta x\Delta y} \quad (34b)$$

$$+ \frac{u^n_{i-1,j-1,k} - u^n_{i-1,j+1,k}}{4\Delta x\Delta y} + O(\Delta x^2, \Delta y^2) \quad (34c)$$

to evaluate the mean curvature in Eq. (33). Since parabolic PDEs have no mathematical domain of dependence, the CFL stability condition described in the previous section does not apply - or more correctly is not sufficient. Instead one has to perform a *Von Neumann stability analysis* [Strikwerda 1989] on the FD scheme described above. This is an error analysis in Fourier space which leads to the following stability constraint on the time steps,

$$\Delta t < \left(\frac{2\alpha}{\Delta x^2} + \frac{2\alpha}{\Delta y^2} + \frac{2\alpha}{\Delta z^2}\right)^{-1}. \quad (35)$$

It should be noted that in this stability constraint Δt is $O(\Delta w^2)$, which is significantly more stringent than in the hyperbolic case in Eq. (31) where Δt is only $O(\Delta w)$. This is a consequence of the fact that the CFL condition is a necessary but not always sufficient stability condition for a numerical FD scheme.

3.2.3 Non-linear Speed Functions

In the two previous sections we have described the numerical FD schemes used to solve two relatively simple cases when the speed

function is either independent of ϕ or linearly dependent on the mean curvature of the level set surface. However for some of the editing operators described in [Museth et al. 2002a] the situation is not so straightforward. This is for instance the case for the blending and smoothing operators where we apply a non-linear filter or cut-off function to the mean curvature flow in order to control surface properties of the final surface. Other examples include when the geometric flow depends on other curvature measures, e.g. the principle curvatures. In these cases the corresponding level set PDE cannot be classified as either hyperbolic or parabolic, which complicates the choice of discretization. As a result great care has to be taken and some experimentation is almost inevitable when implementing numerical schemes for level set equations. We shall therefore only address the case of non-linear dependence on mean curvature used in the blending and smoothing operations.

We found that the central finite differences scheme described in the previous section occasionally produced instabilities and small oscillations. As an alternative we developed a different FD scheme for mean curvature that proved more stable and also had the added benefit of easily allowing for the computation of other types of curvature.

The principle curvatures and principle directions are the eigenvalues and eigenvectors of the *shape matrix* [do Carmo 1976]. For an implicit surface, the shape matrix is the derivative of the normalized gradient (surface normals) projected onto the tangent plane of the surface. If we let the normals be $\mathbf{n} = \nabla\phi/|\nabla\phi|$, the derivative of this is the 3×3 matrix

$$\mathbf{N} = \left(\frac{\partial \boldsymbol{n}}{\partial x} \;\; \frac{\partial \mathbf{n}}{\partial y} \;\; \frac{\partial \mathbf{n}}{\partial z} \right)^T. \tag{36}$$

The projection of this derivative matrix onto the tangent plane gives the shape matrix [do Carmo 1976] $\mathbf{B} = \mathbf{N}(I - \boldsymbol{n} \otimes \boldsymbol{n})$, where $\otimes$ is the exterior product. The eigenvalues of the matrix $\mathbf{B}$ are k_1, k_2 and zero, and the eigenvectors are the principle directions and the normal, respectively. Because the third eigenvalue is zero, we can compute k_1, k_2 and various differential invariants directly from the invariants of $\mathbf{B}$. Thus the weighted curvature flow is computing from $\mathbf{B}$ using the identities $D = ||\mathbf{B}||_2$, $H = \mathrm{Tr}(\mathbf{B})/2$, and $K = 2H^2 - D^2/2$. The choice of numerical methods for computing $\mathbf{B}$ is discussed in the following section. The principle curvatures are calculated by solving the quadratic equation

$$k_{1,2} = H \pm \sqrt{\frac{D^2}{2} - H^2}. \tag{37}$$

In many circumstances, the curvature term, which is a kind of directional diffusion that does not suffer from overshooting, can be computed directly from first- and second-order derivatives of ϕ using central difference schemes. However, we have found that central differences do introduce instabilities when computing flows that rely on quantities other than the mean curvature. Therefore we use the method of *differences of normals* [Rudin et al. 1992; Whitaker and Xue 2001] in lieu of central differences. The strategy computes normalized gradients at staggered grid points and takes the difference of these staggered normals to get centrally located approximations to $\mathbf{N}$. See Figure 10. The shape matrix $\mathbf{B}$ is computed with gradient estimates based on central differences. The resulting curvatures are treated as speed functions (motion in the normal direction), and the associated gradient magnitude is computed using the up-wind scheme. For instance the normal vector centered at the green triangle in Figure 10 is approximated using the following first

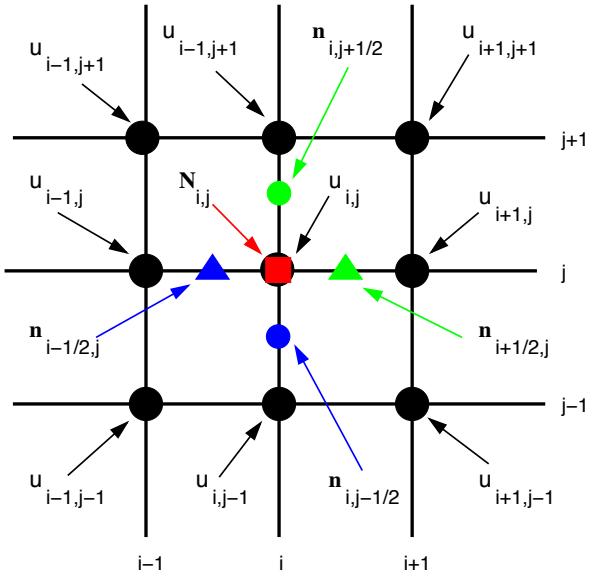


Figure 10: The normal derivative matrix $\mathbf{N}$, defined in Eq. (36), is computed by using the central finite differences of staggered (i.e. not grid-centered) normals. For instance, the normal vector centered at the green triangle is approximated using Eqs. (38a) (38b) and (38c).

order difference expressions

$$\frac{\partial u^n_{i+\frac{1}{2},j,k}}{\partial x} = \delta^+_x u^n_{i,j,k} + O(\Delta x) \tag{38a}$$

$$\frac{\partial u^n_{i+\frac{1}{2},j,k}}{\partial y} = \frac{1}{2}\left(\delta_y u^n_{i+1,j,k} + \delta_y u^n_{i,j,k}\right) + O(\Delta y) \tag{38b}$$

$$\frac{\partial u^n_{i+\frac{1}{2},j,k}}{\partial z} = \frac{1}{2}\left(\delta_z u^n_{i+1,j,k} + \delta_z u^n_{i,j,k}\right) + O(\Delta z) \tag{38c}$$

which involve the six nearest neighbors (18 in 3D).

The stability constraints on the non-linear level set PDEs can also be hard to estimate since the usual Von Neumann stability analysis typically cannot be performed. For practical purposes we found the conservative time constraints of the parabolic PDE to give good estimates for Δt.

3.2.4 Sparse-Field Solutions

The up-wind solution to the equations described in the previous section evolves the level set model over the entire range of the embedding, i.e., for all values of k in Eq. (3). However, this method requires updating *every voxel in the volume for each iteration*, making the computation time a function of the volume rather than the surface area of the model. Because surface editing only requires a single model (one level set), it is unnecessary to calculate solutions over the entire range of iso-values.

The literature has shown that computations can be limited by the use of *narrow-band* methods, which compute solutions only in the narrow band of voxels that surround the level set of interest [Adalsteinsson and Sethian 1995; Peng et al. 1999]. In previous work [Whitaker 1998] described an alternative numerical algorithm, called the sparse-field method, that evaluates the level set in a small subset of voxels in the range and requires a fraction of the computation time required by previous algorithms.[9] We have

[9]The sparse-field algorithms presented here are available in the VIS-

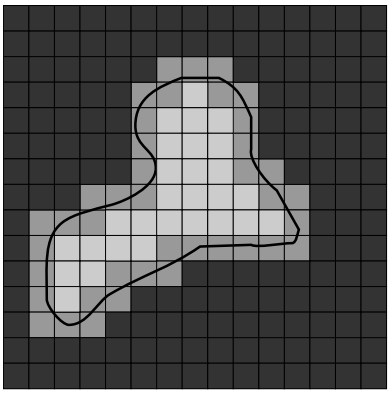


Figure 11: A level curve of a 2-D scalar field passes through a finite set of grid points. Only those grid points and their nearest neighbors are relevant to the evolution of that curve.

shown two advantages to this method. The first is a significant improvement in computation times. The second is increased accuracy when fitting models to forcing functions that are defined to subvoxel accuracy.

The sparse-field algorithm takes advantage of the fact that a k-level surface, S, of a discrete image u (of any dimension) has a set of cells through which it passes, as shown in Figure 11. The set of grid points adjacent to the level set is called the *active set*, and the individual elements of this set are called *active points*. As a first-order approximation, the distance of the level set from the center of any active point is proportional to the value of u divided by the gradient magnitude at that point. We compute the evolution given by Eq. (9b) on the active set and then update the neighborhood around the active set using a fast "city-block" approximation to the distance transform. See Section 3.1.1. Because active points must be adjacent to the level set model, their positions lie within a fixed distance to the model. Therefore the values of u for elements in the active set must lie within a certain range of greyscale values. When active point values move out of this *active range* they are no longer adjacent to the model. They must be removed from the set and other grid points. Those whose values are moving into the active range must be added to take their place. The precise ordering and execution of these operations is important to the operation of the algorithm.

The values of the points in the active set can be updated using the up-wind scheme described in the previous section. In order to maintain stability, one must update the neighborhoods of active grid points in a way that allows grid points to enter and leave the active set without those changes in status affecting their values. Grid points should be removed from the active set when they are no longer the nearest grid point to the zero crossing. If we assume that the embedding u is a discrete approximation to the distance transform of the model, then the distance of a particular grid point, (i, j, k), to the level set is given by the value of u at that grid point. If the distance between grid points is defined to be unity, then we should remove a point from the active set when the value of u at that point no longer lies in the interval $[-1/2, 1/2]$. If the neighbors of that point maintain their distance of 1, then those neighbors will move into the active range just as (i, j, k) is ready to be removed.

There are two operations that are significant to the evolution of the active set. First, the values of u at active points change from one iteration to the next. Second, as the values of active points pass out of the active range they are removed from the active set and other neighboring grid points are added to the active set to take their

PACK library at http://www.cs.utah.edu/∼whitaker/vispack.

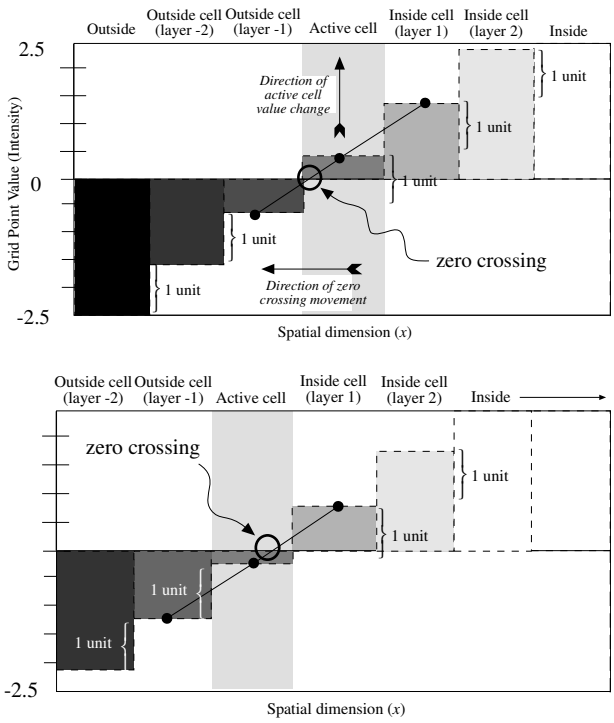


Figure 12: The status of grid points and their values at two different points in time show that as the zero crossing moves, *activity* is passed one grid point to another.

place. Formal definitions of active sets and the operations that affect them are detailed in [Whitaker 1998], and it is shown that active sets will always form a boundary between positive and negative regions in the image, even as control of the level set passes from one set of active points to another.

Because grid points that are near the active set are kept at a fixed value difference from the active points, active points serve to control the behavior of adjacent nonactive grid points. The neighborhoods of the active set are defined in *layers*, $L_{+1}, \ldots, L_{\ell}, \ldots, L_{+N}$ and $L_{-1}, \ldots, L_{-\ell}, \ldots, L_{-N}$, where the ℓ indicates the (city block) distance from the nearest active grid point, and negative numbers are used for the outside layers. For notational convenience the active set is denoted L_0. The number of layers should coincide with the size of the footprint or neighborhood used to calculate derivatives. In this way, the inside and outside grid points undergo no changes in their values that affect or distort the evolution of the zero set. Our work in general has used second-order derivatives of ϕ, which are calculated using nearest neighbors (6 connected). Therefore only 5 layers are necessary (2 inside layer, 2 outside layer, and the active set). These layers are denoted L_2, L_1, L_{-1}, L_{-2}, and L_0. The active set has grid point values in the range $[-1/2, 1/2]$. The values of the grid points in each neighborhood layer are kept 1 unit from the next layer closest to the active set as shown in Figure 12. Thus the values of layer L_{ℓ} fall in the interval $[\ell - 1/2, \ell + 1/2]$. For $2N + 1$ layers, the values of the grid points that are totally inside and outside are $N + 1/2$ and $-N - 1/2$, respectively.

This algorithm can be implemented efficiently using linked-list data structures combined with arrays to store the values of the grid points and their states as shown in Figure 13. This requires only those grid points whose values are changing, the active points and their neighbors, to be visited at each time step. The computation time grows as m^2, where m is the number of grid points along one

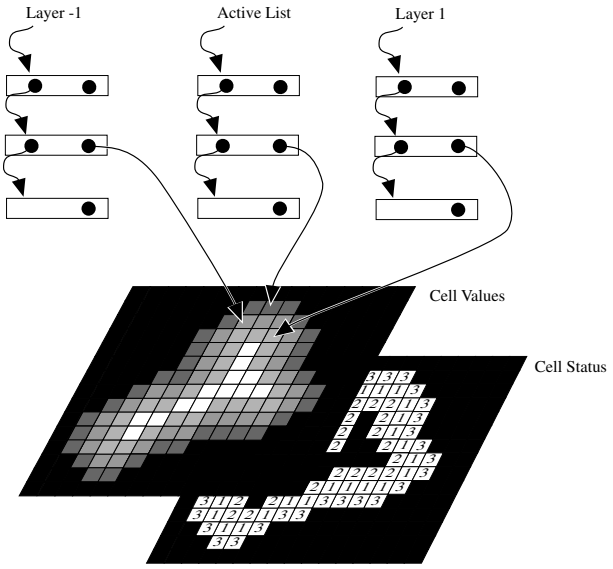


Figure 13: Linked-list data structures provide efficient access to those grid points with values and status that must be updated.

dimension of U (sometimes called the resolution of the discrete sampling). The m^2 growth in computation time for the sparse-field models is consistent with conventional (parameterized) models, for which computation times increase with surface area rather than volume.

Another advantage of the sparse-field approach is higher effective resolution. Eq. (9b) describes a process whereby all of the level sets of ϕ are pushed toward the zero-set of $\mathcal{F}()$. The result is a *shock*, a discontinuity in $|\nabla\phi|$. In discrete volumes these shocks take the form of high-contrast areas, which cause aliasing in the resulting models. This results in surface models that are unacceptable for many computer graphics applications.

When using the sparse-field method, the active points serve as a set of control points on the level set. Changing the values of these voxels changes the position of the level set. The forcing function is sampled not at the grid point, but at the location of the nearest level set, which generally lies between grid points. Using a first-order approximation to ϕ produces results that avoid the aliasing problems associated with the shocks that typically occur with level set models. Previous work has shown significant increases in the accuracy of fitting level set models using the first-order modification to the sparse-field method [Whitaker 1998].

The procedure for updating the image and the active set based on surface movements is as follows:

1. For each active grid point (i, j, k):
   (a) Use first-order derivatives and Newton's method (see 3.1) to calculate the position (i', j', k') of the nearest zero-crossing to (i, j, k).
   (b) Calculate $\mathcal{F}(i', j', k')$, and spatial derivatives at (i, j, k) using nearest neighbors according to the FD schemes discussed in Section 3.2.
   (c) Compute Δt according to Eq. (31) or Eq. (35).
   (d) Compute the net change of $u^n_{i,j,k}$, based on $\mathcal{F}(i', j', k')$ and the values of its derivatives using the up-wind scheme (Eq. (29) or 30).
2. For each active grid point (i, j, k) add the change to the grid point value and determine if the new value $u^{n+1}_{i,j,k}$ falls outside the $[-1/2, 1/2]$ interval. If so, put (i, j, k) on lists of grid points that are changing status, called the *status list*; S_1 or S_{-1}, for $u^{n+1}_{i,j,k} > \frac{1}{2}$ or $u^{n+1}_{i,j,k} < -\frac{1}{2}$, respectively.
3. Visit the grid points in the $2N$ layers L_ℓ in the order $\ell = \pm1, \ldots \pm N$, and update the grid point values based on the values (by adding or subtracting one unit) of the next inner layer, $L_{\ell\mp1}$. If more than one $L_{\ell\mp1}$ neighbor exists then use the neighbor that indicates a level set closest to that grid point, i.e., use the maximum for the outside layers and minimum for the inside layers. If a grid point in layer L_ℓ has no $L_{\ell\mp1}$ neighbors, then it is demoted to $L_{\ell\pm1}$, the next level away from the active set.
4. For each status list $S_{\pm1}, S_{\pm2}, \ldots, S_{\pm N}$:
   (a) For each element (i, j, k) on the status list S_ℓ, remove (i, j, k) from the list $L_{\ell\mp1}$, and add it to the L_ℓ list, or, in the case of $\ell = \pm(N + 1)$, remove it from all lists.
   (b) Add all $L_{\ell\mp1}$ neighbors to the $S_{\ell\pm1}$ list.

More details on the sparse-field method and its properties can be found in [Whitaker 1998].

3.2.5 Level Set Subvolumes

One of the most effective techniques for increasing interactivity in our level set editing system involves restricting computations to a subregion of the volume dataset. This is feasible because many of the editing operators by their very nature are local. The selection of the proper subvolume during the editing process is implemented with grid-aligned bounding boxes. Having the bounding boxes axis-aligned makes them straightforward to compute and manipulate, and having them grid-aligned guarantees that intersections directly correspond to valid subvolumes. The bounding box position and size are based on the geometric primitive, e.g. superellipsoid, triangle mesh or point set, utilized by a particular operator.

Employing bounding boxes within the local level set editing operators (blending, smoothing, sharpening and embossing) significantly lessens the computation time during the editing process. These operators are defined by speed functions ($\mathcal{F}()$) that specify the speed of the deformation on the surface. For the smoothing, sharpening and embossing operators, the user specifies the portion of the model to be edited by positioning a region-of-influence (ROI) primitive. The speed function is defined to be zero outside of the ROI primitive. During a blending operation a set of intersection voxels (those containing both surfaces being blended) are identified and blending only occurs within a user-specified distance of these voxels. The speed function is zero beyond this distance. In both cases no level set computation is needed in the outer regions. Given the ROI primitive and the distance information from the set of intersection voxels, a grid/axis-aligned bounding box that contains only those regions where the speed function is non-zero can be defined. A subvolume is "carved" out from the complete model by performing a CSG intersection operation with the signed distance field associated with the bounding box and the model's volume. The resulting subvolume is then passed to the level set solver, and inserted back into the model's volume after processing.

3.3 Efficient Mesh Extraction

As indicated by the green box in Figure 1 level set surfaces may either be rendered directly by means of ray casting or indirectly by a simple two-step procedure (a polygonal mesh is extracted from the volume dataset and rasterized on graphics hardware). We have successfully tested both (See Figures **??** and 17.) and found the

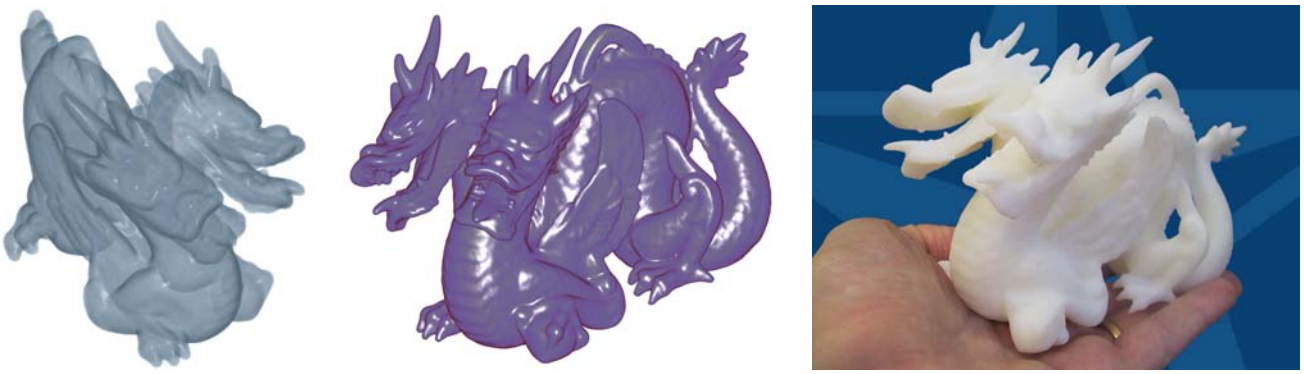


Figure 14: Volume renderings (left & center) of a winged, two-headed dragon created by merging pieces from a griffin and dragon model. A physical model (right) manufactured from the level set model.

latter to perform and scale better with the size of our volumes. Implementing a few straightforward mesh extraction procedures make the overhead of the indirect rendering approach insignificant. Conventional graphics hardware is then capable of providing interactive frame-rates for all of the models presented in this paper.

3.3.1 Fast Marching Cubes

Much work has been presented over the years on improving the quality of the triangle meshes extracted from volume datasets, the fundamental data structure of level set models [Wood et al. 2000; Gavriliu et al. 2001; Kobbelt et al. 2001]. However, these improvements come at a cost, and sacrifice speed for improved mesh structure. Fortunately, the simplicity of the original Marching Cubes (MC) algorithm [Lorensen and Cline 1987] allows us to easily optimize mesh extraction in the level set editing system.

The first optimization relies on the fact that level set models are represented by a signed distance field. This allows us to easily leap-frog through the volume as opposed to marching through the entire volume. An effective implementation of this idea is simply to increment in inner-most loop in the triple-nested for-loop of the MC algorithm by the distance of the current voxel value (i.e. $floor[|u^n_{i,j,k}|]$). While more sophisticated space-pruning schemes can certainly be designed we found this straightforward step balances the potential complexity of leap-frogging and the relatively fast hash table look-up of the MC algorithm.

Another variation of the MC algorithm that works effectively with our level set models utilizes the sparse-field representation presented in Section 3.2.4. Since the sparse-field method implements a narrow-banded distance field with a linked list of active voxels, we know at each step which voxels contain the level set of interest. The list is traversed and only those voxels needed to generate the MC mesh are processed.

3.3.2 Incremental Mesh Extraction

Even though the procedures described so far significantly improve the original MC algorithm they still do not make our indirect rendering approach truly interactive. Fortunately there are other algorithms that can be employed to achieve the goal of interactive rendering of the deforming level set surfaces. Mesh extraction can be significantly accelerated by incrementally updating the mesh only in regions where the level set surface changes.

We start by making the following observations about the bounding boxes introduced in Section 3.2.5. First, the definition of the speed functions that utilize bounding boxes guarantees that the mesh outside of the bounding boxes is unchanged after a local editing operation. Second, the bounding boxes are by definition grid-aligned and all vertices of a MC mesh lie, by construction, on grid edges. These observations lead to the following incremental mesh extraction algorithm. Given a complete global mesh we first trim away all triangles with vertices inside a bounding box. Next, for each subsequent iteration of the level set calculation, new triangles are only extracted from the sub-volume defined by the bounding box. The resulting new triangles are then incrementally added to the trimmed mesh, which by construction properly connect without the need for additional triangle clipping.

Given the collection of these procedures the mesh of the deforming level set surface may be interactively displayed while the level set equation is being iteratively solved, allowing the user to view the evolving surface and terminate processing once a desired result is achieved.

Modules \ Algorithms	Distance Calculations	Scan Conversion	Closest Point in Set	Fast Marching Methods	Bounding Boxes	Numerical Integration	Narrow Band Methods
Input Model Generation	X	X		X			X
CSG Operations							X
LS Blending			X		X	X	X
LS Smoothing/Sharpening	X				X	X	X
LS Embossing	X		X		X	X	X
Morphological Operations				X			X
Mesh Extraction					X		X

Table 1: Distribution of algorithms used in each module in our interactive level set model editing system.

4 System Modules

Table 1 identifies the specific algorithms utilized in each of the modules in our interactive level set model editing system. Since a wide variety of geometric models may be imported into our system, many algorithms are needed to perform the necessary conversions, including shortest distance calculations (Sections 3.1.3, 3.1.4), scan

conversion (Section 3.1.2) and the Fast Marching Method (Section 3.1.6). All of the level set deformation operators (blending, smoothing, sharpening and embossing) use bounding boxes (Section 3.2.5), numerical integration (Section 3.2) and the sparse-field techniques (Section 3.2.4). The blending and embossing operators use K-D trees (Section 3.1.5) to quickly find closest points. The smoothing, sharpening and embossing operators utilize shortest distance calculations (Section 3.1.3) for localizing computation. The morphological operators employ the Fast Marching Method (Section 3.1.6) to calculate the needed distance information. Our mesh extraction algorithm also extensively utilizes bounding boxes and the active list of the level set solver to implement an incremental version of the Marching Cubes algorithm [Lorensen and Cline 1987]. All of the modules use some kind of narrow band calculation to either limit computation to only those voxels near the level set of interest (Section 3.2.4), or to re-establish proper distance information in the narrow band after performing its operation (Section 3.1.1).

5 Results

We have produced numerous models with our level set editing system. The teapot (NURBS surface), dragon (scanned volume), human head and bust (polygonal surfaces) and eyelet on the winged dragon's back (superquadric) in Figures 9, 14, 15, 17 demonstrate that we are able to import several types of models into our system. The CSG operators with blending were utilized to produce the winged, double-headed dragon and repaired bust in Figures 14 and 17. The images of the dragon are volume rendered and were interactively produced by VTK's Volview program utilizing TeraRecon's VolumePro 1000 volume rendering hardware. The smoothing operator is used to fix problems in a model produced by an early, unfinished version of the NURBS scan conversion code in Figure 16. The embossing operator produced the result in Figure 9. The results of our morphological operators [Serra 1982] are presented in Figure 15. It should be noted that the images in Figure 17 are screen shots from an interactive editing session with our system, running on a Linux PC with an AMD Athlon 1.7GHz processors. All of the following timing information is produced on this computer.

A level set editing session, as illustrated in Figure 17, begins by first importing a level set model into our system. The process of generating an initial level set model, e.g. with scan conversion, is not incorporated into the system. It is considered a separate preprocessing step. Once a model[10] is brought into the system, it and the tools to modify it may be interactively (at $\sim$30Hz) manipulated and viewed. Once a level set editing operation (e.g. blending, smoothing, embossing, and opening) is invoked, an iterative computational process modifies the model. After each iteration the current state of the model is displayed, allowing the user to stop the operation, once a desired result is produced. We have found that most operations need approximately 10 iterations to produce a satisfactory result. Each iteration takes approximately 1/2 to 1 second, this includes level set evolution, mesh extraction and display. Therefore most level set operations take 5 to 10 seconds to complete. The CSG operations are not iterative and require less than one second of computation time. These computation times provide an environment that allows a user to quickly specify an operation, and then wait just a few seconds for it to complete. Our system includes an undo facility, giving the user the ability to rapidly try numerous editing operations until the best result is found.

[10]The models in this paper are represented by volume datasets with a resolution of approximately 256^3.

6 Conclusions

This paper has described the collection of techniques and algorithms (some new, some pre-existing) needed to create an interactive editing system for level set models. It has summarized the algorithms for producing level set input models and, more importantly, for localizing/minimizing computation during the editing process. These algorithms include distance calculations, scan conversion, closest point determination, fast marching methods, bounding box creation, incremental and fast mesh extraction, numerical integration, and narrow band techniques. Together these algorithms provide the capabilities required for the interactive editing of level set models.

7 Acknowledgments

We would like to thank Alan Barr and the other members of the Caltech Computer Graphics Group for their assistance and support. Additional thanks to Katrine Museth and Santiago Lombeyda for assistance with the figures and Jason Wood for developing useful visualization tools. The teapot model and the manufactured dragon figurine were provided by the University of Utah's Geometric Design and Computation Group. The Greek bust and human head models were provided by Cyberware Inc. The dragon and griffin models were provided by the Stanford Computer Graphics Laboratory and Caltech's Multires Modeling Group. This work was financially supported by National Science Foundation grants ASC-8920219, ACI-9982273, ACI-0083287 and ACI-0089915, and subcontract B341492 under DOE contract W-7405-ENG-48.

References

ADALSTEINSSON, D., AND SETHIAN, J. 1995. A fast level set method for propagating interfaces. *Journal of Computational Physics*, 269–277.

ARATA, H., TAKAI, Y., TAKAI, N., AND YAMAMOTO, T. 1999. Free-form shape modeling by 3D cellular automata. In *Proc. Shape Modeling International Conference*, 242–247.

ARYA, S., AND MOUNT, D. 1993. Algorithms for fast vector quantization. In *Proc. IEEE Data Compression Conference*, 381–390.

ARYA, S., MOUNT, D., NETANYAHU, N., SILVERMAN, R., AND WU, A. 1998. An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM 45*, 891–923.

AVILA, R., AND SOBIERAJSKI, L. 1996. A haptic interaction method for volume visualization. In *Proc. IEEE Visualization Conference*, 197–204.

BAERENTZEN, J., AND CHRISTENSEN, N. 2002. Volume sculpting using the level-set method. In *Proc. Shape Modeling International Conference*, 175–182.

BARR, A. 1981. Superquadrics and angle-preserving transformations. *IEEE Computer Graphics and Applications 1*, 1, 11–23.

BLOOMENTHAL, J., ET AL., Eds. 1997. *Introduction to Implicit Surfaces*. Morgan Kaufmann.

BOLZ, J., FARMER, I., GRINSPUN, E., AND SCHRÖDER, P. 2003. Sparse matrix solvers on the GPU: Conjugate gradients and multigrid. *ACM Trans. on Graphics (Proc. SIGGRAPH) 22*, 3 (July), 917–924.

BREEN, D., AND WHITAKER, R. 2001. A level set approach for the metamorphosis of solid models. *IEEE Trans. on Visualization and Computer Graphics 7*, 2, 173–192.

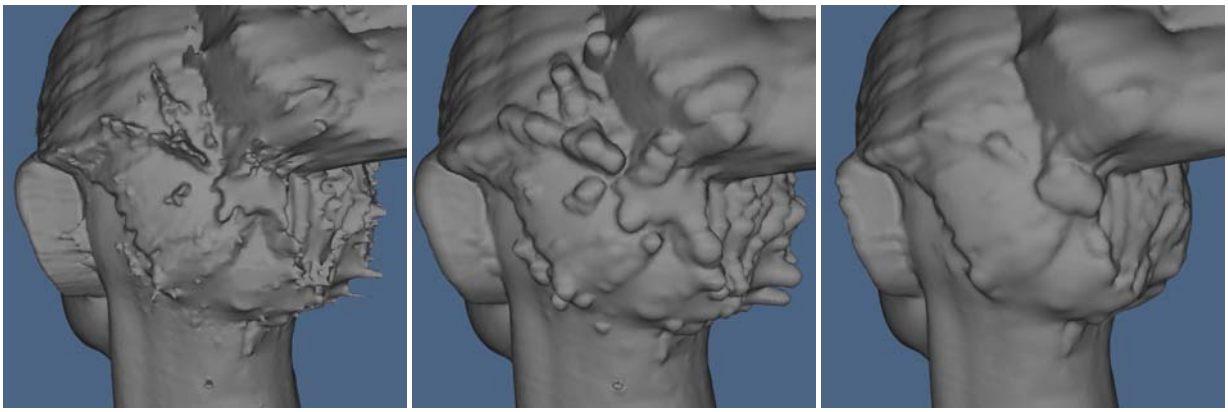


Figure 15: Applying a morphological opening to a laser scan reconstruction of a human head (left). The opening performs global smoothing by removing protruding structures smaller than a user-defined value d. First an offset surface a distance d outwards (dilation) is created (center), followed by an offset surface distance d inwards (erosion) to produce the smoothed result (right).

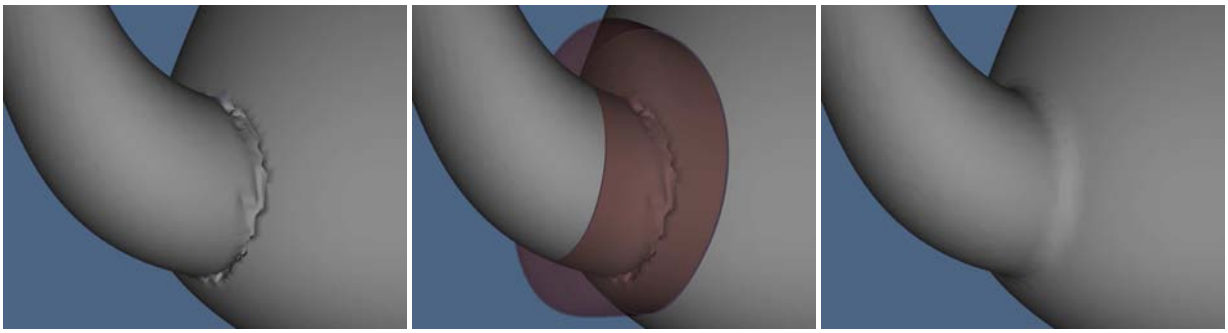


Figure 16: Scan conversion errors near the teapot spout (left). These errors were produced by an early pre-debugged version of our software. Placing a (red) superellipsoid around the errors (middle). The errors are smoothed away with a level set smoothing operator (right).

BREEN, D., MAUCH, S., AND WHITAKER, R. 2000. 3D scan conversion of CSG models into distance, closest-point and colour volumes. In *Volume Graphics*, M. Chen, A. Kaufman, and R. Yagel, Eds. Springer, London, 135–158.

BURDEN, R., AND FAIRES, J. 2001. *Numerical Analysis, Seventh Edition.* Brooks/Cole Publishing, Pacific Grove, CA.

CAMERON, S. 1997. Enhancing GJK: Computing minimum penetration distances between convex polyhedra. In *Proc. Int. Conf. On Robotics and Automation*, 3112–3117.

CASALE, M., AND STANTON, E. 1985. An overview of analytic solid modeling. *IEEE Computer Graphics and Applications 5*, 2 (February), 45–56.

CORMEN, T., LEISERSON, C., RIVEST, R., AND STEIN, C. 2001. *Introduction to Algorithms, Second Edition.* MIT Press, Cambridge, MA.

COURANT, R., FRIEDRICHS, K. O., AND LEWY, H. 1928. Uber die partiellen differenzengleichungen der mathematisches physik. *Math. Ann. 100*, 32–74.

CUTLER, B., DORSEY, J., MCMILLAN, L., MÜLLER, M., AND JAGNOW, R. 2002. A procedural approach to authoring solid models. *ACM Trans. on Graphics (Proc. SIGGRAPH) 21*, 3 (July), 302–311.

DE BERG, M., VAN KREVELD, M., OVERMARS, M., AND SCHWARZKOPF, O. 1997. *Computational Geometry: Algorithms and Applications*. Springer, Berlin.

DESBRUN, M., AND CANI, M.-P. 1998. Active implicit surface for animation. In *Proc. Graphics Interface*, 143–150.

DIJKSTRA, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik 1*, 333–352.

DO CARMO, M. 1976. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, Englewood Cliffs, NJ.

ENRIGHT, D., MARSCHNER, S., AND FEDKIW, R. 2002. Animation and rendering of complex water surfaces. *ACM Trans. on Graphics (Proc. SIGGRAPH) 21*, 3 (July), 736–744.

FANG, S., AND CHEN, H. 2000. Hardware accelerated voxelization. *Computers & Graphics 24*, 3 (June), 433–442.

FERLEY, E., CANI, M.-P., AND GASCUEL, J.-D. 2000. Practical volumetric sculpting. *The Visual Computer 16*, 8, 211–221.

FERLEY, E., CANI, M.-P., AND GASCUEL, J.-D. 2001. Resolution adaptive volume sculpting. *The Visual Computer 63*, 6 (November), 459–478.

FRISKEN, S., PERRY, R., ROCKWOOD, A., AND JONES, T. 2000. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *SIGGRAPH 2000 Proceedings*, 249–254.

GALYEAN, T., AND HUGHES, J. 1991. Sculpting: An interactive volumetric modeling technique. In *Proc. SIGGRAPH '91*, 267–274.

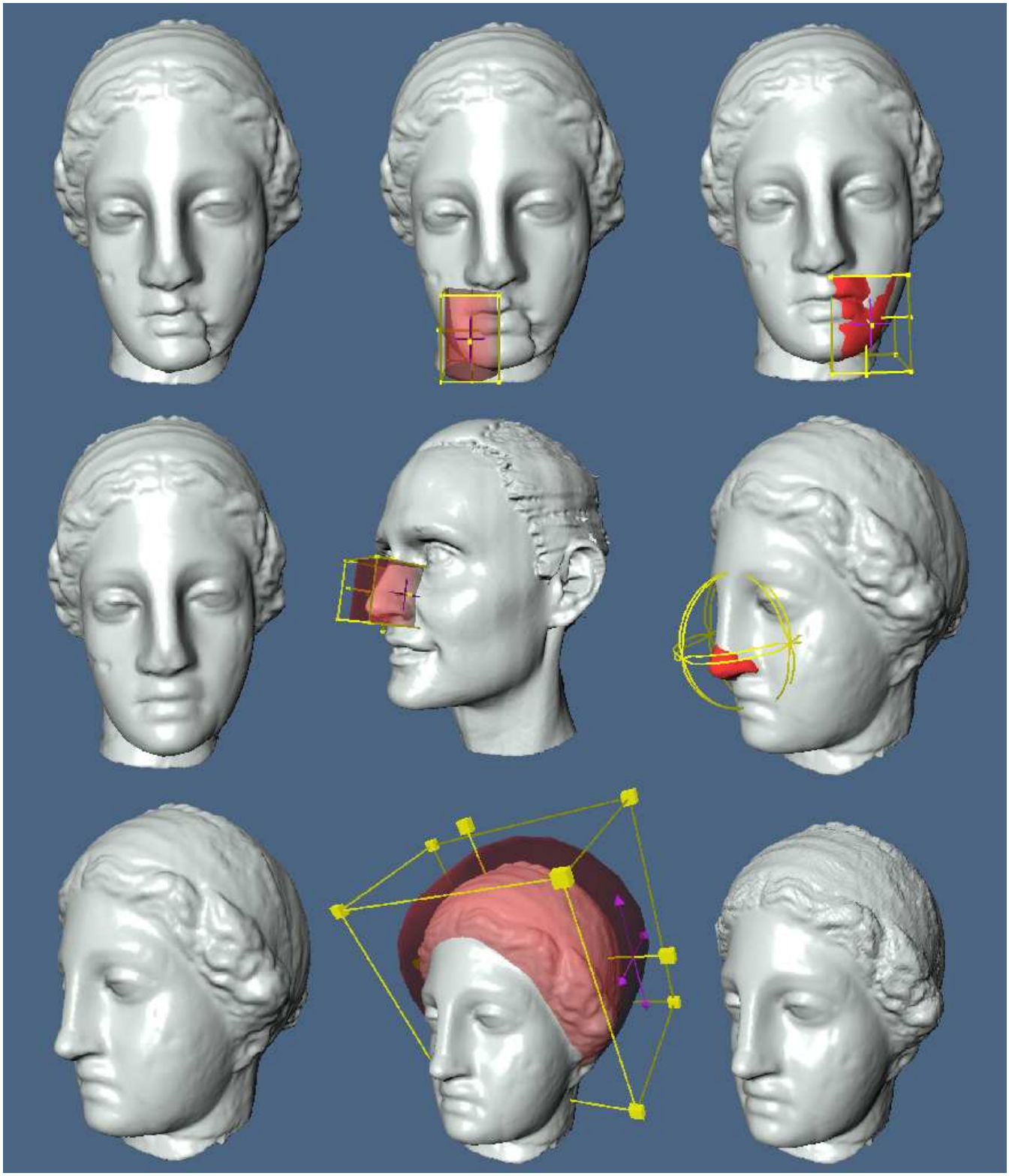


Figure 17: Repairing a Greek bust. The right cheek is first copied, mirrored, pasted, and blended back onto the left side of the bust. Next a nose is copied from a human head model, scaled and blended onto the broken nose of the Greek bust. Finally the hair of the bust is chiseled by a localized sharpening operation.

GAVRILIU, M., CARRANZA, J., BREEN, D., AND BARR, A. 2001. Fast extraction of adaptive multiresolution meshes with guaranteed properties from volumetric data. In *Proceedings of IEEE Visualization 2001*, 295–302.

GILBERT, E., JOHNSON, D., AND KEERTHI, S. 1988. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Trans. on Robotics and Automation 4*, 2 (April), 193–203.

HOFFMANN, C. 1989. *Geometric and Solid Modeling*. Morgan Kaufmann.

HOPF, M., AND ERTL, T. 2000. Accelerating morphological analysis with graphics hardware. In *Proceedings of Workshop on Vision, Modeling and Visualization*, 337–345.

KOBBELT, L. P., BOTSCH, M., SCHWANECKE, U., AND SEIDEL, H.-P. 2001. Feature sensitive surface extraction from volume data. In *Proc. SIGGRAPH 2001*, 57–66.

LEFOHN, A., KNISS, J., HANSEN, C., AND WHITAKER, R. 2003. Interactive deformation and visualization of level set surfaces using graphics hardware. In *Proceedings of IEEE Visualization 2003*, 75–82.

LORENSEN, W., AND CLINE, H. 1987. Marching Cubes: A high resolution 3D surface construction algorithm. In *Proc. SIGGRAPH '87*, 163–169.

MACCRACKEN, R., AND ROY, K. 1996. Free-form deformations with lattices of arbitrary topology. In *Proc. SIGGRAPH '96*, 181–188.

MALLADI, R., SETHIAN, J., AND VEMURI, B. 1995. Shape modeling with front propagation. *IEEE Trans. PAMI 17*, 2, 158–175.

MARAGOS, P. 1996. Differential morphology and image processing. *IEEE Trans. on Image Processing 5*, 6 (June), 922–937.

MAUCH, S. 2003. *Effecient Algorithms for Solving Static Hamilton-Jacobi Equations*. PhD thesis, California Institute of Technology, Pasadena, California.

MAUCH, S. 2004. A fast algorithm for computing the closest point and distance transform. to appear in the SIAM Journal of Scientific Computing.

MCDONNELL, K., QIN, H., AND WLODARCZYK, R. 2001. Virtual clay: A real-time sculpting system with haptic toolkits. In *Proc. Symposium on Interactive 3D Graphics*, ACM SIGGRAPH, 179–190.

MEAGHER, P. 1982. Geometric modeling using octree encoding. *Computer Graphics and Image Processing 19*, 2 (June), 129–147.

MUSETH, K., BREEN, D., WHITAKER, R., AND BARR, A. 2002. Level set surface editing operators. *ACM Trans. on Graphics (Proc. SIGGRAPH) 21*, 3 (July), 330–338.

MUSETH, K., BREEN, D., ZHUKOV, L., AND WHITAKER, R. 2002. Level set segmentation from multiple non-uniform volume datasets. In *Proc. IEEE Visualization Conference*, 179–186.

OSHER, S., AND FEDKIW, R. 2002. *Level Set Methods and Dynamic Implicit Surfaces*. Springer, Berlin.

OSHER, S., AND SETHIAN, J. 1988. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics 79*, 12–49.

PENG, D., MERRIMAN, B., OSHER, S., ZHAO, H.-K., AND KANG, M. 1999. A PDE-based fast local level set method. *Journal of Computational Physics 155*, 410–438.

PERRY, R., AND FRISKEN, S. 2001. Kizamu: A system for sculpting digital characters. In *Proc. SIGGRAPH 2001*, 47–56.

PIEGL, L., AND TILLER, W. 1998. Geometry-based triangulation of trimmed NURBS surfaces. *Computer-Aided Design 30*, 11–18.

RAVIV, A., AND ELBER, G. 2000. Three-dimensional freeform sculpting via zero sets of scalar trivariate functions. *Computer-Aided Design 32* (August), 513–526.

ROUY, E., AND TOURIN, A. 1992. A viscosity solutions approach to shape-from-shading. *SIAM J. Num. Anal. 29*, 867–884.

RUDIN, L., OSHER, S., AND FATEMI, C. 1992. Nonlinear total variation based noise removal algorithms. *Physica D 60*, 259–268.

RUMPF, M., AND STRZODKA, R. 2001. Level set segmentation in graphics hardware. In *Proceedings of ICIP '01*, vol. 3, 1103–1106.

RUMPF, M., AND STRZODKA, R. 2001. Nonlinear diffusion in graphics hardware. In *Proceedings of EG/IEEE TCVG Symposium on Visualization*, 75–84.

SAPIRO, G., AND TANNENBAUM, A. 1993. Affine invariant scale space. *International Journal of Computer Vision 11*, 25–44.

SAPIRO, G. 2001. *Geometric Partial Differential Equations and Image Analysis*. Cambridge University Press, Cambridge, UK.

SERRA, J. 1982. *Image Analysis and Mathematical Morphology*. Academic Press.

SETHIAN, J. 1996. A fast marching level set method for monotonically advancing fronts. In *Proceedings of the National Academy of Science*, vol. 93 of 4, 1591–1595.

SETHIAN, J. 1999. *Level Set Methods and Fast Marching Methods*, second ed. Cambridge University Press, Cambridge, UK.

SHANTZ, M., AND CHANG, S.-L. 1988. Rendering trimmed NURBS with adaptive forward differencing. In *Proc. SIGGRAPH '88*, 189–198.

SHERBONDY, A., HOUSTON, M., AND NAPEL, S. 2003. Fast volume segmentation with simultaneous visualization using programmable graphics hardware. In *Proceedings of IEEE Visualization 2003*, 171–176.

SIGG, C., PEIKERT, R., AND GROSS, M. 2003. Signed distance transform using graphics hardware. In *Proceedings of IEEE Visualization 2003*, 83–90.

STRIKWERDA, J. 1989. *Finite Difference Schemes and Partial Differential Equations*. Wadsworth Publishing Co., Belmont, CA.

TSITSIKLIS, J. 1995. Efficient algorithms for globally optimal trajectories. *IEEE Trans. on Automatic Control 40*, 9, 1528–1538.

WANG, S., AND KAUFMAN, A. 1994. Volume-sampled 3D modeling. *IEEE Computer Graphics and Applications 14*, 5 (September), 26–32.

WANG, S., AND KAUFMAN, A. 1995. Volume sculpting. In *1995 Symposium on Interactive 3D Graphics*, ACM SIGGRAPH, 151–156.

WHITAKER, R., AND BREEN, D. 1998. Level-set models for the deformation of solid objects. In *The Third International Workshop on Implicit Surfaces*, Eurographics, 19–35.

WHITAKER, R., AND XUE, X. 2001. Variable-conductance, level-set curvature for image denoising. In *Proc. IEEE International Conference on Image Processing*, 142–145.

WHITAKER, R., BREEN, D., MUSETH, K., AND SONI, N. 2001. Segmentation of biological datasets using a level-set framework. In *Volume Graphics 2001*, M. Chen and A. Kaufman, Eds. Springer, Vienna, 249–263.

WHITAKER, R. 1998. A level-set approach to 3D reconstruction from range data. *International Journal of Computer Vision 29*, 3, 203–231.

WOOD, Z., M.DESBRUN, SCHRÖDER, P., AND BREEN, D. 2000. Semi-regular mesh extraction from volumes. In *Proceedings of IEEE Visualization 2000*, 275–282.

WYVILL, B., GALIN, E., AND GUY, A. 1999. Extending the CSG tree. warping, blending and boolean operations in an implicit surface modeling system. *Computer Graphics Forum 18*, 2 (June), 149–158.

YANG, R., AND WELCH, G. 2002. Fast image segmentation and smoothing using commodity graphics hardware. *Journal of Graphics Tools 7*, 4, 91–100.

ZHAO, H.-K., CHEN, T., MERRIMAN, B., AND OSHER, S. 1996. A variational level set approach to multiphase motion. *J. Comput. Phys. 127*, 179–195.

ZHAO, H.-K., OSHER, S., AND FEDKIW, R. 2001. Fast surface reconstruction using the level set method. In *Proc. 1st IEEE Workshop on Variational and Level Set Methods*, 194–202.

Session 4

Level Set Segmentation and Simulation

Level Set Segmentation of Biological Volume Datasets

David Breen [1] *Ross Whitaker* [2]
Ken Museth [3] *Leonid Zhukov* [4]

[1]Department of Computer Science, Drexel University, Philadelphia, PA 19104
[2]School of Computing, University of Utah, Salt Lake City, UT 84112
[3]Department of Science and Technology, Linkoeping University, 601 74 Norrkoeping, Sweden
[4]Department of Computer Science, California Institute of Technology, Pasadena, CA 91125

Abstract

This chapter describes level set techniques for extracting surface models from a broad variety of biological volume datasets. These techniques have been incorporated into a more general framework that includes other volume processing algorithms. The volume datasets are produced from standard 3D imaging devices, and are all noisy samplings of complex biological structures with boundaries that have low and often varying contrasts. The level set segmentation method, which is well documented in the literature, creates a new volume from the input data by solving an initial value partial differential equation (PDE) with user-defined feature-extracting terms. Given the local/global nature of these terms, proper initialization of the level set algorithm is extremely important. Thus, level set deformations alone are not sufficient, they must be combined with powerful pre-processing and data analysis techniques in order to produce successful segmentations. This chapter describes the pre-processing and data analysis techniques that have been developed for a number of segmentation applications, as well as the general structure of our framework. Several standard volume processing algorithms have been incorporated into the framework in order to segment datasets generated from MRI, CT and TEM scans. A technique based on moving least-squares has been developed for segmenting multiple non-uniform scans of a single object. New scalar measures have been defined for extracting structures from diffusion tensor MRI scans. Finally, a direct approach to the segmentation of incomplete tomographic data using density parameter estimation is described. These techniques, combined with level set surface deformations, allow us to segment many different types of biological volume datasets.

0.1 Introduction

This chapter addresses the common problem of building meaningful 3D models of complex structures from noisy datasets generated from 3D imaging devices. In certain circumstances such data can be visualized *directly* [1, 2, 3, 4]. While direct techniques can provide useful insights into volume data, they are insufficient for many problems. For instance, direct volume rendering techniques typically do not remove occluding structures, i.e., they do not allow one to "peel back" the various layers of the data to expose the inner structures that might

be of interest. They also do not generate the models needed for quantitative study/analysis of the visualized structures. Furthermore, direct visualization techniques typically do not perform well when applied directly to noisy data, unless one filters the data first. Techniques for filtering noisy data are abundant in the literature, but there is a fundamental limitation—filtering that reduces noise tends to distort the shapes of the objects in the data. The challenge is to find methods which present the best tradeoff between fidelity and noise.

Level set segmentation relies on a surface-fitting strategy, which is effective for dealing with both small-scale noise and smoother intensity fluctuations in volume data. The level set segmentation method, which is well documented in the literature [5, 6, 7, 8], creates a new volume from the input data by solving an initial value partial differential equation (PDE) with user-defined feature-extracting terms. Given the local/global nature of these terms, proper initialization of the level set algorithm is extremely important. Thus, level set deformations alone are not sufficient, they must be combined with powerful initialization techniques in order to produce successful segmentations. Our level set segmentation approach consists of defining a set of suitable pre-processing techniques for initialization and selecting/tuning different feature-extracting terms in the level set algorithm. We demonstrate that combining several pre-processing steps, data analysis and level set deformations produces a powerful toolkit that can be applied, under the guidance of a user, to segment a wide variety of volumetric data.

There are more sophisticated strategies for isolating meaningful 3D structures in volume data. Indeed, the so called *segmentation problem* constitutes a significant fraction of the literature in image processing, computer vision, and medical image analysis. For instance, statistical approaches [9, 10, 11, 12] typically attempt to identify tissue types, voxel by voxel, using a collection of measurements at each voxel. Such strategies are best suited to problems where the data is inherently multi-valued or where there is sufficient prior knowledge [13] about the shape or intensity characteristics of the relevant anatomy. Alternatively, anatomical structures can be isolated by grouping voxels based on local image properties. Traditionally, image processing has relied on collections of edges, i.e. high-contrast boundaries, to distinguish regions of different types [14, 15, 16]. Furthermore deformable models, incorporating different degrees of domain-specific knowledge, can be *fitted* to the 3D input data [17, 18].

This chapters describes a level set segmentation framework, as well as the the pre-processing and data analysis techniques needed to segment a diverse set biological volume datasets. Several standard volume processing algorithms have been incorporated into framework for segmenting conventional datasets generated from MRI, CT and TEM scans. A technique based on moving least-squares has been developed for segmenting multiple non-uniform scans of a single object. New scalar measures have been defined for extracting structures from diffusion tensor MRI scans. Finally, a direct approach to the segmentation of incomplete tomographic data using density parameter estimation is described. These techniques, combined with level set surface deformations, allow us to segment many different types of biological volume datasets.

0.2 Level Set Surface Models

When considering deformable models for segmenting 3D volume data, one is faced with a choice from a variety of surface representations, including triangle meshes [19, 20], superquadrics [21, 22, 23], and many others [18, 24, 25, 26, 27, 28, 29]. Another option is an implicit level set model, which specifies the surface as a *level set* of a scalar volumetric function, $\phi : U \mapsto \mathbb{R}$, where $U \subset \mathbb{R}^3$ is the range of the surface model. Thus, a surface S is

$$S = \{\boldsymbol{s} | \phi(\boldsymbol{s}) = k\}, \tag{1}$$

with an isovalue k. In other words, S is the set of points $\boldsymbol{s}$ in $\mathbb{R}^3$ that composes the kth isosurface of ϕ. The embedding ϕ can be specified as a regular sampling on a rectilinear grid.

Our overall scheme for segmentation is largely based on the ideas of Osher and Sethian [30] that model propagating surfaces with (time-varying) curvature-dependent speeds. The surfaces are viewed as a specific level set of a higher-dimensional function ϕ – hence the name level set methods. These methods provide the mathematical and numerical mechanisms for computing surface deformations as isovalues of ϕ by solving a partial differential equation on the 3D grid. That is, the level set formulation provides a set of numerical methods that describes how to manipulate the greyscale values in a volume, so that the isosurfaces of ϕ move in a prescribed manner (shown in Figure 1). This paper does not present a comprehensive review of level set methods, but merely

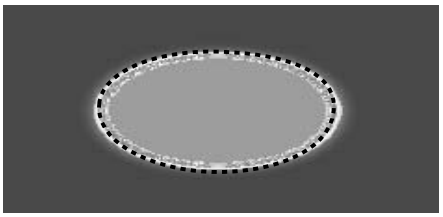


Figure 1a: Level set models represent curves and surfaces implicitly using greyscale images. For example an ellipse is represented as the level set of an image shown here.

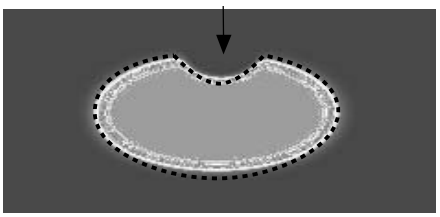


Figure 1b: To change the shape of the ellipse we modify the greyscale values of the image by solving a PDE.

introduces the basic concepts and demonstrates how they may be applied to the problem of volume segmentation. For more details on level set methods see [7, 31].

There are two different approaches to defining a deformable surface from a level set of a volumetric function as described in Equation 1. Either one can think of $\phi(\boldsymbol{s})$ as a *static* function and change the isovalue $k(t)$ or alternatively fix k and let the volumetric function *dynamically* change in time, i.e. $\phi(\boldsymbol{s}, t)$. Thus, we can mathematically express the static and dynamic model respectively as

$$\phi(\boldsymbol{s}) = k(t) \tag{2a}$$

$$\phi(\boldsymbol{s}, t) = k. \tag{2b}$$

To transform these definitions into partial differential equations which can be solved by standard numerical techniques, we differentiate both sides of Equation 2 with respect to time t, and apply the chain rule:

$$\nabla\phi(\boldsymbol{s})\frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}t} = \frac{\mathrm{d}k(t)}{\mathrm{d}t} \tag{3a}$$

$$\frac{\partial\phi(\boldsymbol{s}, t)}{\partial t} + \nabla\phi(\boldsymbol{s}, t) \cdot \frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}t} = 0. \tag{3b}$$

The static Equation 3a defines a boundary value problem for the time-independent volumetric function ϕ. This static level set approach has been solved [32, 33] using "Fast Marching Methods". However it inherently has some serious limitations following the simple definition in Equation 2a. Since ϕ is a function (i.e. single-valued), isosurfaces cannot self intersect over time, i.e. shapes defined in the static model are strictly expanding or contracting over time. However,

the dynamic level set approach of Equation 3b is much more flexible and shall serve as the basis of the segmentation scheme in this paper. Equation 3b is sometimes referred to as a "Hamilton-Jacobi-type" equation and defines an initial value problem for the time-dependent ϕ. Throughout the remainder of this paper we shall for simplicity refer to this dynamical approach as the level set method – and not consider the static alternative.

Thus, to summarize the essence of the (dynamic) level set approach; let $\mathrm{d}\boldsymbol{s}/\mathrm{d}t$ be the movement of a point on a surface as it deforms, such that it can be expressed in terms of the position of $\boldsymbol{s} \in U$ and the geometry of the surface at that point, which is, in turn, a differential expression of the implicit function, ϕ. This gives a partial differential equation on ϕ: $\boldsymbol{s} \equiv \boldsymbol{s}(t)$

$$\frac{\partial \phi}{\partial t} = -\nabla\phi \cdot \frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}t} = \|\nabla\phi\| \, \mathcal{F}(\boldsymbol{s}, \boldsymbol{n}, \phi, \mathrm{D}\phi, \mathrm{D}^2\phi, \ldots) \tag{4a}$$

$$\mathcal{F}() \equiv \boldsymbol{n} \cdot \frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}t}, \tag{4b}$$

where $\mathcal{F}()$ is a user-created "speed" term that defines the speed of the level set at point $\boldsymbol{s}$ in the direction of the local surface normal $\boldsymbol{n}$ at $\boldsymbol{s}$. $\mathcal{F}()$ may depend on a variety of local and global measures including the order-n derivatives of ϕ, $D^n\phi$, evaluated at $\boldsymbol{s}$, as well as other functions of $\boldsymbol{s}$, $\boldsymbol{n}$, ϕ and external data. Because this relationship applies to every level set of ϕ, i.e. all values of k, this equation can be applied to all of U, and therefore the movements of *all* the level set surfaces embedded in ϕ can be calculated from Equation 4.

The level set representation has a number of practical and theoretical advantages over conventional surface models, especially in the context of deformation and segmentation. First, level set models are topologically flexible, they easily represent complicated surface shapes that can, form holes, split to form multiple objects, or merge with other objects to form a single structure. These models can incorporate many (millions) of degrees of freedom, and therefore they can accommodate complex shapes such as the dendrite in Figure 6. Indeed, the shapes formed by the level sets of ϕ are restricted only by the resolution of the sampling. Thus, there is no need to reparameterize the model as it undergoes significant changes in shape.

The solutions to the partial differential equations described above are computed using finite differences on a discrete grid. The use of a grid and discrete time steps raises a number of numerical and computational issues that are important to the implementation. However, it is outside of the scope of this paper

to give a detailed mathematical description of such a numerical implementation. Rather we shall provide a summary in a later section and refer to the actual source code which is publicly available[1].

Equation 4 can be solved using finite forward differences if one uses the up-wind scheme, proposed by Osher and Sethian [30], to compute the spatial derivatives. This up-wind scheme produces the motion of level set models over the entire range of the embedding, i.e., for all values of k in Equation 2. However, this method requires updating *every voxel in the volume for each iteration.*, which means that the computation time increases as a function of the volume, rather than the surface area, of the model. Because segmentation requires only a single model, the calculation of solutions over the entire range of iso-values is an unnecessary computational burden.

This problem can be avoided by the use of *narrow-band* methods, which compute solutions only in a narrow band of voxels that surround the level set of interest [34, 35]. In previous work [36] we described an alternative numerical algorithm, called the sparse-field method, that computes the geometry of only a small subset of points in the range and requires a fraction of the computation time required by previous algorithms. We have shown two advantages to this method. The first is a significant improvement in computation times. The second is increased accuracy when fitting models to forcing functions that are defined to sub-voxel accuracy.

0.3 Segmentation Framework

The level set segmentation process has two major stages, initialization and level set surface deformation, as seen in Figure 2. Each stage is equally important for generating a correct segmentation. Within our framework a variety of core operations are available in each stage. A user must "mix-and-match" these operations in order to produce the desired result [37]. Later sections describe specialized operations for solving specific segmentation problems that build upon and extend the framework.

[1]The level set software used to produce the morphing results in this paper is available for public use in the VISPACK libraries at http://www.cs.utah.edu/∼whitaker/vispack.

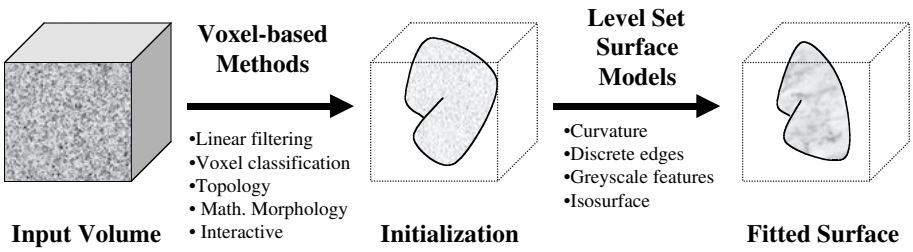


Figure 2: Level set segmentation stages – initialization and surface deformation.

0.3.1 Initialization

Because level set models move using gradient descent, they seek *local solutions*, and therefore the results are strongly dependent on the initialization, i.e., the starting position of the surface. Thus, one controls the nature of the solution by specifying an initial model from which the surface deformation process proceeds. We have implemented both computational (i.e. "semi-automated") and manual/interactive initialization schemes that that may be combined to produce reasonable initial estimates directly from the input data.

Linear filtering: We can filter the input data with a low-pass filter (e.g. Gaussian kernel) to blur the data and thereby reduce noise. This tends to distort shapes, but the initialization need only be approximate.

Voxel classification: We can classify pixels based on the filtered values of the input data. For greyscale images, such as those used in this paper, the classification is equivalent to high and low thresholding operations. These operations are usually accurate to only voxel resolution (see [12] for alternatives), but the deformation process will achieve sub-voxel results.

Topological/logical operations: This is the set of basic voxel operations that takes into account position and connectivity. It includes unions or intersections of voxel sets to create better initializations. These logical operations can also incorporate user-defined primitives. Topological operations consist of connected-component analyses (e.g. flood fill) to remove small pieces or holes from objects.

Morphological filtering: This includes binary and greyscale morphological operators on the initial voxel set. For the results in the paper we

implement openings and closings using *morphological propagators* [38, 39] implemented with level set surface models. This involves defining offset surfaces of ϕ by expanding/contracting a surface according to the following PDE,

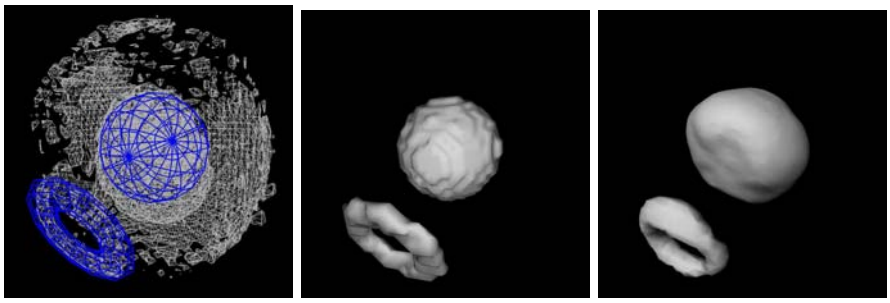
$$\frac{\partial \phi}{\partial t} = \pm |\nabla \phi|, \tag{5}$$

up to a certain time t. The value of t controls the offset distance from the original surface of $\phi(t = 0)$. A dilation of size α, D_α, corresponds to the solution of Equation 5 at $t = \alpha$ using the positive sign, and likewise erosion, E_α, uses the negative sign. One can now define a morphological opening operator O_α by first applying an erosion followed by a dilation of ϕ, i.e. $\mathrm{O}_\alpha \phi = \mathrm{D}_\alpha \circ \mathrm{E}_\alpha \phi$, which removes small pieces or thin appendages. A closing is defined as $\mathrm{C}_\alpha \phi = \mathrm{E}_\alpha \circ \mathrm{D}_\alpha \phi$, and closes small gaps or holes within objects. Both operations have the qualitative effect of low-pass filtering the isosurfaces in ϕ—an opening by removing material and a closing by adding material. Both operations tend to distort the shapes of the surfaces on which they operate, which is acceptable for the initialization because it will be followed by a surface deformation.

User-specified: For some applications it is desirable and easier for the user to interactively specify the initial model. Here, the user creates a Constructive Solid Geometry (CSG) model which defines the shape of the initial surface. In Figure 3a the CSG model in blue is interactively positioned relative to a Marching Cubes mesh extracted from the original dataset. The CSG model is scan-converted into a binary volume, with voxels simply marked as inside (1) or outside (0), using standard CSG evaluation techniques [40]. An isosurface of the initialization volume dataset generated from the torus and sphere is presented in Figure 3b. This volume dataset is then deformed to produce the final result seen in Figure 3c.

0.3.2 Level Set Surface Deformation

The initialization should position the model near the desired solution while retaining certain properties such as smoothness, connectivity, etc. Given a



a b c

Figure 3: (a) Interactively positioning a CSG model relative to a Marching Cubes mesh. (b) Isosurface of a binary scan conversion of the initialization CSG model. (c) Final internal embryo structures.

rough initial estimate, the surface deformation process moves the surface model toward specific features in the data. One must choose those properties of the input data to which the model will be attracted and what role the shape of the model will have in the deformation process. Typically, the deformation process combines a data term with a smoothing term, which prevents the solution from fitting too closely to noise-corrupted data. There are a variety of surface-motion terms that can be used in succession or simultaneously, in a linear combination to form $\mathcal{F}(\boldsymbol{x})$ in Equation 4.

Curvature: This is the smoothing term. For the work presented here we use the mean curvature of the isosurface H to produce

$$\mathcal{F}_{curv}(\boldsymbol{x}) = H = \left(\nabla \cdot \frac{\nabla \phi}{|\nabla \phi|} \right). \tag{6}$$

The mean curvature is also the normal variation of the surface area (i.e., minimal surface area). There are a variety of options for second-order smoothing terms [41], and the question of efficient, effective higher-order smoothing terms is the subject of on-going research [7, 42, 31]. For the work in this paper, we combine mean curvature with one of the following three terms, weighting it by a factor β, which is tuned to each specific application.

Edges: Conventional edge detectors from the image processing literature produce sets of "edge" voxels that are associated with areas of high con-

trast. For this work we use a gradient magnitude threshold combined with non-maximal suppression, which is a 3D generalization of the method of Canny [16]. The edge operator typically requires a scale parameter and a gradient threshold. For the scale, we use small, Gaussian kernels with standard deviation $\sigma = [0.5, 1.0]$ voxel units. The threshold depends on the contrast of the volume. The distance transform on this edge map produces a volume that has minima at those edges. The gradient of this volume produces a field that attracts the model to these edges. The edges are limited to voxel resolution because of the mechanism by which they are detected. Although this fitting is not sub-voxel accurate, it has the advantage that it can pull models toward edges from significant distances, and thus inaccurate initial estimates can be brought into close alignment with high-contrast regions, i.e. edges, in the input data. If $\mathcal{E}$ is the set of edges, and $D_{\mathcal{E}}(\boldsymbol{x})$ is the distance transform to those edges, then the movement of the surface model is given by

$$\mathcal{F}_{edge}(\boldsymbol{x}) = \boldsymbol{n} \cdot \nabla D_{\mathcal{E}}(\boldsymbol{x}). \tag{7}$$

Greyscale features—gradient magnitude: Surface models can also be attracted to certain greyscale features in the input data. For instance, the gradient magnitude indicates areas of high contrast in volumes. By following the gradient of such greyscale features, surface models are drawn to minimum or maximum values of that feature. Typically greyscale features, such as the gradient magnitude are computed with a scale operator, e.g., a derivative-of-Gaussian kernel. If models are properly initialized, they can move according to the *gradient of the gradient magnitude* and settle onto the edges of an object at a resolution that is finer than the original volume.

If $G(\boldsymbol{x})$ is some greyscale feature, for instance $G(\boldsymbol{x}) = |\nabla I(\boldsymbol{x})|$, where $I(\boldsymbol{x})$ is the input data (appropriately filtered—we use Gaussian kernels with $\sigma \approx 0.5$), then

$$\mathcal{F}_{grad}(\boldsymbol{x}) = \boldsymbol{n} \cdot (\pm \nabla G(\boldsymbol{x})), \tag{8}$$

where a positive sign moves surfaces towards maxima and the negative sign towards minima.

Isosurface: Surface models can also expand or contract to conform to isosurfaces in the input data. To a first order approximation, the distance from a point $\boldsymbol{x} \in U$ to the k-level surface of I is given by $(I(\boldsymbol{x}) - k) / |\nabla I|$. If we let $g(\alpha)$ be a fuzzy threshold, e.g., $g(\alpha) = \alpha / \sqrt{1 + \alpha^2}$, then

$$\mathcal{F}_{iso}(\boldsymbol{x}) = g\left(\frac{I(\boldsymbol{x}) - k}{|\nabla I|}\right) \tag{9}$$

causes the surfaces of ϕ to expand or contract to match the k isosurface of I. This term combined with curvature or one of the other fitting terms can create "quasi-isosurfaces" that also include other considerations, such as smoothness or edge strength.

0.3.3 Framework Results

Figure 4 presents one slice from an MRI scan of a mouse embryo, and an iso-surface model of its liver extracted from the unprocessed dataset. Figure 5 presents 3D renderings of the sequence of steps performed on the mouse MRI data to segment the liver. The first step is the initialization, which includes smoothing the input data, thresholding followed by a a flood fill to remove isolated holes, and finally applying morphological operators to remove small gaps and protrusions on the surface. The second (surface deformation) step first involves fitting to discrete edges and then to the gradient magnitude. This produces a significant improvement over the result in Figure 4. Figure 8a presents several other structures that were segmented from the mouse embryo dataset. The skin (grey) and the liver (blue) were isolated using computational initialization. The brain ventricles (red) and the eyes (green) were segmented with interactive initialization.

The same set of initialization and surface deformation steps may be combined to extract a model of a spiny dendrite from the TEM scan presented in Figure 6a. An iso-surface extracted from the scan is presented in Figure 6b. Figures 7 shows the results of the proposed method compared to the results of a manual segmentation, which took approximately 10 hours of slice-by-slice hand contouring. The manual method suffers from slice-wise artifacts, and, because of the size and complexity of the dataset, the manual segmentation is unable to capture the level of detail that we obtain with the surface-fitting results. Manual segmentation can, however, form connections that are not well supported by the data in order to complete the "spines" that cover this

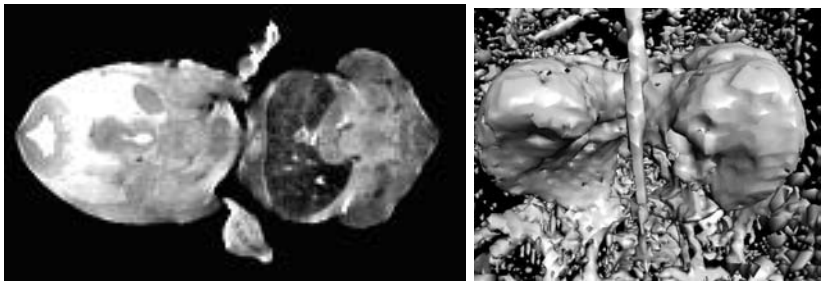


Figure 4: (left) One slice of a $256 \times 128 \times 128$ MR scan of a mouse embryo. The central dark structure is its liver. (right) A dual-threshold surface rendering hightlights the segmentation problem.

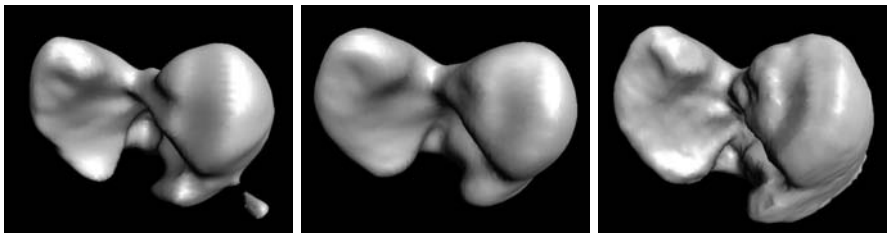


Figure 5: (left) The initialization of a mouse liver dataset using morphology to remove small pieces and holes. (center) Surface fitting to discrete edges. (right) The final fit to maxima of gradient magnitude.

dendrite. These types of "judgments" that humans make when they perform such tasks by hand are a mixed blessing. Humans can use high-level knowledge about the problem to fill in where the data is weak, but the expectations of a trained operator can interfere with seeing unexpected or unusual features in the data.

Figure 8 presents models from four samples of an MR series of a developing frog embryo. The top left image (Hour 9) shows the first evident structure, the blastocoel, in blue, surrounded by the outside casing of the embryo in grey. The top right image (Hour 16) demonstrates the expansion of the blastocoel and the development of the blastoporal lip in red. In the bottom left image (Hour 20) the blastoporal lip has collapsed, the blastocoel has contracted, and the archenteron in green has developed. In the bottom right image (Hour 30) the blastocoel has collapsed and only the archenteron is present. For this

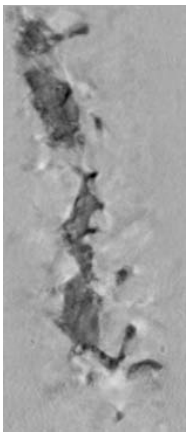


Figure 6a: One slice of a $154 \times 586 \times 270$ TEM scan of a spiny dendrite shows low contrast and high noise content in a relatively complex dataset.

Figure 6b: An isosurface rendering, with prefiltering, shows how noise and inhomogenieties in density interfere with visualizing the 3D structure of the dendrite.

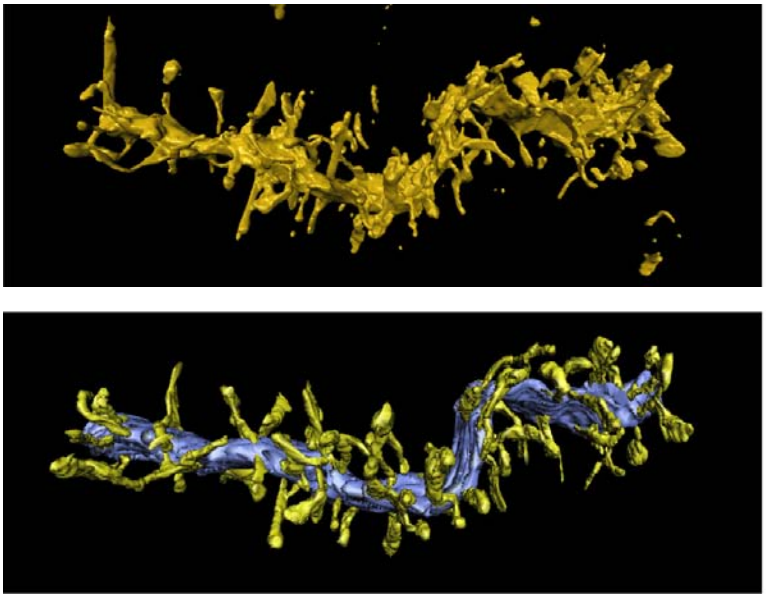


Figure 7: (top) Rendering of a dendrite segmented using our the proposed method. (bottom) Rendering of a manual segmentation of the same dendrite.

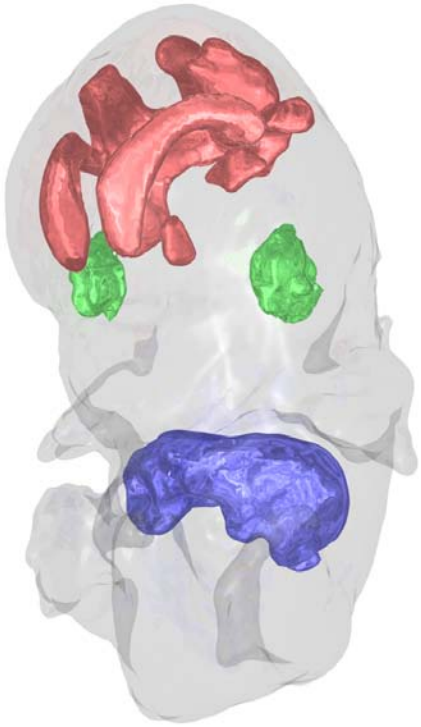


Figure 8a: Final mouse embryo model with skin (grey), liver (blue), brain ventricles (red), and eyes (green).

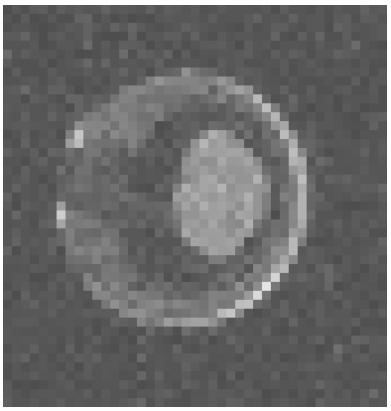


Figure 8b: Hour 16 dataset.

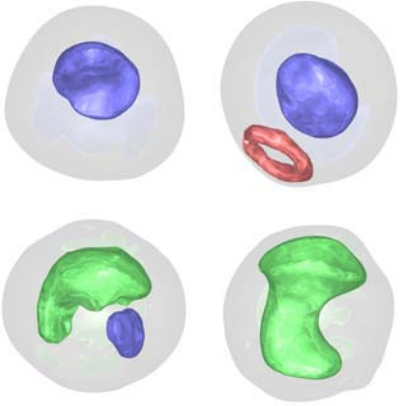


Figure 8c: Geometric structures extracted from MRI scans of a developing frog embryo, with blastocoel (blue), blastoporal lip (red), and archenteron (green). Hour 9 (top left). Hour 16 (top right). Hour 20 (bottom left). Hour 30 (bottom right).

dataset it was difficult to isolate structures only based on their voxel values. We therefore used our interactive techniques to isolate (during initialization) most of the structures in the frog embryo samples.

Table 1 describes for each dataset the specific techniques and parameters we used for the results in this section. These parameters were obtained by first making a sensible guess based on the contrasts and sizes of features in the data and then using trial and error to obtain acceptable results. Each dataset was processed between 4 and 8 times to achieve these results. More tuning could improve things further, and once these parameters are set, they work moderately well for similar modalities with similar subjects. The method is iterative, but the update times are proportional to the surface area. On an SGI 180MHz MIPS 10000 machine, the smaller mouse MR dataset required approximately 10 minutes of CPU time, and the dendrite dataset ran for approximately 45 minutes. Most of this time was spent in the initialization (which requires several complete passes through the data) and in the edge detection. The frog embryo datasets needed only a few minutes of processing time, because they did not require computational initialization and are significantly smaller than the other example datatsets.

0.4 Segmentation From Multiple Non-Uniform Volume Datasets

Many of today's volumetric datasets are generated by medical MR, CT and other scanners. A typical 3-D scan has a relatively high resolution in the scanning $X - Y$ plane, but much lower resolution in the axial Z direction. The difference in resolution between the in-plane and out-of-plane samplings can easily range between a factor of 5 to 10, see Figure **??**. This occurs both because of physical constraints on the thickness of the tissue to be excited during scanning (MR), total tissue irradiation (CT), and scanning time restrictions. Even when time is not an issue, most scanners are by design incapable of sampling with high resolution in the out-of-plane direction, producing anisotropic "brick-like" voxels.

The non-uniform sampling of an object or a patient can create certain problems. The inadequate resolution in the Z direction implies that small or thin structures will not be properly sampled, making it difficult to capture them

Dataset	Initialization	Surface Fitting
Dendrite	1. Gaussian blur $\sigma = 0.5$ 2. Threshold: $I < 127$ 3. Fill isolated holes 4. Morphology: $\mathrm{O}_{0.5} \circ \mathrm{C}_{1.5}$	1. Edge fitting: $\sigma = 0.75$, threshold $= 6$, $\beta = 0.1$ 2. Gradient magnitude fitting: $\sigma = 0.5$, $\beta = 1.0$
Mouse	1. Gaussian blur $\sigma = 0.5$ 2. Threshold: $I > 3$, $I < 60$ 3. Fill isolated holes 4. Morphology: $\mathrm{O}_{2.0} \circ \mathrm{C}_{3.0}$	1. Edge fitting: $\sigma = 0.75$, threshold $= 20$, $\beta = 2$ 2. Gradient magnitude fitting: $\sigma = 0.5$, $\beta = 16.0$
Frog	1. Interactive	1. Gradient magnitude fitting: $\sigma = 1.25$, $\beta = 1.0$

Table 1: Parameters for processing example datasets.

during surface reconstruction and object segmentation. One way to address this problem is to scan the same object from multiple directions, with the hope that the small structures will be adequately sampled in one of the scans. Generating several scans of the same object then raises the question of how to properly combine the information contained in these multiple datasets. Simply merging the individual scans does not necessarily assemble enough samples to produce a high resolution volumetric model. To address this problem we have developed a method for deforming a level set model using velocity information derived from multiple volume datasets with non-uniform resolution in order to produce a single high-resolution 3D model [43]. The method locally approximates the values of the multiple datasets by fitting a distance-weighted polynomial using moving least-squares (MLS) [44, 45]. Directional 3D edge information that may be used during the surface deformation stage is readily derived from MLS, and integrated within our segmentation framework.

The proposed method has several beneficial properties. Instead of merging all of the input volumes by global resampling (interpolation), we locally approximate the derivatives of the intensity values by MLS. This local versus global approach is feasible because the level set surface deformation only requires edge information in a narrow band around the surface. Consequently the MLS calculation is only performed in a small region of the volume, rather than throughout the whole volume, making the computational cost proportional to the object surface area [36]. As opposed to many interpolation schemes the MLS method is stable with respect to noise and imperfect registrations [46]. Our implementation also allows for small intensity attenuation artifacts between the multiple scans thereby providing gain-correction. The distance-based weighting employed in our method ensures that the contributions from each scan is properly merged into the final result. If a slice of data from one scan is closer to a point of interest on the model, the information from this scan will contribute more heavily to determining the location of the point.

To the best of our knowledge there is no previous work on creating deformable models directly from multiple volume datasets. While there has been previous work on 3D level set segmentation and reconstruction[41, 6, 5, 8, 47], it has not been based on multiple volume datasets. However, 3D models have been generated from multiple range maps [48, 49, 36, 29], but the 2D nature of these approaches is significantly different from the 3D problem being addressed

here. The most relevant related projects involve merging multiple volumes to produce a single high-resolution volume dataset [50, 51], and extracting edge information from a single non-uniform volume [52]. Our work does not attempt to produce a high-resolution merging of the input data. Instead, our contribution stands apart from previous work because it deforms a model based on local edge information derived from multiple non-uniform volume datasets.

We have demonstrated the effectiveness of our approach on three multi-scan datasets. The first two examples are derived from a single high resolution volume dataset that has been sub-sampled in the X, Y and Z directions. Since these non-uniform scans are extracted from a single dataset they are therefore perfectly aligned. The first scan is derived from a high resolution MR scan of a 12-day-old mouse embryo, which has already had its outer skin isolated with a previous segmentation process. The second example is generated from a laser scan reconstruction of a figurine. The third example consists of multiple MR scans of a zucchini that have been imperfectly aligned by hand. The first two examples show that our method is able to perform level set segmentation from multiple non-uniform scans of an object, picking up and merging features only found in one of the scans. The second example demonstrates that our method generates satisfactory results, even when there are misalignments in the registration.

0.4.1 Method Description

We have formulated our approach to 3D reconstruction of geometric models from multiple non-uniform volumetric datasets within our level set segmentation framework. Recall that speed function $\mathcal{F}()$ describes the velocity at each point on the evolving surface in the direction of the local surface normal. All of the information needed to deform a surface is encapsulated in the speed function, providing a simple, unified approach to evolving the surface. In this section we define speed functions that allow us to solve the multiple-data segmentation problem. The key to constructing suitable speed terms is 3D directional edge information derived from the multiple datasets. This problem is solved using a moving least-squares scheme that extracts edge information by locally fitting sample points to high-order polynomials.

0.4.1.1 Level Set Speed Function for Segmentation

Many different speed functions have been proposed over the years for segmentation of a single volume dataset [41, 6, 5, 8]. Typically such speed functions consist of a (3D) image-based feature attraction term and a smoothing term which serves as a regularization term that lowers the curvature and suppresses noise in the input data. From computer vision it is well known that features, *i.e.* significant changes in the intensity function, are conveniently described by an edge-detector [53]. There exists a very large body of work devoted to the problem of designing optimal edge detectors for 2D images [14, 16], most of which are readily generalized to 3D. For this project we found it convenient to use speed functions with a 3D directional edge term that moves the level set toward the maximum of the gradient magnitude. This gives a term equivalent to Equation 8,

$$\mathcal{F}_{grad}(\boldsymbol{x}, \boldsymbol{n}, \phi) = \alpha \boldsymbol{n} \cdot \nabla \|\nabla V_g\| \tag{10}$$

where α is a scaling factor for the *image-based* feature attraction term $\nabla\|\nabla V_g\|$ and $\boldsymbol{n}$ is the normal to the level set surface at $\boldsymbol{x}$. V_g symbolizes some global uniform merging of the multiple non-uniform input volumes. This feature term is effectively a 3D directional edge-detector of V_g. However there are two problems associated with using this speed function exclusively. The first is that we cannot expect to compute reliable 3D directional edge information in all regions of space simply because of the nature of the non-uniform input volumes. In other words V_g cannot be interpolated reliably in regions of space where there are no nearby sample points. Hence the level set surface will not experience any image-based forces in these regions. The solution is to use a regularization term that imposes constraints on the mean curvature of the deforming level set surface. We include the smoothing term from Equation 6 and scale it with parameter β, in order to smooth the regions where no edge information exists as well as suppress noise in the remaining regions thereby preventing excessive aliasing.

Normally the feature attraction term, $\nabla\|\nabla V_g\|$, creates only a narrow range of influence. In other words, this feature attraction term will only reliably move the portion of the level set surface that is in close proximity to the actual edges in V_g. Thus, a good initialization of the level set surface is needed before solving Equation 10. A reasonable initialization of the level set surface may be obtained by computing the CSG union of the multiple input volumes, which

are first tri-linearly resampled to give a uniform sampling. However, if the input volumes are strongly non-uniform, i.e. they are severely undersampled in one or more directions, their union produces a poor initial model. To improve the initialization we attract the CSG union surface to the Canny edges [16] computed from V_g using the distance transform produced from those edges. See Equation 7. This approach allows us to move the initial surface from a long range, but only with pixel-level accuracy.

Canny edges are non-directional edges defined from the zero-crossing of the second derivative of the image in the direction of the local normal. In 3D this is

$$\frac{\partial^2}{\partial \boldsymbol{n}_g^2} V_g = 0 \tag{11}$$

where $\boldsymbol{n}_g \equiv \nabla V_g / \|\nabla V_g\|$ is the local normal vector of V_g. Using the expression $\partial / \partial \boldsymbol{n}_g = \boldsymbol{n}_g \cdot \nabla$ we can rewrite Equation 11 as

$$\frac{\partial^2}{\partial \boldsymbol{n}_g^2} V_g = \boldsymbol{n}_g \cdot \nabla \left[\boldsymbol{n}_g \cdot \nabla V_g \right] = \boldsymbol{n}_g \cdot \nabla \|\nabla V_g\|. \tag{12}$$

The next section focuses on the methods needed to reliably compute the vectors $\boldsymbol{n}_g$ and $\nabla \|\nabla V_g\|$. In preparation, the latter may be explicitly expressed in terms of the derivatives of the merged volume V_g

$$\nabla \|\nabla V_g\| = \frac{\nabla V_g \ \widehat{H} V_g}{\|\nabla V_g\|} \tag{13}$$

where we have defined the gradient vector and the Hessian matrix,

$$\widehat{\nabla} V_g = \left(\frac{\partial V_g}{\partial x}, \frac{\partial V_g}{\partial y}, \frac{\partial V_g}{\partial z} \right) \tag{14a}$$

$$\widehat{H} V_g = \begin{pmatrix} \frac{\partial^2 V_g}{\partial x^2} & \frac{\partial^2 V_g}{\partial y \partial x} & \frac{\partial^2 V_g}{\partial z \partial x} \\ \frac{\partial^2 V_g}{\partial x \partial y} & \frac{\partial^2 V_g}{\partial y^2} & \frac{\partial^2 V_g}{\partial z \partial y} \\ \frac{\partial^2 V_g}{\partial x \partial z} & \frac{\partial^2 V_g}{\partial y \partial z} & \frac{\partial^2 V_g}{\partial z^2} \end{pmatrix}. \tag{14b}$$

Thus, in closing we note that the level set propagation needed for segmentation only needs information about the first and second order partial derivatives of the input volumes, not the interpolated intensity values themselves.

0.4.1.2 Computing Partial Derivatives

As outlined above the speed function $\mathcal{F}$ in the level set equation, Equation 4, is based on edge information derived from the input volumes. This requires

estimating first and second order partial derivatives from the multiple non-uniform input volumes. We do this by means of moving least-squares (MLS), which is an effective and well established numerical technique for computing derivatives of functions whose values are known only on irregularly spaced points [44, 45, 46].

Let us assume we are given the input volumes $\widehat{V}_d,\ d = 1, 2, .., D$ which are volumetric samplings of an object on the non-uniform grids $\{\widehat{\boldsymbol{x}}_d\}$. We shall also assume that the local coordinate frames of $\{\widehat{\boldsymbol{x}}_d\}$ are scaled, rotated and translated with respect to each other. Hence, we define a world coordinate frame (typically one of the local frames) in which we solve the level set equation. Now, let us define the world sample points $\{\boldsymbol{x}_d\}$ as

$$\boldsymbol{x}_d \equiv \mathbf{T}^{(d)}[\widehat{\boldsymbol{x}}_d] \tag{15}$$

where $\mathbf{T}^{(d)}$ is the coordinate transformation from a local frame d to the world frame. Next we locally approximate the intensity values from the input volumes $\widehat{V}_d$ with a 3D polynomial expansion. Thus, we define the N-order polynomials

$$V_N^{(d)}(\boldsymbol{x}) = C_{000}^{(d)} + \sum_{i+j+k=1}^{N} C_{ijk}^{(0)} x^i y^j z^k, \quad d = 1, 2, \ldots, D \tag{16}$$

where the C coefficients are unknown. Note that these local approximations to the intensity values share coefficients $C_{ijk}^{(0)}$ of order higher than zero, *i.e.* all of the functions $V_N^{(d)},\ d = 1, 2, .., D$ have the *same edges*. The fact that the zero-order term in Equation 16 is input volume dependent means we allow for local constant offsets between the input volumes $\widehat{V}_d$. This effectively provides built-in gain-correction in the scheme, since it can handle small intensity attenuation artifacts between the multiple scans.

Moving Least-Squares To solve for the expansion coefficients C in Equation 16 we define the moving least-squares functional

$$E(\boldsymbol{x}_0) = \sum_{d=1}^{D} \sum_{\boldsymbol{x}_d} w_d(\boldsymbol{x}_d - \boldsymbol{x}_0) \left[V_N^{(d)}(\boldsymbol{x}_d - \boldsymbol{x}_0) - V_d(\boldsymbol{x}_d) \right]^2 \tag{17}$$

where $\boldsymbol{x}_0$ is the expansion point from where we are seeking edge information, $V_d(\boldsymbol{x}_d) \equiv \widehat{V}_d(\widehat{\boldsymbol{x}}_d)$ and where

$$w_d(\boldsymbol{x}) \equiv \begin{cases} 1 - 2(\|\boldsymbol{x}\|/\Delta)^2 & \text{for } 0 \leq \|\boldsymbol{x}\| \leq \Delta/2 \\ 2(\|\boldsymbol{x}\|/\Delta - 1)^2 & \text{for } \Delta/2 < \|\boldsymbol{x}\| < \Delta \\ 0 & \text{for } \|\boldsymbol{x}\| \geq \Delta \end{cases} \tag{18}$$

is a "moving filter" that weights the contribution of different sampling points, $\boldsymbol{x}_d$, according to their Euclidean distance, $\|\boldsymbol{x}_d - \boldsymbol{x}_0\|$, to the expansion point, $\boldsymbol{x}_0$. Other expressions for this weighting function could of course be used, but Equation 18 is fast to compute, has finite support (by the window parameter Δ), and its tangent is zero at the endpoints. After substitution of Equation 16 into Equation 35 we obtain the functional

$$\begin{aligned} E(\boldsymbol{x}_0) = \sum_{d=1}^{D} \sum_{\boldsymbol{x}_d} w_d(\boldsymbol{x}_d - \boldsymbol{x}_0) \Big[C_{000}^{(d)} - \widehat{V}_d(\boldsymbol{x}_d) \\ + \sum_{i+j+k=1}^{N} C_{ijk}^{(0)} (x_d - x_0)^i (y_d - y_0)^j (z_d - z_0)^k \Big]^2 . \end{aligned} \tag{19}$$

The minimization of this moving least-squares functional with respect to the expansion coefficients C requires the partial derivatives to vanish, *i.e.*

$$\begin{aligned} \frac{\partial \widehat{E}(\boldsymbol{x}_0)}{\partial C_{000}^{(d)}} = 0 = 2 \sum_{\boldsymbol{x}_d} w_d(\boldsymbol{x}_d - \boldsymbol{x}_0) \Big[C_{000}^{(d)} - \widehat{V}_d(\boldsymbol{x}_d) \\ + \sum_{i+j+k=1}^{N} C_{ijk}^{(0)} (x_d - x_0)^i (y_d - y_0)^j (z_d - z_0)^k \Big] \end{aligned} \tag{20a}$$

$$\begin{aligned} \frac{\partial \widehat{E}(\boldsymbol{x}_0)}{\partial C_{lnm}^{(0)}} = 0 = 2 \sum_{d=1}^{D} \sum_{\boldsymbol{x}_d} w_d(\boldsymbol{x}_d - \boldsymbol{x}_0) \Big[C_{000}^{(d)} - \widehat{V}_d(\boldsymbol{x}_d) \\ + \sum_{i+j+k=1}^{N} C_{ijk}^{(0)} (x_d - x_0)^i (y_d - y_0)^j (z_d - z_0)^k \Big] \\ \times (x_d - x_0)^l (y_d - y_0)^m (z_d - z_0)^n . \end{aligned} \tag{20b}$$

This defines a system of linear equations in the expansion coefficients $C_{ijk}^{(r)}$, that can be solved using standard techniques from numerical analysis, see Equation 21 and Equation 23.

Equation 20a and Equation 20b can then be conveniently expressed as

$$\sum_{\boldsymbol{q}} \mathbf{A}_{\boldsymbol{p},\boldsymbol{q}}\, \boldsymbol{c}_{\boldsymbol{q}} = \boldsymbol{b}_{\boldsymbol{p}} \tag{21}$$

where $\mathbf{A}$ is a diagonal matrix, and $\boldsymbol{b}$, $\boldsymbol{c}$ are vectors. In this equation we have also introduced the compact index notations $\boldsymbol{p} \equiv (i,j,k,r)$ and $\boldsymbol{q} \equiv (l,m,n,s)$ defined as

$$\begin{aligned}
\boldsymbol{p} \in & \left\{ i,j,k,r \in \mathcal{N}^+ \middle|\, i=j=k=0,\ 1 \le r \le D \right\} \\
& \cup \left\{ i,j,k,r \in \mathcal{N}^+ \middle|\, 1 \le i+j+k \le N,\ r=0 \right\} && \text{(22a)} \\
\boldsymbol{q} \in & \left\{ l,m,n,s \in \mathcal{N}^+ \middle|\, l=m=n=0,\ 1 \le s \le D \right\} \\
& \cup \left\{ l,m,n,s \in \mathcal{N}^+ \middle|\, 1 \le l+m+n \le N,\ s=0 \right\}. && \text{(22b)}
\end{aligned}$$

The diagonal matrix $\mathbf{A}$, and the vectors $\boldsymbol{b}$, $\boldsymbol{c}$ in Equation 21 are defined as

$$\begin{aligned}
\mathbf{A}_{\boldsymbol{p},\boldsymbol{q}} \equiv & \sum_d \left(\delta_{r,d} + \delta_{r,0}\right)\left(\delta_{s,d} + \delta_{s,0}\right) \sum_{\boldsymbol{x}_d} w_d(\boldsymbol{x}_d - \boldsymbol{x}_0) \\
& \times (x_d - x_0)^i (y_d - y_0)^j (z_d - z_0)^k && \text{(23a)} \\
& \times (x_d - x_0)^l (y_d - y_0)^m (z_d - z_0)^n \\
\boldsymbol{b}_{\boldsymbol{p}} \equiv & \sum_d \left(\delta_{r,d} + \delta_{r,0}\right) w_d(\boldsymbol{x}_d - \boldsymbol{x}_0) \widehat{V}_d(\boldsymbol{x}_d) \\
& \times (x_d - x_0)^i (y_d - y_0)^j (z_d - z_0)^k && \text{(23b)} \\
\boldsymbol{c}_{\boldsymbol{p}} \equiv & \, C^{(r)}_{ijk}. && \text{(23c)}
\end{aligned}$$

Next the matrix equation $\mathbf{A}\boldsymbol{c} = \boldsymbol{b}$ must be solved for the vector $\boldsymbol{c}$ of dimension $\binom{N+3}{3} + D - 1$, where N is the order of the expansion in Equation 16 and D is the number of non-uniform input volumes. As is well known for many moving least-square problems it is possible for the condition number of the matrix $\mathbf{A}$ to become very large. Any matrix is singular if its condition number is infinite and can be defined as ill-conditioned if the reciprocal of its condition number approaches the computer's floating-point precision. This can occur if the problem is over-determined (number of sample points, $\boldsymbol{x}_d$ greater than number of coefficients C) and under-determined (ambiguous combinations of the coefficients C work equally well or equally bad). To avoid such numerical problems, a singular value decomposition (SVD) linear equation solver is recommended for use in combination with the moving least-squares method. The

SVD solver identifies equations in the matrix $\mathbf{A}$ that are, within a specified tolerance, redundant (*i.e.* linear combinations of the remaining equations) and eliminates them thereby improving the condition number of the matrix. We refer the reader to reference [54] for a helpful discussion of SVD pertinent to linear least-squares problems.

Once we have the expansion coefficients $\boldsymbol{c}$ we can readily express the Hessian matrix and the gradient vector of the combined input volumes as

$$\nabla V = (C^{(0)}_{100}, C^{(0)}_{010}, C^{(0)}_{001}) \tag{24a}$$

$$\mathbf{H}V = \begin{pmatrix} 2C^{(0)}_{200} & C^{(0)}_{110} & C^{(0)}_{101} \\ C^{(0)}_{110} & 2C^{(0)}_{020} & C^{(0)}_{011} \\ C^{(0)}_{101} & C^{(0)}_{011} & 2C^{(0)}_{002} \end{pmatrix} \tag{24b}$$

evaluated at the moving expansion point $\boldsymbol{x}_0$. This in turn is used in Equation 13 to compute the edge information needed to drive the level set surface.

0.4.1.3 Algorithm Overview

Algorithm 1 describes the main steps of our approach. The initialization routine, Algorithm 2, is called for all of the multiple non-uniform input volumes, V_d. Each non-uniform input dataset is uniformly resampled in a common coordinate frame (V_0's) using tri-linear interpolation. Edge information and the union, V_0, of the V_d's is then computed. Algorithm 2 calculates Canny and 3D directional edge information using moving least-squares in a narrow band in each of the resampled input volumes, V_d, and buffers this in V_{edge} and $\boldsymbol{V}_{grad}$. Next Algorithm 1 computes the distance transform of the zero-crossings of the Canny edges and takes the gradient of this scalar volume to produce a vector field $\boldsymbol{V}_{edge}$, which pulls the initial level set model to the Canny edges. Finally the level set model is attracted to the 3D directional edges of the multiple input volumes, $\boldsymbol{V}_{grad}$, and a Marching Cubes mesh is extracted for visualization. The level set solver, described in Algorithm 3, solves Equation 4 using the "upwind scheme" (not explicitly defined) and the sparse-field narrow-band method of [36], with V_0 as the initialization and $\boldsymbol{V}_{edge}$ and $\boldsymbol{V}_{grad}$ as the force field in the speed function.

Algorithm 1: MAIN$(V_1, \ldots, V_D)$

comment: $V_1, \ldots, V_D$ are non-uniform samplings of object V

global $V_{edge}, \boldsymbol{V}_{grad}$

do
- $V_0 \leftarrow$ uniform sampling of empty space
- **for** $d \leftarrow 1$ **to** D
  - **do** $V_0 \leftarrow V_0 \cup$ INITIALIZATION(V_d)
- $\boldsymbol{V}_{edge} \leftarrow \boldsymbol{\nabla}$[distance transform[zero-crossing[V_{edge}]]]
- $V_0 \leftarrow$ SOLVELEVELSETEQ$(V_0, \boldsymbol{V}_{edge}, \alpha, 0)$
- $V_0 \leftarrow$ SOLVELEVELSETEQ$(V_0, \boldsymbol{V}_{grad}, \alpha, \beta)$

return (Marching Cubes mesh of V_0)

Algorithm 2: INITIALIZATION(V_d)

comment: Pre-processing to produce good LS initialization

do
- $V_d \leftarrow$ Uniform tri-linear resampling of V_d
- $\Gamma_d \leftarrow$ Set of voxels in narrow band of iso-surface of V_d
- **for each** $\boldsymbol{x}_0 \in \Gamma_d$
  - **do**
    - Solve moving least-squares problem at $\boldsymbol{x}_0$
    - $V_{edge}(\boldsymbol{x}_0) \leftarrow$ scalar Canny edge, *cf.* Equation 12
    - $\boldsymbol{V}_{grad}(\boldsymbol{x}_0) \leftarrow$ 3D directional edge, *cf.* Equation 13

return (V_d)

Algorithm 3: SOLVELEVELSETEQ$(V_0, \boldsymbol{V}, \alpha, \beta)$

comment: Solve Equation 4 with initial condition $\phi(t{=}0) = V_0$

do
- $\phi \leftarrow V_0$
- **repeat**
  - $\Gamma \leftarrow$ Set of voxels in narrow band of iso-surface of ϕ
  - $\Delta t \leftarrow \gamma / \sup_{\boldsymbol{x} \in \Gamma} \|\boldsymbol{V}(\boldsymbol{x})\|, \ \gamma \leq 1$
  - **for each** $\boldsymbol{x} \in \Gamma$
    - **do**
      - $\boldsymbol{n} \leftarrow$ upwind scheme$[-\nabla\phi(\boldsymbol{x}) / \|\nabla\phi(\boldsymbol{x})\|]$
      - $\dot{\phi}(\boldsymbol{x}) \leftarrow \|\nabla\phi(\boldsymbol{x})\|(\alpha \boldsymbol{V}(\boldsymbol{x}) \cdot \boldsymbol{n} + \beta \nabla \cdot \boldsymbol{n})$
      - $\phi(\boldsymbol{x}) \leftarrow \phi(\boldsymbol{x}) + \dot{\phi}(\boldsymbol{x}) \Delta t$
  - **until** $\sup_{\boldsymbol{x} \in \Gamma} \|\dot{\phi}(\boldsymbol{x})\| \leq \epsilon$

return (ϕ)

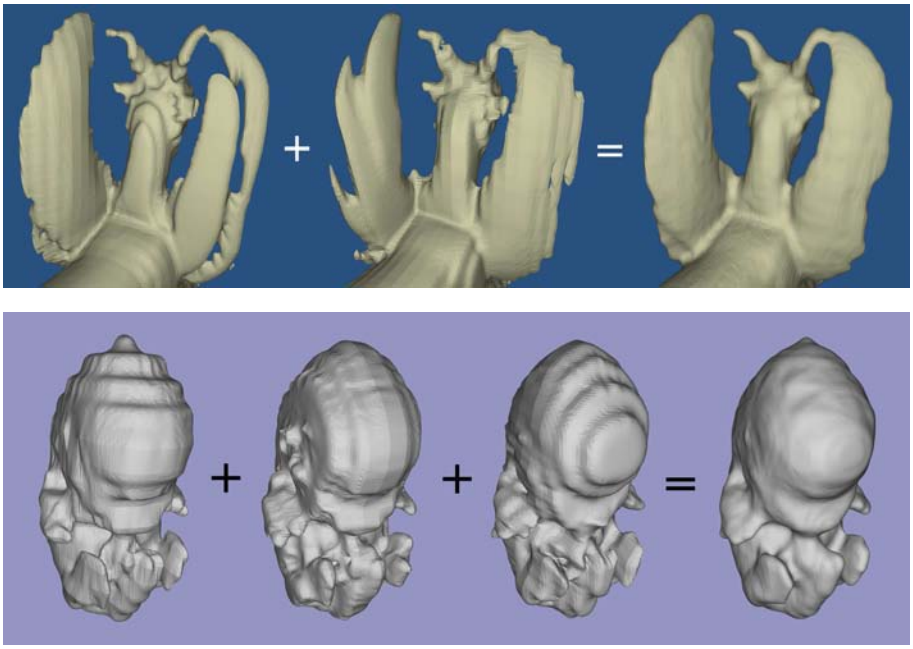


Figure 9: Non-uniform datasets merged to produce high resolution level set models, (top) laser scan of a figurine, (bottom) MR scan of a mouse embryo.

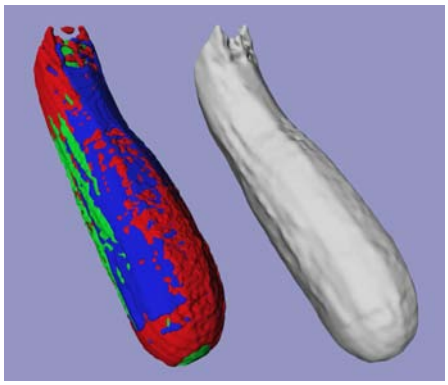


Figure 10: Three low resolution MR scans of a zucchini that have been individually colored and overlaid to demonstrate their imperfect alignment. The level set model on the right is derived from the three low resolution scans.

0.4.2 Multiple Volume Results

We have applied our segmentation method to several multi-scan non-uniform datasets to produce high resolution level set models. The parameters used for these segmentations are listed in Table 2. α and β are weights that the user adjusts to balance attraction to edges with curvature-based smoothing during the level set deformation process.

Table 2: Maximum in-plane to out-of-plane sampling ratios of non-uniform input datasets, and parameters for the two level set speed terms defined in Equation 6 and Equation 10.

Model	Origin	Ratio	α	β
Griffin	Laser scan	6/10:1	1.0	0.5
Mouse	MR scan	10:1	1.0	0.5
Zucchini	MR scan	10:1	1.0	0.5

0.4.2.1 Griffin Dataset

The griffin dataset was created with a volumetric laser scan reconstruction algorithm [49]. This algorithm creates a high resolution volumetric representation of an object by merging multiple depth maps produced via a laser scan. The original griffin dataset has a resolution of $312 \times 294 \times 144$. We have extracted two non-uniform datasets from this high resolution representation by copying every sixth plane of data in the X direction and every tenth plane in the Y direction. The two derived non-uniform griffin datasets have the following resolution: $52 \times 294 \times 144$ and $312 \times 30 \times 144$. Iso-surfaces have been extracted from these datasets, appropriately scaled in the low resolution direction, and are presented in the first two images in Figure 9 (top). Each low resolution scan inadequately captures some important geometric feature of the griffin. We have performed a reconstructions from the undersampled non-uniform scans to produce the result in Figure 9 (top). The method produces a high resolution ($312 \times 294 \times 144$) level set model that contains all of the significant features of the original scan.

0.4.2.2 Mouse Embryo Dataset

The first three scans in Figure 9 (bottom) are derived from a high resolution MR scan of a mouse embryo. They are subsampled versions of a 256×128×128 volume dataset, and have the following resolutions: 26×128×128, 256×16×128 and 256×128×13. The last image in Figure 9 presents the result produced by our multi-scan segmentation method. The information in the first three scans has been successfully used to create a level set model of the embryo with a resolution of 256×128×130. The finer features of the mouse embryo, namely its hands and feet, have been reconstructed.

0.4.2.3 Zucchini Dataset

The final dataset consists of three individual MRI scans of an actual zucchini. The separate scans have been registered manually and are presented on the left side of Figure 10, each with a different color. The resolutions of the individual scans are 28×218×188, 244×25×188 and 244×218×21. This image highlights the rough alignment of the scans. The right side of Figure 10 presents the result of our level set segmentation. It demonstrates that our approach is able to extract a reasonable model from multiple datasets that are imperfectly aligned.

0.5 Segmentation of DT-MRI Brain Data

Diffusion tensor magnetic resonance imaging[55, 56] (DT-MRI) is a technique used to measure the diffusion properties of water molecules in tissues. Anisotropic diffusion can be described by the equation

$$\frac{\partial C}{\partial t} = \nabla \cdot (\mathbf{D} \nabla C) \tag{25}$$

where C is the concentration of water molecules and $\mathbf{D}$ is a diffusion coefficient, which is a symmetric second order tensor

$$\mathbf{D} = \begin{pmatrix} D_{xx} & D_{xy} & D_{xz} \\ D_{yx} & D_{yy} & D_{yz} \\ D_{zx} & D_{zy} & D_{zz} \end{pmatrix}. \tag{26}$$

Figure 11 presents a "slice" of the diffusion tensor volume data of human brain used in our study. Each sub-image presents the scalar values of the associated diffusion tensor component for one slice of the dataset.

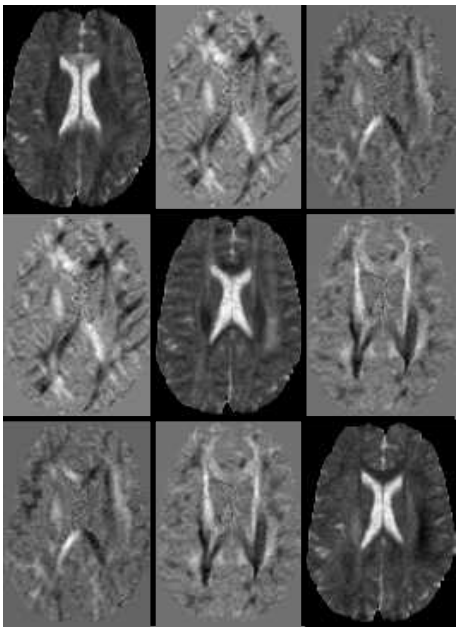


Figure 11: Slice of a tensor volume where every "element" of the image matrix corresponds to one component of the tensor **D**.

Tissue segmentation and classification based on DT-MRI offers several advantages over conventional MRI, since diffusion data contains additional physical information about the internal structure of the tissue being scanned. However, segmentation and visualization using diffusion data is not entirely straightforward. First of all, the diffusion matrix itself is not invariant with respect to rotations, and the elements that form the matrix will be different for different orientations of the sample or field gradient and therefore cannot themselves be used for classification purposes. Moreover, 3D visualization and segmentation techniques available today are predominantly designed for scalar and sometimes vector fields. Thus, there are three fundamental problems in tensor imaging: a) finding an invariant representation of a tensor that is independent of a frame of reference, b) constructing a mapping from the tensor field to a scalar or vector field, and c) visualization and classification of tissue using the derived scalar fields.

The traditional approaches to diffusion tensor imaging involve converting the tensors into an eigenvalue/eigenvector representation, which is rotationally

invariant. Every tensor may then be interpreted as an ellipsoid with principal axes oriented along the eigenvectors and radii equal to the corresponding eigenvalues. This ellipsoid describes the probabilistic distribution of a water molecule after a fixed diffusion time.

Using eigenvalues/eigenvectors one can compute different anisotropy measures [55, 57, 58, 59] that map tensor data onto scalars and can be used for further visualization and segmentation. Although eigenvalue/vector computation of the 3x3 matrix is not expensive, it must be repeatedly performed for every voxel in the volume. This calculation easily becomes a bottleneck for large datasets. For example, computing eigenvalues and eigenvectors for a 512^3 volume requires over 20 CPU-minutes on a powerful workstation. Another problem associated with eigenvalue computation is stability - a small amount of noise will not only change the values but also the ordering of the eigenvalues[60]. Since many anisotropy measures depend on the ordering of the eigenvalues, the calculated direction of diffusion and classification of tissue will be significantly altered by the noise normally found in diffusion tensor datasets. Thus it is desirable to have an anisotropy measure which is rotationally invariant, does not require eigenvalue computations and is stable with respect to noise. Tensor invariants with these characteristics were first proposed by Ulug and Zijl[61]. In Section 0.5.1 we formulate a new anisotropy measure for tensor field based on these invariants.

Visualization and model extraction from the invariant 3D scalar fields is the second issue addressed in this paper. One of the popular approaches to tensor visualization represents a tensor field by drawing ellipsoids associated with the eigenvectors/values[62]. This method was developed for 2D slices and creates visual cluttering when used in 3D. Other standard CFD visualization techniques like tensor-lines do not provide meaningful results for the MRI data due to rapidly changing directions and magnitudes of eigenvector/values and the amount of noise present in the data. Recently Kindlmann[63] developed a volume rendering approach to tensor field visualization using eigenvalue-based anisotropy measures to construct transfer functions and color maps that highlight some brain structures and diffusion patterns.

In our work we perform iso-surfacing on the 3D scalar fields derived from our tensor invariants to visualize and segment the data [64]. An advantage of iso-surfacing over other approaches is that it can provide the shape information

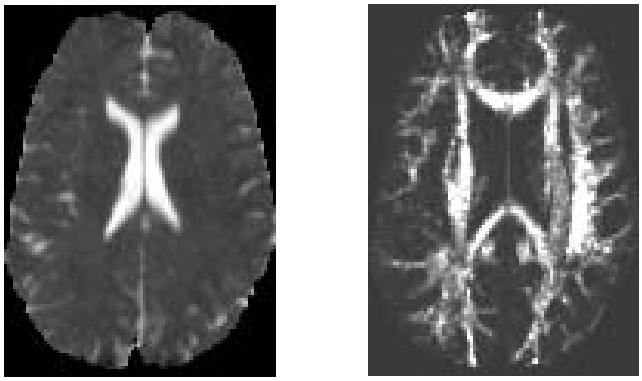


Figure 12: Isotropic C_1 (left) and anisotropic C_a (right) tensor invariants for the tensor slice shown in Figure 11.

needed for constructing geometric models, and computing internal volumes and external surface areas of the extracted regions. There has also been a number of recent publications[65, 66] devoted to brain fiber tracking. This is a different and more complex task than the one addressed in this paper and requires data with a much higher resolution and better signal-to-noise ratio than the data used in our study.

0.5.1 Tensor Invariants

Tensor invariants (rotational invariants) are combinations of tensor elements that do not change after the rotation of the tensor's frame of reference, and thus do not depend on the orientation of the patient with respect to the scanner when performing DT imaging. The well known invariants are the eigenvalues of the diffusion tensor (matrix) D, which are the roots of corresponding characteristic equation

$$\lambda^3 - C_1 \cdot \lambda^2 + C_2 \cdot \lambda - C_3 = 0, \tag{27}$$

with coefficients

$$\begin{aligned} C_1 &= D_{xx} + D_{yy} + D_{zz} \\ C_2 &= D_{xx}D_{yy} - D_{xy}D_{yx} + D_{xx}D_{zz} - D_{xz}D_{zx} + \\ &\quad D_{yy}D_{zz} - D_{yz}D_{zy} \\ C_3 &= D_{xx}(D_{yy}D_{zz} - D_{zy}D_{yz}) \\ &- D_{xy}(D_{yx}D_{zz} - D_{zx}D_{yz}) + D_{xz}(D_{yx}D_{zy} - D_{zx}D_{yy}). \end{aligned} \tag{28}$$

Since the roots of Equation (27) are rotational invariants, the coefficients C_1, C_2 and C_3 are also invariant. In the eigen- frame of reference they can be easily expressed through the eigenvalues

$$\begin{aligned} C_1 &= \lambda_1 + \lambda_2 + \lambda_3 \\ C_2 &= \lambda_1\lambda_2 + \lambda_1\lambda_3 + \lambda_2\lambda_3 \\ C_3 &= \lambda_1\lambda_2\lambda_3 \end{aligned} \tag{29}$$

and are proportional to the sum of the radii, surface area and the volume of the "diffusion" ellipsoid. Then instead of using $(\lambda_1, \lambda_2, \lambda_3)$ to describe the dataset, we can use (C_1, C_2, C_3). Moreover, since C_i are the coefficients of the characteristic equation, they are less sensitive to noise, then roots λ_i of the same equation.

Any combination of the above invariants is, in turn, an invariant. We consider the following dimensionless combination: C_1C_2/C_3. In the eigenvector frame of reference it becomes

$$\frac{C_1C_2}{C_3} = 3 + \frac{\lambda_2 + \lambda_3}{\lambda_1} + \frac{\lambda_1 + \lambda_3}{\lambda_2} + \frac{\lambda_1 + \lambda_2}{\lambda_3} \tag{30}$$

and we can define a new dimensionless anisotropy measure

$$C_a = \frac{1}{6}\left[\frac{C_1C_2}{C_3} - 3\right]. \tag{31}$$

It is easy to show that for isotropic diffusion, when $\lambda_1 = \lambda_2 = \lambda_3$, the coefficient $C_a = 1$. In the anisotropic case, this measure is identical for both linear, directional diffusion ($\lambda_1 >> \lambda_2 \approx \lambda_3$) and planar diffusion ($\lambda_1 \approx \lambda_2 >> \lambda_3$) and is equal to

$$C_a^{limit} \approx \frac{1}{3}\left[1 + \frac{\lambda_1}{\lambda_3} + \frac{\lambda_3}{\lambda_1}\right]. \tag{32}$$

Thus C_a is always $\sim \lambda_{max}/\lambda_{min}$ and measures the magnitude of the diffusion anisotropy. We again want to emphasize that we use the eigenvalue representation here only to analyze the behavior of the coefficient C_a, but we use invariants (C_1, C_2, C_3) to compute it using Equations (5) and (31).

0.5.2 Geometric Modeling

Two options are usually available for viewing the scalar volume datasets, direct volume rendering[1, 4] and volume segmentation[67] combined with con-

ventional surface rendering. The first option, direct volume rendering, is only capable of supplying images of the data. While this method may provide useful views of the data, it is well-known that it is difficult to construct the exact transfer function that highlights the desired structures in the volume dataset[68]. Our approach instead focuses on extracting geometric models of the structures embedded in the volume datasets. The extracted models may be used for interactive viewing, but the segmentation of geometric models from the volume datasets provides a wealth of additional benefits and possibilities. The models may be used for quantitative analysis of the segmented structures, for example the calculation of surface area and volume; quantities that are important when studying how these structures change over time. The models may be used to provide the shape information necessary for anatomical studies and computational simulation, for example EEG/MEG modeling within the brain[69]. Creating separate geometric models for each structure allows for the straightforward study of the relationship between the structures, even though they come from different datasets. The models may also be used within a surgical planning/simulation/VR environment[70], providing the shape information needed for collision detection and force calculations. The geometric models may even be used for manufacturing real physical models of the structures[71]. It is clear that there are numerous reasons to develop techniques for extracting geometric models from diffusion tensor volume datasets.

The most widely used technique for extracting polygonal models from volume datasets is the Marching Cubes algorithm[72]. This technique creates a polygonal model that approximates the iso-surface embedded in a scalar volume dataset for a particular iso-value. While the Marching Cubes algorithm is easy to understand and straightforward to implement, applying it directly to raw volume data from scanners can produce undesirable results, as seen in the first images in Figures 13 and 16. The algorithm is susceptible to noise and can produce many unwanted triangles that mask the central structures in the data. In order to alleviate this problem, we utilize the tools in our level set framework to smooth the data and remove the noise-related artifacts.

0.5.3 Segmentation

In this section we demonstrate the application of our methods to the segmentation of DT-MRI data of the human head. We use a high resolution dataset from

a human volunteer which contains 60 slices each of 128x128 pixels resolution. The raw data is sampled on a regular uniform grid.

We begin by generating two scalar volume datasets based on the invariants described in Section 0.5.1. The first scalar volume dataset ($\mathcal{V}_1$) is formed by calculating the trace (C_1) of the tensor matrix for each voxel of the diffusion tensor volume. It provides a single number that characterizes the total diffusivity at each voxel within the sample. Higher values signify greater total diffusion irrespective of directionality in the region represented by a particular voxel. A slice from this volume can be seen in Figure 12 (left). The second scalar volume dataset ($\mathcal{V}_2$) is formed by calculating (C_1, C_2, C_3) invariants for each voxel and combining them into C_a. It provides a measure of the magnitude of the anisotropy within the volume. Higher values identify regions of greater spatial anisotropy in the diffusion properties. A slice from the second scalar volume is presented in Figure 12 (right). The measure C_a does not by definition distinguish between linear and planar anisotropy. This is sufficient for our current study since the brain does not contain measurable regions with planar diffusion anisotropy. We therefore only need two scalar volumes in order to segment the DT dataset.

We then utilize our level set framework to extract smoothed models from the two derived scalar volumes. First the input data is filtered with a low-pass Gaussian filter ($\sigma \approx 0.5$) to blur the data and thereby reduce noise. Next, the volume voxels are classified for inclusion/exclusion in the initialization based on the filtered values of the input data ($k \approx 7.0$ for $\mathcal{V}_1$ and $k \approx 1.3$ for $\mathcal{V}_2$). For grey scale images, such as those used in this paper, the classification is equivalent to high and low thresholding operations. The last initialization step consists of performing a set of topological (e.g. flood fill) operations in order to remove small pieces or holes from objects. This is followed by a level set deformation that pulls the surface toward local maxima of the gradient magnitude and smooths it with a curvature-based motion. This moves the surface toward specific features in the data, while minimizing the influence of noise in the data.

Figures 13 and 14 present two models that we extracted from DT-MRI volume datasets using our techniques. Figure 13 contains segmentations from volume $\mathcal{V}_1$, the measure of total diffusivity. The top image shows a Marching Cubes iso-surface using an iso-value of 7.5. In the bottom we have extracted

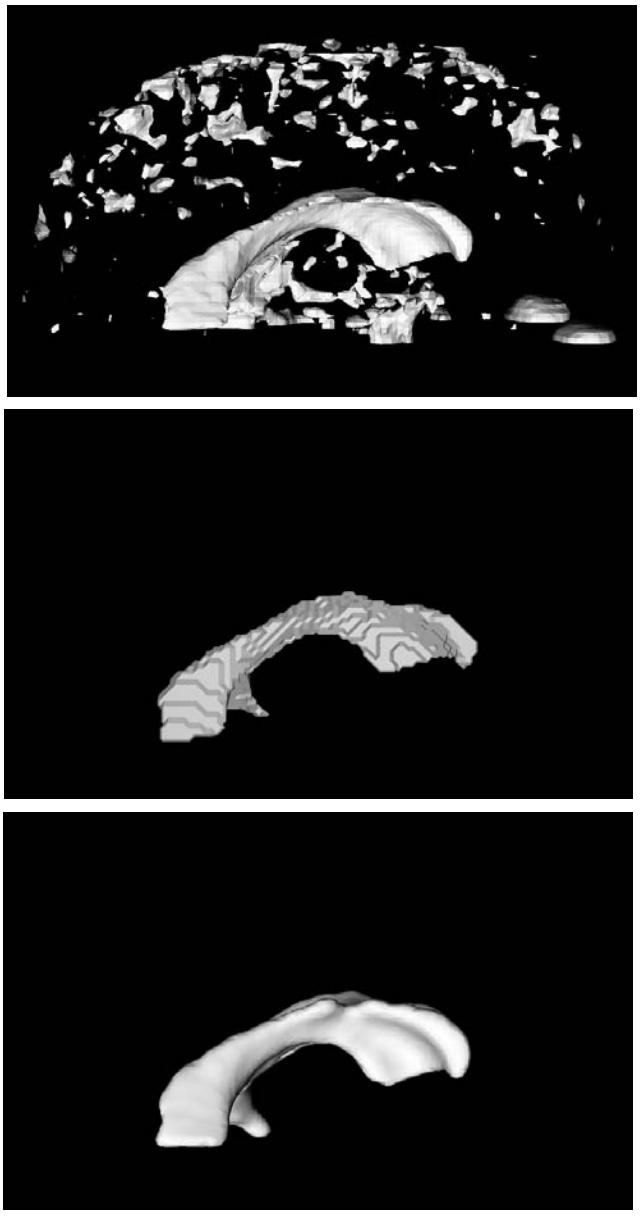


Figure 13: Segmentation from isotropic measure volume $\mathcal{V}_1$ for the first DT-MRI dataset. The first row is the marching cubes iso-surface with iso-value 7.5. The second row is the result of flood-fill algorithm applied to the same volume and used for initialization. The third row is the final level set model.

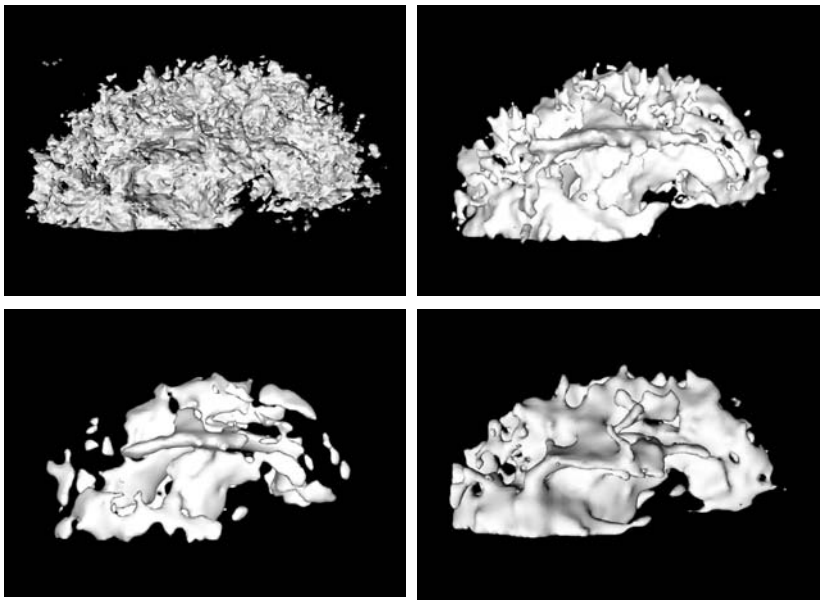


Figure 14: Model segmentation from volume $\mathcal{V}_2$. Top left image is an iso-surface of value 1.3, used for initialization of the level set. Clockwise, are the results of level set development with corresponding β values of 0.2, 0.4 and 0.5.

just the ventricles from $\mathcal{V}_1$. This is accomplished by creating an initial model with a flood-fill operation inside the ventricle structure shown in the middle image. This identified the connected voxels with value of 7.0 or greater. The initial model was then refined and smoothed with a level set deformation, using a β value of 0.2.

Figure 14 again provides the comparison between direct iso-surfacing and and level set modeling, but on the volume $\mathcal{V}_2$. The image in the top-left corner is a Marching Cubes iso-surface using an iso-value of 1.3. There is significant high-frequency noise and features in this dataset. The challenge here was to isolate coherent regions of high anisotropic diffusion. We applied our segmentation approach to the dataset and worked with neuroscientists from LA Childrens Hospital, City of Hope Hospital and Caltech to identify meaningful anatomical structures. We applied our approach using a variety of parameter values, and presented our results to them, asking them to pick the model that they felt best represented the structures of the brain. Figure 14 contains three models extracted from $\mathcal{V}_2$ at different values of smoothing parameter β used during

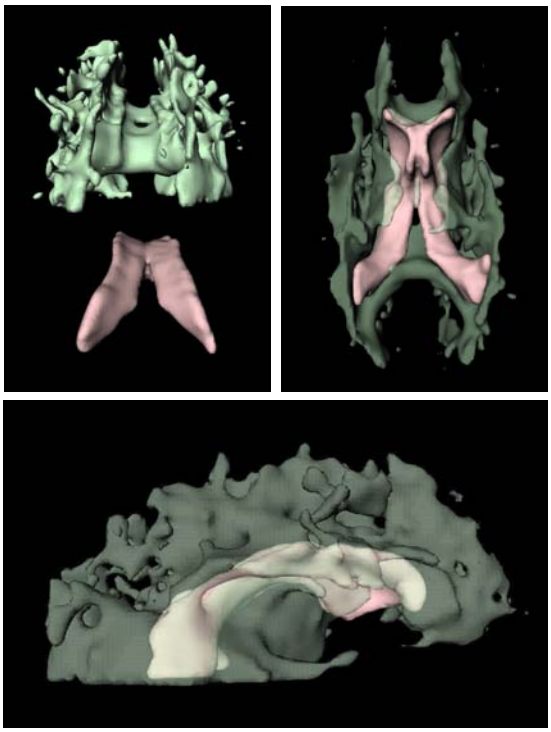


Figure 15: Combined model of ventricles and (semi-transparent) anisotropic regions: rear, exploded view (left), bottom view (right), side view (bottom). Note how model of ventricles extracted from isotropic measure dataset $\mathcal{V}_1$ fits into model extracted from anisotropic measure dataset $\mathcal{V}_2$.

segmentation. Since we were not looking for a single connected structure in this volume, we did not use a seeded flood-fill for initialization. Instead we initialized the deformation process with an iso-surface of value 1.3. This was followed by a level set deformation using a β value of 0.2. The result of this segmentation is presented on the bottom-left side of Figure 14. The top-right side of this figure presents a model extracted from $\mathcal{V}_2$ using an initial iso-surface of value 1.4 and a β value of 0.5. The result chosen as the "best" by our scientific/medical collaborators is presented on the bottom-right side of Figure 14. This model is produced with an initial iso-surface of 1.3 and a β value of 0.4. Our collaborators were able to identify structures of high diffusivity in this model, for example the corpus callosum, the internal capsul, the optical nerve tracks, and other white matter regions.

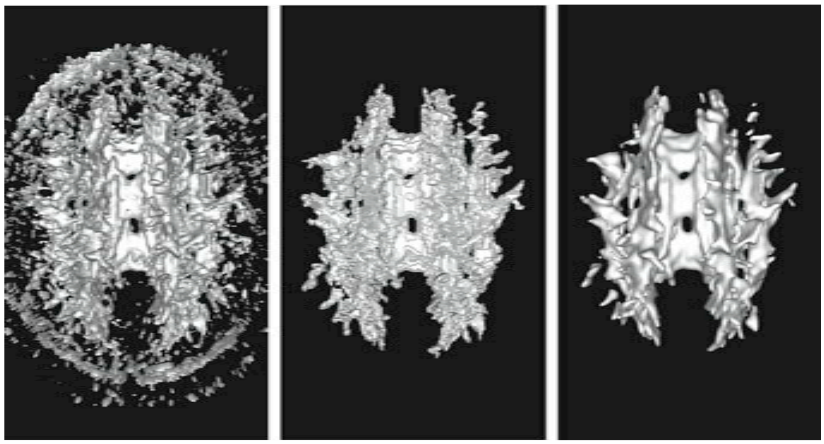


Figure 16: Segmentation using anisotropic measure $\mathcal{V}_2$ from the second DT-MRI dataset. (left) Marching cubes iso-surface with iso-value 1.3. (middle) Result of flood-fill algorithm applied to the volume and used for initialization. (right) Final level set model.

We can also bring together the two models extracted from datasets $\mathcal{V}_1$ and $\mathcal{V}_2$ into a single image. They will have perfect alignment since they are derived from the same DT-MRI dataset. Figure 15 demonstrates that we are able to isolate different structures in the brain from a single DT-MRI scan and show their proper spatial inter-relationship. For example, it can be seen that the corpus callosum lies directly on top of the ventricles, and that the white matter fans out from both sides of the ventricles.

Finally, to verify the validity of our approach we applied it to the second dataset from a different volunteer. This dataset has 20 slices of the 256x256 resolution. We generated the anisotropy measure volume $\mathcal{V}_2$ and performed the level set model extraction using the same iso-values and smoothing parameters as for $\mathcal{V}_2$. The results are shown in Figure 16, and demonstrate the generality of our approach.

0.6 Direct Estimation of Surfaces in Tomographic Data

The radon transform is invertable (albeit, marginally so) when the measured data consists of a sufficient number of good quality, properly spaced projections [73]. However, for many applications the number of realizable projections is insufficient, and direct greyscale reconstructions are susceptible to artifacts. We will refer to such problems as *under-constrained* tomographic problems. Cases of under-constrained tomographic problems usually fall into one of two classes. The first class is where the measuring device produces a relatively dense set of projections (i.e. adequately spaced) that do not span a full 180 degrees. In these cases, the sinogram contains regions without measured data. Considering the radon transform in the Fourier domain, these missing regions of the sinogram correspond to a transform with angular wedges (pie slices) that are null, making the transform noninvertable. We assume that these missing regions are large enough to preclude any straight-forward interpolation in the frequency domain. The second class of incomplete tomographic problems are those that consist of an insufficient number of widely spaced projections. We assume that these sparse samples of the sinogram space are well distributed over a wide range of angles. For this discussion the precise spacing is not important. This problem is characterized by very little data in the Fourier domain, and direct inversion approaches produce severe artifacts. Difficulties in reconstructing volumes from such incomplete tomographic datasets are often aggravated by noise in the measurements and misalignments among projections.

Under-constrained problems are typically solved using one or both of two different strategies. The first strategy is to choose from among feasible solutions (those that match the data) by imposing some additional criterion, such as finding the solution that minimizes an energy function. This additional criterion should be designed to capture some desirable property, such as minimum entropy. The second strategy is to parameterize the solution in a way that reduces the number of degrees of freedom. Normally, the model should contain few enough parameters so that the resulting parameter estimation problem is over constrained. In such situations solutions are allowed to differ from the data in a way that accounts for noise in the measurements.

In this section we consider a special class of under constrained tomographic

problems that permits the use of a simplifying model. The class of problems we consider are those in which the imaging process is targeted toward tissues or organs that have been set apart from the other anatomy by some contrast agent. This agent could be an opaque dye, as in the case of transmission tomography, or an emissive metobolite, as in nuclear medicine. We assume that this agent produces relatively homogeneous patches that are bounded by areas of high contrast. This assumption is reasonable, for instance, in subtractive angiography or CT studies of the colon. The proposed approach, therefore, seeks to find the boundaries of different regions in a volume by estimating sets of closed surface models and their associated density parameters directly from the incomplete sinogram data [74]. Thus, the reconstruction problem is converted to a segmentation problem. Of course, we can never expect real tissues to exhibit truly homogeneous densities. However, we assert that when inhomogeneities are somewhat uncorrelated and of low contrast the proposed model is adequate to obtain acceptable reconstructions.

0.6.1 Related Work

Several areas of distinct areas of research in medical imaging, computer vision, and inverse problems impact this work. Numerous tomographic reconstruction methods are described in the literature [75, 76], and the method of choice depends on the quality of projection data. Filtered back projection (FBP), the most widely used approach, works well in the case the fully constrained reconstruction where one is given *enough high-quality projections over 180 degree angular range*. Statistical, iterative approaches such as maximum likelihood (ML) and maximum a posteriori (MAP) estimation have been proven to work well with *noisy* projection data, but do not systematically address the under constrained reconstruction problem and generally rely on complete datasets. An exception is [77], which proposes an iterative algebraic approach that includes some assumptions about the homogeneity of the solution to compute a full greyscale reconstruction. Also, some hybrid approaches [78, 79] are specifically developed to deal with *limited-angle* tomography by extrapolating the missing sinogram data.

Other tomographics reconstruction techniques have been proposed, for example those that utilize discrete tomagraphy strategies [80, 73, 81, 82], and deformable models [83, 84, 85, 86, 87]. The literature also describes many

examples of level sets as curve and surface models for image segmentation [7, 6, 41, 88]. The authors have examined their usefulness for 3D segmentation of TEM *reconstructions* [37]. Several authors have proposed solving *inverse problems* using level sets [89, 90, 91, 92, 93, 94, 95], but are mostly limited to solving 2D problems.

We make several important contributions to this previous body of work; first we give a formal derivation of the motion of deformable surface models as the first variation of an error term that relates the projected model to the noisy tomographic data. This formulation does not assume any specific surface representation, and therefore applies to a wide range of tomographic, surface-fitting problems. Second we present a level set implementation of this formulation that computes incremental changes in the radon transform of the projected model only along the wave front, which makes it practical on large datasets. Third we examine the specific problem of initializing the deformable surface in the absence of complete sinogram data, and demonstrate, using real and synthetic data, the effectiveness of direct surface estimation for a specific class of tomographic problems which are under constrained.

0.6.2 Mathematical Formulation

As an introduction, we begin with the derivation of surface estimation problem in two dimensions. The goal is to simultaneously estimate the interface between two materials and their densities, β_0 and β_1. Thus we have a background, with density β_0 and collection of solid objects with density β_1. We denote the (open) set of points in those regions as Ω, the closure of that set, the surface, as $\mathcal{S}$.

The projection of a 2D signal $f(x, y)$ produces a sinogram given by the radon transform as

$$p(s, \theta) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x, y)\delta(R_\theta \boldsymbol{x} - s)d\boldsymbol{x} \ , \tag{33}$$

where $R_\theta \boldsymbol{x} = x\cos(\theta)+y\sin(\theta)$ is a rotation and projection of a point $\boldsymbol{x} = (x, y)$ onto the imaging plane associated with θ. The 3D formulation is the same, except that the signal $f(x, y, z)$ produces a collection of images. We denote the projection of the model, which includes estimates of the objects and the background, as $\widehat{p}(s, \theta)$. For this work we denote the angles associated with a discrete set of projections as $\theta_1, \ldots \theta_N$ and denote the domain of each projection as $S = s_1, \ldots s_M$. Our strategy is to find Ω, β_0, and β_1 by maximizing the

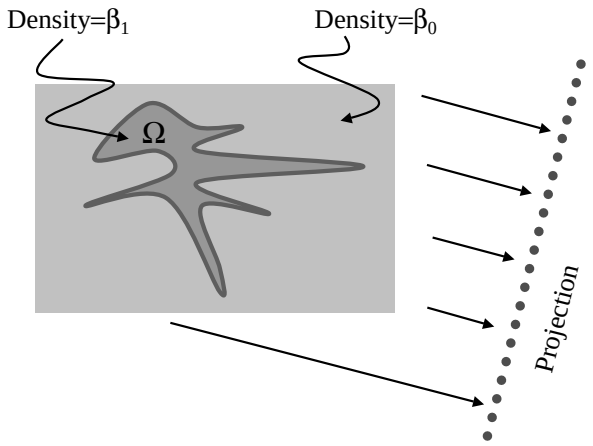


Figure 17: The model is the interface between two densities, which are projected onto the imaging plane to create $\widehat{p}(s, \theta_i)$.

likelihood.

If we assume the projection measurements are corrupted by independent noise, the log likelihood of a collection of measurements for a specific shape and density estimate is the probability of those measurements conditional on the model.

$$\ln P(p(s_1, \theta_1), p(s_2, \theta_1), \ldots, p(s_M, \theta_N) | \mathcal{S}, \beta_0, \beta_1) = \sum_i \sum_j \ln P(p(s_j, \theta_i) | \mathcal{S}, \beta_0, \beta_1). \quad (34)$$

We call the negative log likelihood the *error* and denote it E_{data}. Normally, the probability density of a measurement is parameterized by the ideal value, which gives

$$E_{\text{data}} = \sum_{i=1}^{N} \sum_{j=1}^{M} E\left(\widehat{p}_{ij}, p_{ij}\right), \quad (35)$$

where $E(\widehat{p}_{i,j}, p_{i,j}) = -\ln P(\widehat{p}_{i,j}, p_{i,j})$ is the error associated with a particular point in the radon space, and $p_{i,j} = p(s_j, \theta_i)$. In the case of independent Gaussian noise, E is a quadratic, and the log likelihood results in a weighted least squares in the radon space. For all of our results, we use a Gaussian noise model. Next we apply the object model, shown in Figure 0.6.2, to the reconstruction of f. If we let $g(x, y)$ be a binary inside-outside function on Ω,

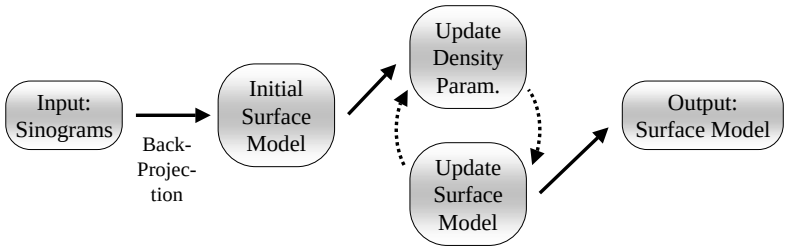


Figure 18: The reconstruction strategy starts with an initial surface estimate and iteratively modifies its shape and the associated density parameters to achieve a good fit to the input data.

then we have the following approximation to $f(x, y)$:

$$f(x, y) \approx \beta_0 + [\beta_1 - \beta_0] g(x, y). \tag{36}$$

Applying the radon transform to the model and substituting for $\widehat{p}$, gives

$$E_{\text{data}} = \sum_{i=1}^{N} \sum_{j=1}^{M} E\left(\beta_0 K(s_j, \theta_i) + [\beta_1 - \beta_0] \int_{\Omega} \delta(R_{\theta_i} \boldsymbol{x} - s_j) d\boldsymbol{x}, p_{ij} \right), \tag{37}$$

where $K(s_j, \theta_i)$ is the projection of the background— it depends on the geometry of the region over which the data is taken, and is independent of the surface estimate. For some applications we know that $\beta_0 = 0$, and the term $\beta_0 K$ is zero. The integral over Ω results from integrating g over the entire domain.

The proposed strategy is to alternately (i.e. separately) update the shape of the surface model and the density parameters. For the surface shape, a gradient descent minimization approach describes the deformation of the surface, with respect to an evolution parameter t, as it progressively improves its fit to the sinogram data. The incremental change in the likelihood is

$$\frac{dE_{\text{data}}}{dt} = \int_{\mathcal{S}} \sum_{i=1}^{N} \sum_{j=1}^{M} \frac{d}{dt} E\left(\widehat{p}_{ij}, p_{i,j}\right) \boldsymbol{d\boldsymbol{x}} = \int_{\mathcal{S}} \sum_{i=1}^{N} \sum_{j=1}^{M} E'\left(\widehat{p}_{ij}, p_{ij}\right) \frac{d\widehat{p}_{ij}}{dt} \boldsymbol{d\boldsymbol{x}}, \tag{38}$$

where $E' = \partial E / \partial \widehat{p}$, which, for Gaussian noise, is simply the difference between $\widehat{p}$ and p. Next we must formulate $d\widehat{p}/dt$, which, by the transport equation, is

$$\begin{aligned} \frac{d\widehat{p}_{ij}}{dt} &= [\beta_1 - \beta_0] \frac{d}{dt} \int_{\Omega} \delta(R_{\theta_i} \boldsymbol{x} - s_j) d\boldsymbol{x} \\ &= [\beta_1 - \beta_0] \int_{\mathcal{S}} \delta(R_{\theta_i} \boldsymbol{x} - s_j) \boldsymbol{n}(\boldsymbol{x}) \cdot \boldsymbol{v}(\boldsymbol{x}) d\boldsymbol{x}, \end{aligned} \tag{39}$$

where $\boldsymbol{n}$ is an outward pointing surface normal and $\boldsymbol{v}(\boldsymbol{x})$ is the velocity of the surface at the point $\boldsymbol{x}$. The derivative of E_{data} with respect to surface motion is therefore

$$\frac{dE_{\text{data}}}{dt} = [\beta_1 - \beta_0] \int_{\mathcal{S}} \sum_{i=1}^{N} \sum_{j=1}^{M} E'\left(\widehat{p}_{i,j}, p_{ij}\right) \delta(R_{\theta_i}\boldsymbol{x} - s_j)\boldsymbol{n}(\boldsymbol{x}) \cdot \boldsymbol{v}(\boldsymbol{x})\, d\boldsymbol{x}. \quad (40)$$

Note that the integral over $d\boldsymbol{x}$ and the δ functional serve merely to associate s_j in the ith scan with the appropriate $\boldsymbol{x}$ point. If the samples in each projection are sufficiently dense, we can approximate the sum over j as an integral over the image domain, and thus for every $\boldsymbol{x}$ on the surface there is a mapping back into the ith projection. We denote this point $s_i(\boldsymbol{x})$. This gives a closed-form expression for the derivative of the derivative of E_{data} in terms of the surface velocity.

$$\frac{dE_{\text{data}}}{dt} = [\beta_1 - \beta_0] \int_{\mathcal{S}} \sum_{i=1}^{N} e_i(\boldsymbol{x})\boldsymbol{n}(\boldsymbol{x}) \cdot \boldsymbol{v}(\boldsymbol{x}) d\boldsymbol{x}, \quad (41)$$

where $e_i(\boldsymbol{x}) = E'(\widehat{p}(s_i(\boldsymbol{x}), \theta_i), p(s_i(\boldsymbol{x}), \theta_i))$ is the derivative of the error associated with the point $s_i(\boldsymbol{x})$ in the ith projection. The result shown in (41) does not make any specific assumptions about the surface shape or its representation. Thus, this equation could be mapped onto any set of shape parameters by inserting the derivative of a surface point with respect to those parameters. Of course one would have to compute the surface integral, and methods for solving such equations on parametric models (in the context of range data) are described in [96].

For this work we are interested in *free-form* deformations, where each point on the surface can move independently from the rest. If we let $\boldsymbol{x}_t$ represent the velocity of a point on the surface, the gradient descent surface free-form surface motion is

$$\boldsymbol{x}_t = -\frac{dE_{\text{data}}}{d\boldsymbol{x}} = (\beta_0 - \beta_1) \sum_{i=1}^{N} e_i(\boldsymbol{x})\boldsymbol{n}(\boldsymbol{x}). \quad (42)$$

Thus, at a point $\boldsymbol{x} \in \mathcal{S}$, the ith projection has the effect of causing the surface to expand or contract according to the difference between the projected model values and the measured data at the point $s_i(\boldsymbol{x})$, the projection of $\boldsymbol{x}$ (Figure 0.6.2). The surface motion prescribed by a collection of projections is the sum of motions from the individual projections. In the case of continuous set of angles, the surface motion at a point is proportional to the sinusoidal line integral on the *error sinogram*, which is $e(s, \theta)$.

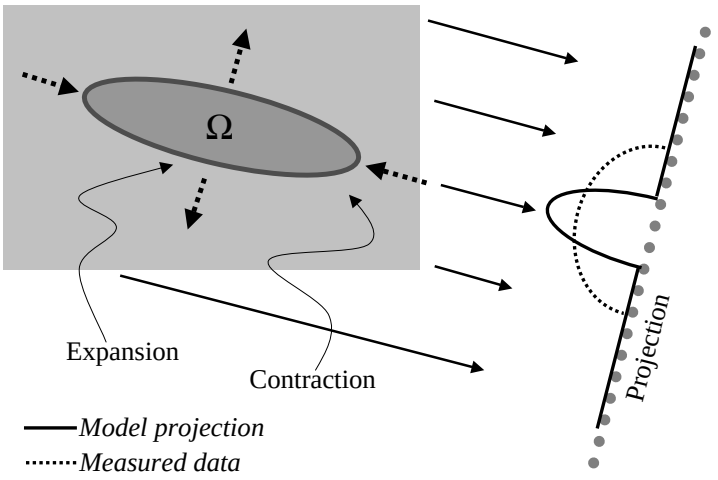


Figure 19: The model expands or contracts based on the difference in the sinograms between the projected model and the measured data.

0.6.2.1 Density Parameter Estimation

The density parameters also affect the error term in equation (37). We propose to update the estimate of the surface model iteratively, and at each iteration we re-estimate the quantities β_0 and β_1 in such a way that the energy, E_{data} is minimized. Treating Ω as fixed, (37) has two unknowns, β_0 and β_1, which are computed from the following system:

$$\frac{\partial E_{\text{data}}}{\partial \beta_0} = 0, \quad \frac{\partial E_{\text{data}}}{\partial \beta_1} = 0. \tag{43}$$

In the case of a Gaussian noise model (43) is a linear system. Because of variations in instrumentation, the contrast levels of images taken at different angles can vary. In such cases we estimate sets of such parameters, i.e., $\beta_0(\theta_i)$ and $\beta_1(\theta_i)$ for $i = 1 \ldots N$.

To extend the domain to higher dimensions, we have $\boldsymbol{x} \in \mathbb{R}^n$, and $S \subset \mathbb{R}^{n-1}$ and the mapping $\boldsymbol{s}_i : \mathbb{R}^n \mapsto S$ models the projective geometry of the imaging system (e.g. orthographic, cone beam, or fan beam). Otherwise the formulation is the same as 2D.

One important consideration is to model more complex models of density. If β_0 and β_1 are smooth, scalar functions defined over the space in which the

surface model deforms, and g is a binary function, the density model is:

$$f(\boldsymbol{x}) = \beta_0(\boldsymbol{x}) + (\beta_1(\boldsymbol{x}) - \beta_0(\boldsymbol{x}))\, g(x, y). \tag{44}$$

The first variation of the boundary is simply

$$\frac{d\boldsymbol{x}}{dt} = [\beta_1(\boldsymbol{x}) - \beta_0(\boldsymbol{x})] \sum_{i=1}^{N} e_i(\boldsymbol{x})\boldsymbol{n}(\boldsymbol{x}). \tag{45}$$

Note, this formulation is different from that of Yu and Fessler [95], who address the problem of reconstruction from noisy tomographic using using a single density function f with a smoothing term that interacts with a set of deformable edge models Γ. The edges models are surfaces, represented using level sets. In that case variational framework for deforming Γ requires differentiation of f across the edge, precisely where the proposed model exhibits (intentionally) a discontinuity.

0.6.2.2 Prior

The analysis above maximizes the likelihood. For a full MAP estimation, we include a prior term. Because we are working with the logarithm of the likelihood, the effect of the prior is additive:

$$\boldsymbol{x}_t = -\frac{dE_{\text{data}}}{d\boldsymbol{x}} - \frac{dE_{\text{prior}}}{d\boldsymbol{x}}. \tag{46}$$

Thus in addition to the noise model, we can incorporate some knowledge about the kinds of shapes that give rise to the measurements. With appropriately fashioned priors, we can push the solution toward desirable shapes or density values, or penalize certain shape properties, such as roughness or complexity. The choice of prior is intimately tied to the choice of surface representation and the specific application, but is independent of the formulation that describes the relationship between the estimate and the data, given in (37).

Because the data are noisy and incomplete it is useful to introduce a simple, low-level prior on the surface estimate. We therefore use a prior that penalizes surface area, which introduces a second-order smoothing term in the surface motion. That term introduces a free parameter C, which controls the relative influence of the smoothing term. The general question of how best to smooth surfaces remains an important, open question. However, if we restrict ourselves to curvature-based geometric flows, there are several reasonable options in the

literature [7, 31, 97]. The following subsection, which describes the surface representation used for our application, gives a more precise description of our smoothing methods.

0.6.3 Surface Representation and Prior

Our goal is to build an algorithm that applies to a wide range of potentially complicated shapes with arbitrary topologies—topologies that could change as the shapes deform to fit the data. For this reason, we have implemented the free-form deformation given in (42) with an implicit level set representation.

Substituting the expression for $d\boldsymbol{x}/dt$ (from Equations 45 and 46) into the $\mathrm{d}\boldsymbol{s}/\mathrm{d}t$ term of the level set equation (Equation 4a), and recalling that $\boldsymbol{n} = \nabla\phi/|\nabla\phi|$, gives

$$\frac{\partial\phi}{\partial t} = -|\nabla\phi| \left(\sum_{i=1}^{M} e_i(\boldsymbol{x}) + C\kappa(\boldsymbol{x}) \right), \tag{47}$$

where κ represents the effect of the prior, which is assumed to be in the normal direction.

The prior is introduced as a curvature-based smoothing on the level set surfaces. Thus, every level set moves according to a weighted combination of the principle curvatures, k_1 and k_2, at each point. This point-wise motion is in the direction of the surface normal. For instance, the mean curvature, widely used for surface smoothing, is $H = (k_1 + k_2)/2$. Several authors have proposed using Gaussian curvature $K = k_1 k_2$ or functions thereof [97]. Recently [98] have proposed using the minimum curvature, $M = \mathrm{AbsMin}(k_1, k_2)$ for preserving thin, tubular structures, which otherwise have a tendency to *pinch off* under mean curvature smoothing.

In previous work [41], the authors have proposed a weighted sum of mean curvatures that emphasizes the minimum curvature, but incorporates a smooth transition between different surface regions, avoiding the discontinuities (in the derivative of motion) associated with a strict minimum. The *weighted curvature* is

$$W = \frac{k_1^2}{k_1^2 + k_2^2} k_2 + \frac{k_2^2}{k_1^2 + k_2^2} k_1 = \frac{2HK}{D^2}, \tag{48}$$

where $D = \sqrt{k_1^2 + k_2^2}$ is the deviation from flatness [99].

For an implicit surface, the shape matrix [100] is the derivative of the normal map projected onto the tangent plane of the surface. If we let the normal map

be $\boldsymbol{n} = \nabla\phi/|\nabla\phi|$, the derivative of this is the 3×3 matrix

$$N = \left(\frac{\partial \boldsymbol{n}}{\partial x} \;\; \frac{\partial \boldsymbol{n}}{\partial y} \;\; \frac{\partial \boldsymbol{n}}{\partial z} \right)^T . \tag{49}$$

The projection of this derivative matrix onto the tangent plane gives the shape matrix $B = N(I - \boldsymbol{n} \otimes \boldsymbol{n})$, where $\otimes$ is the exterior product and I is the 3×3 identity matrix. The eigenvalues of the matrix B are k_1, k_2 and zero, and the eigenvectors are the principle directions and the normal, respectively. Because the third eigenvalue is zero, we can compute k_1, k_2 and various differential invariants directly from the invariants of B. Thus the weighted-curvature flow is computing from B using the identities $D = ||B||_2$, $H = \mathrm{Tr(B)}/2$, and $K = 2H^2 - D^2/2$. The choice of numerical methods for computing B is discussed in the following section.

0.6.4 Implementation

The level set equations are solved by finite differences on a discrete grid, i.e. a volume. This raises several important issues in the implementation. These issues are the choice of numerical approximations to the PDE, efficient and accurate schemes for representing the volume, and mechanisms for computing the sinogram-based deformation in (47).

0.6.4.1 Numerical Schemes

Osher and Sethian [30] have proposed an up-wind method for solving equations of the form $\phi_t = \nabla\phi \cdot \boldsymbol{v}$, of which $\phi_t = |\nabla\phi| \sum_i e_i(\boldsymbol{x})$, from (47), is an example. The up-wind scheme utilizes one-sided derivatives in the computation of $|\nabla\phi|$, where the direction of the derivative depends, point-by-point, on the sign of the speed term $\sum_i e_i(\boldsymbol{x})$. With strictly regulated time steps, these scheme avoids overshooting (ringing) and instability.

Under normal circumstances, the curvature term, which is a directional diffusion, does not suffer from overshooting; it can be computed directly from first- and second-order derivatives of ϕ using central difference schemes. However, we have found that central differences does introduce instabilities when computing flows that rely on quantities other than the mean curvature. Therefore we use the method of *differences of normals* [101, 102] in lieu of central differences. The strategy is to compute normalized gradients at staggered grid points and

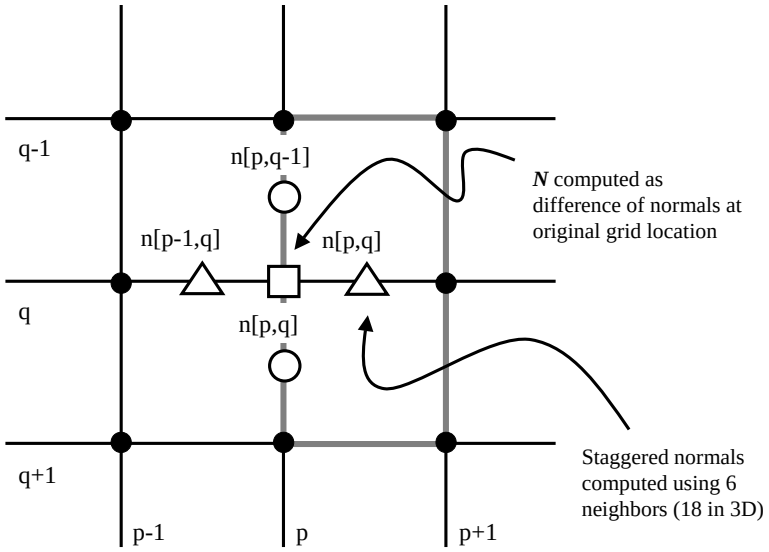


Figure 20: The shape matrix B is computed by using the differences of staggered normals.

take the difference of these staggered normals to get centrally located approximations to N (as in Figure 0.6.4.1). The normal projection operator $\boldsymbol{n} \otimes \boldsymbol{n}$ is computed with gradient estimates from central differences. The resulting curvatures are treated as speed terms (motion in the normal direction), and the associated gradient magnitude is computed using the up-wind scheme.

0.6.4.2 Sparse-Field Method

The computational burden associated with solving the 3D, second-order, nonlinear level set PDE is significant. For this reason several papers [34, 35] have proposed narrow-band methods, which compute solutions only a relatively small set of pixels in the vicinity of k level set. The authors [36] have proposed a sparse-field algorithm, which uses an approximation to the distance transform and makes it feasible to recompute the neighborhood of the level set model at each time step. It computes updates on a band of grid points, called the *active set*, that is one point wide. Several layers around this active set are updated in such a way to maintain a neighborhood in order to calculate derivatives. The position of the surface model is determined by the set of active

points and their values.

0.6.4.3 Incremental Projection Updates

The tomographic surface reconstruction problem entails an additional computational burden, because the measured data must be compared to the projected model at *each iteration*. Specifically, computing $\widehat{p}_{ij}$ can be a major bottleneck. Computing this term requires re-computing the sinogram of the surface/object model as it moves. In the worst case, we would re-project the entire model every iteration.

To address this computational concern, we have developed the method of *incremental projection updates* (IPU). Rather than fully recompute $\widehat{p}$ at every iteration, we maintain a current running version of $\widehat{p}$ and update it to reflect the changes in the model as it deforms. Changes in the model are computed only on a small set of grid points in the volume, and therefore the update time is proportional to the area of the surface, rather than the size of the volume it encloses.

The IPU strategy works with the the sparse-field algorithm as follows. At each iteration, the sparse-field algorithm updates only the active layer (one voxel wide) and modifies the set of active grid points as the surface moves. The incremental projection update strategy takes advantage of this to selectively update the model projection to reflect those changes. At each iteration, the amount of change in an active point's value determines the motion of that particular surface point as well as the percentage of the surrounding voxel that is either inside or outside of the surface. By the linearity of projection, we can map these changes in the object shape, computed at grid points along the surface boundary, back into the sinogram space and thereby *incrementally* update the sinogram. Notice that each 3D grid point has a weighting coefficient (these are precomputed and fixed), which is determined by its geometric mapping of the surrounding voxel back into the sinogram, as in Figure 21. In this way the IPU method maintains sub-voxel accuracy at a relatively low computational cost.

0.6.4.4 Initialization

The deformable model fitting approach requires an initial model, i.e. $\phi(\boldsymbol{x}, t = 0)$. This initial model should be obtained using the "best" information available

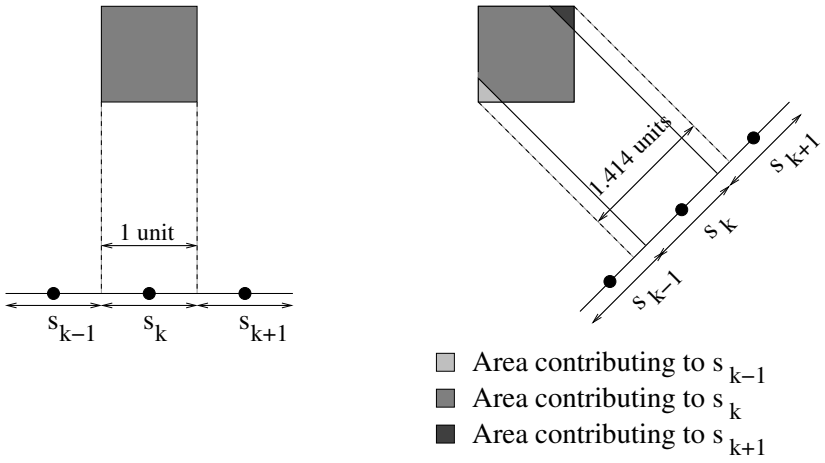


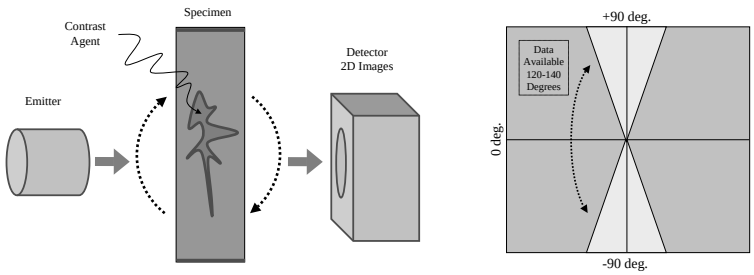
Figure 21: A weighting coefficient for each voxel determines the portions of the discrete sinogram influenced by incremental changes to a grid point.

prior to the surface fitting. In some cases this will mean thresholding a greyscale reconstruction, such as FBP, knowing that it has artifacts. In practice the initial surface estimate is impacted by the reconstruction method and the choice of threshold, and because we perform a local minimization, these choices can affect the final result. Fortunately, the proposed formulation is moderately robust with respect to the initial model, and our results show that the method works well under a range of reasonable initialization strategies.

0.6.5 Results

0.6.5.1 Transmission Electron Microscopy

Transmission electron microscopy is the process of using transmission images of electron beams to reveal biological structures on very small dimensions. Typically TEM datasets are produced using a dye that highlights regions of interest, e.g. the interior of a microscopic structure, such as a cell (see Figure 0.6.5.1a). There are technical limits to the projection angles from which data can be measured. These limits are due to the mechanical apparatus used to tilt the specimens and the trade off between the destructive effects of electron energy and the effective specimen thickness, which increases with tilt angle. Usually, the maximum tilt angle is restricted to about ±60–70 degrees. Figure 0.6.5.1b shows an illustration of the geometry of this limited-angle scenario. The TEM



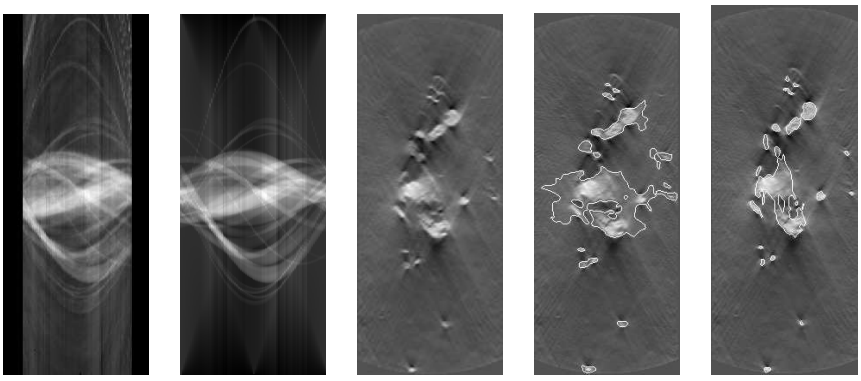
(a) (b)

Figure 22: a) Transmission electron microscopy is used to image very small specimens that have been set apart from the substrate by a contrast agent. b) TEM imaging technology provides projections over a limited set of angles.

reconstruction problem is further aggravated by the degree of electron scattering, which results in projection images (sinograms) that are noisy relative to many other modalities, e.g. X-ray CT. Finally, due to the flexible nature of biological objects and the imperfections in the tilting mechanism, the objects undergo some movements while being tilted. Manual alignment procedures used to account for this tend to produce small misregistration errors.

We applied the proposed algorithm to 3D TEM data obtained from a 3 MeV electron microscope. This 3D dataset consists of 67 tilt series images, each corresponding to one view of the projection. Each tilt series image is of size 424x334. The volume reconstructed by FBP is of size 424x424x334. Figure 23a and 23b show the sinogram corresponding to a single slice of this dataset and the estimate of the same sinogram created by the method. Figure 23e shows the surface estimate intersecting this slice overlaid on the back projected slice. Some structures not seen in the back projection are introduced in the final estimation, but the orientation of the structures introduced suggests that these are valid features that were lost due to reconstruction artifacts from the FBP. Also, the proposed method captures line-by-line brightness variations in the input sinogram (as explained in section 0.6.2.1). This suggests that the density estimation procedure is correct.

Figure 24 shows the 3D initialization and the final 3D surface estimate. The figure also shows enlarged initial and final versions of a small section of the surface. Computing the surface estimate for the TEM dendrite with 150 iterations



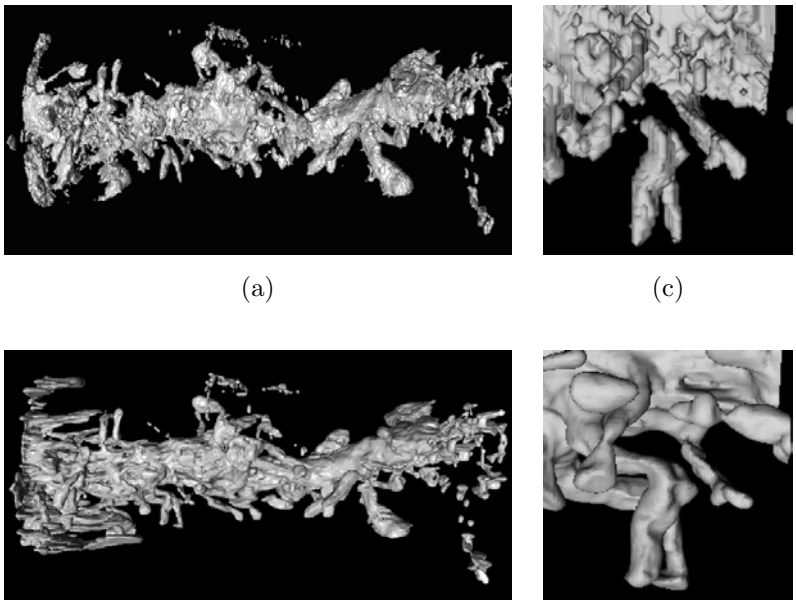
(a) (b) (c) (d) (e)

Figure 23: 2D slice of dendrite data: (a) Sinogram of one slice (b) Sinogram estimated by the proposed method (c) Back projection showing artifacts (d) Initial model obtained by thresholding the back projection (white curve overlaid on the back projection) (e) Final surface estimate.

took approximately 3 hours on a single 300MHz processor of a Silicon Graphics Onyx2 workstation. We consider these results positive for several reasons. First, the biology is such that one expects the spines (small protrusions) to be connected to the dendrite body. The proposed method clearly establishes those connections, based soley on consistency of the model with the projected data. The second piece of evidence is the shapes of the spines themselves. The reconstructed model shows the recurrence of a characteristic shape—a long thin spine with a cup-like structure on the end. This characteristic structure, which often fails to show up in the FBP reconstruction, does appear quite regularly in hand-segmentations of the same datasets.

0.6.5.2 Sinogram Extrapolation

The fitting of surfaces to this data is a simplification. It is justified in the context of segmentation, but there *are* underlying inhomogeneities in the density of this specimen, which could be indicative of relevant structures. Thus for some applications *direct* visualization of the measured data, by volume rendering, offers advantages over the segmented surfaces. We propose to use the surface estimation algorithm as a mechanism for estimating the missing data in the sinograms.



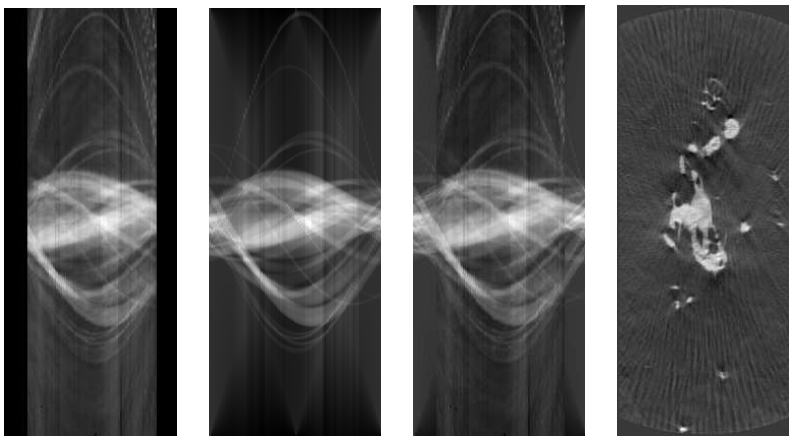
(a) (c)

(b) (d)

Figure 24: 3D results: (a) Surface initialization (b) Final surface estimated after 150 iterations (c) A portion of the initial surface enlarged (d) The corresponding portion in the final surface

Figure 25a and 25b show the input sinogram and the sinogram of the estimated model (for one slice) of the TEM dendrite data. The estimated sinogram demonstrates that the surface estimation method recovers the missing information in a reasonable way. Thus, we combine the sinograms from the model with original sinograms to produce a "full" sinogram that still contains *all* of the orginal, measured data. FBP reconstructions from such augmented sinograms should have fewer limited-angle streak artifacts.

We demonstrate this by comparing volume renderings with and without the augmentation. We create augmented sinograms by using sinogram data from the estimated model only where the data is missing from the measured sinograms. The augmented sinogram for a single slice is shown in Figure 25c. The slice reconstructed (FBP) from the augmented sinogram is shown in Figure 25d. Note that this reconstructed slice does not contain the limited-angle artifacts that appear in the slice in Figure 23c. Maximum intensity projection (MIP) volume renderings of the volume created from original sinograms and



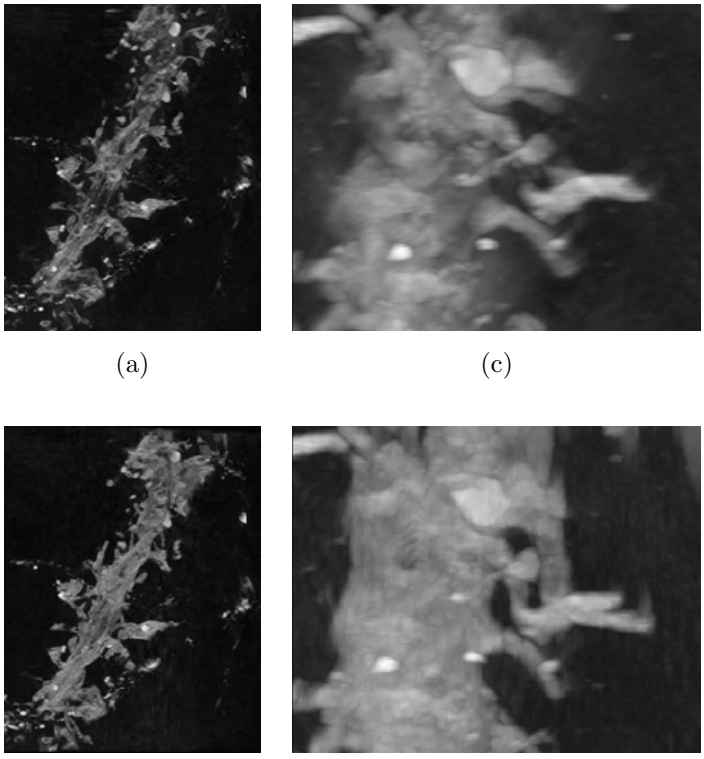
(a) (b) (c) (d)

Figure 25: Sinogram extrapolation for slice number 150 of dendrite data: (a) Input sinogram (b) Sinogram estimated by the proposed method (c) Augmented sinogram constructed using original data and estimating missing data from the segmentation (d) FBP reconstruction of the augmented sinogram.

the volume created from augmented sinograms are compared in Figure 26. The main body of the dendrite, which exhibited a very convoluted and fuzzy boundary, shows better definition. Also, several of the spines which were dangling in the original reconstruction are now connected.

Conclusions

This chapter has described a level set segmentation framework and the preprocessing and data analysis techniques needed for a number of segmentation applications. Several standard volume processing algorithms have been incorporated into the framework in order to segment datasets generated from MRI, CT and TEM scans. A technique based on moving least-squares has been developed for segmenting multiple non-uniform scans of a single object. New scalar measures have been defined for extracting structures from diffusion tensor MRI scans. Finally, a direct approach to the segmentation of incomplete tomographic data using density parameter estimation is described. These techniques, combined with level set surface deformations, allow us to segment many different types of biological volume datasets.



(a) (c)

(b) (d)

Figure 26: Sinogram extrapolation results: (a) MIP volume rendering of volume reconstructed from original sinograms (b) MIP volume rendering of volume reconstructed from augmented (extrapolated) sinograms (c) A portion of original MIP enlarged (d) The corresponding portion in augmented MIP enlarged

Acknowledgements

Several people provided valuable technical assistance and support to our work. They are Dr. Alan Barr, Dr. Jason Wood, Dr. John Wood, Dr. Cyrus Papan, Dr. Russ Jacobs, Dr. Scott Fraser, Dr. J. Michael Tyszka, Dr. Miriam Scadeng, Dr. David Dubowitz, Dr. Eric Ahrens, Dr. Mark Ellisman, Dr. Maryanne Martone, Dr. Chris Johnson and Dr. Mark Bastin. Datasets were provided by Caltech Biological Imaging Center (e.g. Figure 8a), National Center for Microscopy and Imaging Research (e.g. Figure 7, funded by NIH grant #P41-RR04050), Caltech Multi-Res Modeling Group (Figure 9 (Top)), Stanford Computer Graphics Laboratory (Figure 9 (Top)), Childrens Hospital - Los Angeles (Figure 10), University of Utah's SCI Institute (e.g. Figure 14), and the University of Edinburgh, UK (Figure 16).

This work was supported by National Science Foundation grants #ASC-89-20219, #ACI-9982273, #ACI-0083287 and #ACI-0089915, the Office of Naval Research Volume Visualization grant #N00014-97-0227, the National Institute on Drug Abuse and the National Institute of Mental Health, as part of the Human Brain Project, the National Library of Medicine "Insight" Project #N01-LM-0-3503, and the Caltech SURF Program.

Bibliography

[1] Drebin, R., Carpenter, L., and Hanrahan, P., Volume Rendering, In: Proceedings SIGGRAPH 88 Conference, pp. 65–74, 1988.

[2] Levoy, M., Display of Surfaces from Volume Data, IEEE Computer Graphics and Applications, Vol. 9, no. 3, pp. 245–261, 1990.

[3] Laur, D. and Hanrahan, P., Hierarchical splatting: A progressive refinement algorithm for volume rendering, In: SIGGRAPH '91 Proceedings, Sederberg, T. W., ed., pp. 285–288, July 1991.

[4] Parker, S., Parker, M., Livnat, Y., Sloan, P., Hansen, C., and Shirley, P., Interactive Ray Tracing for Volume Visualization, IEEE Transactions on Visualization and Computer Graphics, Vol. 5, no. 3, pp. 238–250, 1999.

[5] Leventon, M., Faugeraus, O., Grimson, W., and W. Wells III, Level Set Based Segmentation with Intensity and Curvature Priors, In: Workshop on Mathematical Methods in Biomedical Image Analysis Proceedings, pp. 4–11, June 2000.

[6] Malladi, R., Sethian, J., and Vemuri, B., Shape Modeling with Front Propagation: A Level Set Approach, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 17, no. 2, pp. 158–175, 1995.

[7] Sethian, J., Level Set Methods and Fast Marching Methods, Cambridge University Press, Cambridge, UK, second ed., 1999.

[8] Staib, L., Zeng, X., Schultz, R., and Duncan, J., Shape Constraints in Deformable Models, In: Handbook of Medical Imaging, Bankman, I., ed., Chapter 9, pp. 147–157, Academic Press, 2000.

[9] Wu, Z., Chung, H.-W., and Wehrli, F. W., A Bayesian Approach to Subvoxel Tissue Classification in NMR Microscopic Images of Trabecular Bone, Journal of Computer Assisted Tomography, Vol. 12, no. 1, pp. 1–9, 1988.

[10] Kao, Y.-H., Sorenson, J. A., and Winkler, S. S., MR Image Segmentation Using Vector Decompsition and Probability Techniques: A General Model and Its Application to Dual-Echo images, Magnetic Resonance in Medicine, Vol. 35, pp. 114–125, 1996.

[11] Cline, H. E., Lorensen, W. E., Kikinis, R., and Jolesz, F., Three-Dimensional Segmentation of MR Images of the Head Using Probability and Connectivity, Journal of Computer Assisted Tomography, Vol. 14, no. 6, pp. 1037–1045, Nov., Dec. 1990.

[12] Laidlaw, D. H., Fleischer, K. W., and Barr, A. H., Partial-Volume Bayesian Classification of Material Mixtures in MR Volume Data using Voxel Histograms, IEEE Transactions on Medical Imaging, Vol. 17, no. 1, pp. 74–86, feb 1998.

[13] Johnson, V. E., A Framework for Incorporating Structural Prior Information into the Estimation of Medical Images, In: Information Processing in Medical Imaging (IPMI'93), Barrett, H. H. and Gmitro, A. F., eds., no. 687 In Lecture Notes in Computer Science, pp. 307–321, Springer-Verlag, 1993.

[14] Marr, D. and Hildreth, E., Theory of Edge Detection, Proceedings of the Royal Society of London, Vol. B, no. 207, pp. 187–217, 1980.

[15] Marr, D., Vision, Freeman, San Francisco, 1982.

[16] Canny, J., A computational approach to edge detection, IEEE Trans. on Pat. Anal. and Mach. Intel., Vol. 8, no. 6, pp. 679–698, 1986.

[17] Cootes, T., Hill, A., Taylor, C., and Haslam, J., The Use of Active Shape Models For Locating Structures in Medical Images, In: Information Processing in Medical Imaging (IPMI'93), Barrett, H. H. and Gmitro, A. F., eds., no. 687 In Lecture Notes in Computer Science, pp. 33–47, Springer-Verlag, 1993.

[18] Stetten, G. and Pizer, S., Medial Node Models to Identify and Measure Objects in Real-Time 3D Echocardiography, IEEE Transactions on Medical Imaging, Vol. 18, no. 10, pp. 1025–1034, 1999.

[19] Wood, Z., M.Desbrun, Schröder, P., and Breen, D., Semi-Regular Mesh Extraction from Volumes, In: Proceedings of Visualization 2000, pp. 275–282, 2000.

[20] Miller, J., Breen, D., Lorensen, W., O'Bara, R., and Wozny, M., Geometrically Deformed Models: A Method for Extracting Closed Geometric Models from Volume Data, In: SIGGRAPH '91 Proceedings, pp. 217–226, July 1991.

[21] Pentland, A. P., Perceptual organization and the representation of natural form, Artificial Intelligence, Vol. 28, pp. 293–331, 1986.

[22] Terzopoulos, D. and Metaxas, D., Dynamic 3D models with local and global deformations: Deformable superquadrics, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 13, no. 7, pp. 703–714, 1991.

[23] Gupta, A. and Bajcsy, R., Volumetric Segmentation of Range Images of 3D Objects Using Superquadric Models, CVGIP: Image Understanding, Vol. 58, no. 3, pp. 302–326, 1993.

[24] Muraki, S., Volumetric shape description of range data using "Blobby Model", In: SIGGRAPH '91 Proceedings, Sederberg, T. W., ed., pp. 227–235, Jul. 1991.

[25] Szeliski, R., Tonnesen, D., and Terzopoulos, D., Modeling Surfaces of Arbitrary Topology with Dynamic Particles, In: Proc. Fourth Int. Conf. on Comp. Vision (ICCV'93), pp. 82–87, IEEE Computer Society Press, Berlin, Germany, May 1993.

[26] McInerney, T. and Terzopoulos, D., A Dynamic Finite Element Surface Model for Segmentation and Tracking in Multidimensional Medical Images with Application to Cardiac 4D Image Analysis, Computerized Medical Imaging and Graphics, Vol. 19, no. 1, pp. 69–83, 1995.

[27] Park, J., Metaxas, D., Young, A. A., and Axel, L., Deformable Models with Parameter Functions for Cardiac Motion Analysis from Tagged MRI Data, IEEE Transactions on Medical Imaging, Vol. 15, no. 3, pp. 278–289, June 1996.

[28] DeCarlo, D. and Metaxas, D., Shape Evolution with Structural and Topological Changes using Blending, IEEE Trans. on Pat. Anal. and Mach. Intel., Vol. 20, no. 11, pp. 1186–1205, November 1998.

[29] Ramamoorthi, R. and Arvo, J., Creating Generative Models from Range Images, In: SIGGRAPH '99 Proceedings, pp. 195–204, August 1999.

[30] Osher, S. and Sethian, J., Fronts Propagating with Curvature-Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations, Journal of Computational Physics, Vol. 79, pp. 12–49, 1988.

[31] Osher, S. and Fedkiw, R., Level Set Methods and Dynamic Implicit Surfaces, Springer, Berlin, 2002.

[32] Sethian, J., A Fast Marching Level Set Method for Monotonically Advancing Fronts, In: Proceedings of the National Academy of Science, Vol. 93 of 4, pp. 1591–1595, 1996.

[33] Tsitsiklis, J., Efficient Algorithms for Globally Optimal Trajectories, IEEE Trans. on Automatic Control, Vol. 40, no. 9, pp. 1528–1538, 1995.

[34] Adalsteinsson, D. and Sethian, J. A., A Fast Level Set Method for Propogating Interfaces, Journal of Computational Physics, Vol. 118, no. 2, pp. 269–277, may 1995.

[35] Peng, D., Merriman, B., Osher, S., Zhao, H.-K., and Kang, M., A PDE-Based Fast Local Level Set Method, Journal of Computational Physics, Vol. 155, pp. 410–438, 1999.

[36] Whitaker, R., A Level-Set Approach to 3D Reconstruction From Range Data, International Journal of Computer Vision, Vol. 29, no. 3, pp. 203–231, 1998.

[37] Whitaker, R., Breen, D., Museth, K., and Soni, N., Segmentation of Biological Datasets Using a Level-Set Framework, In: Volume Graphics 2001, Chen, M. and Kaufman, A., eds., pp. 249–263, Springer, Vienna, 2001.

[38] van den Boomgaard, R. and Smeulders, A. W. M., The morphological structure of images, the differential equations of morphological scale-space, IEEE Trans. on Pat. Anal. and Mach. Intel., Vol. 16, no. 11, pp. 1101–1113, 1994.

[39] Maragos, P., Differential Morphology and Image Processing, IEEE Trans. on Image Processing, Vol. 5, no. 6, pp. 922–937, June 1996.

[40] Requicha, A. and Voelcker, H., Boolean Operations in Solid Modeling: Boundary Evaluation and Merging Algorithms, Proceedings of the IEEE, Vol. 73, no. 1, pp. 30–44, 1985.

[41] Whitaker, R. T., Volumetric Deformable Models: Active Blobs, In: Visualization In Biomedical Computing, Robb, R. A., ed., pp. 122–134, SPIE, Mayo Clinic, Rochester, Minnesota, 1994.

[42] Sapiro, G., Geometric Partial Differential Equations and Image Analysis, Cambridge University Press, Cambridge, UK, 2001.

[43] Museth, K., Breen, D., Zhukov, L., and Whitaker, R., Level Set Segmentation from Multiple Non-Uniform Volume Datasets, In: Proc. IEEE Visualization Conference, pp. 179–186, October 2002.

[44] Shepard, D., A two-dimensional interpolation function for irregularly spaced points, In: Proc. ACM Nat. Conf., pp. 517–524, 1968.

[45] Lancaster, P. and Salkauskas, K., Surfaces generated by moving least squares methods, Math. Comp., Vol. 37, pp. 141–159, 1981.

[46] Farwig, R., Multivariate interpolation of arbitrarily spaced data by moving least-squares methods, J. of Comp. and App. Math., Vol. 16, pp. 79–93, 1986.

[47] Zhao, H.-K., Osher, S., and Fedkiw, R., Fast Surface Reconstruction using the Level Set Method, In: Proc. 1st IEEE Workshop on Variational and Level Set Methods, pp. 194–202, 2001.

[48] Turk, G. and Levoy, M., Zippered Polygon Meshes from Range Images, In: Proc. of SIGGRAPH '94, pp. 311–318, ACM SIGGRAPH, August 1994.

[49] Curless, B. and Levoy, M., A Volumetric Method for Building Complex Models from Range Images, In: Proc. SIGGRAPH '96, pp. 303–312, 1996.

[50] Tamez-Pena, J., Totterman, S., and Parker, K., MRI Isotropic Resolution Reconstruction from Two Orthogonal Scans, In: Proceedings SPIE Medical Imaging, Vol. 4322, pp. 87–97, 2001.

[51] Goshtasby, A. and Turner, D. A., Fusion of Short-Axis and Long-Axis Cardiac MR Images, In: IEEE Workshop on Mathematical Methods in Biomedical Image Analysis, pp. 202–211, San Francisco, June 1996.

[52] Brejl, M. and Sonka, M., Directional 3D Edge Detection in Anisotropic Data: Detector Design and Performance Assessment, Computer Vision and Image Understanding, Vol. 77, pp. 84–110, 2000.

[53] Haralick, R. M. and Shapiro, L. G., Computer and Robot Vision, Addison-Wesley, 1991.

[54] Press, W., Flannery, B., Teukolsky, S., and Vetterling, W., Numerical Recipes in C (2nd edition), Cambridge University Press, New York, NY, 1992.

[55] Basser, P. J., Mattielo, J., and Bihan, D. L., Estimation of the effective self-diffusion tensor from the NMR spin echo, Journal of Magnetic Resonance, Series B, Vol. 103, no. 3, pp. 247–254, 1994.

[56] Basser, P. J., Mattielo, J., and Bihan, D. L., MR diffusion tensor spectroscopy and imaging, Biophysical Journal, Vol. 66, no. 1, pp. 259–267, 1994.

[57] Basser, P. J. and Pierpaoli, C., Microstructural and Physiological Features of Tissues Elucidated by Quantitative-Diffusion-Tensor MRI, Journal of Magnetic Resonance, Series B, Vol. 111, no. 3, pp. 209–219, 1996.

[58] Westin, C.-F., Peled, S., Gudbjartsson, H., Kikinis, R., and Jolesz, F. A., Geometrical Diffusion Measures for MRI from Tensor Basis Analysis, In: Proceedings ISMRM 5th Annual Meeting, p. 1742, 1997.

[59] Peled, S., Gudbjartsson, H., Westin, C., Kikinis, R., and Jolesz, F., Magnetic Resonance Imaging Shows Orientation and Assymetry in White Matter Fiber Tracts, Brain Research, Vol. 780, pp. 27–33, 1998.

[60] Basser, P. and Pajevic, S., Statistical Artifacts in Diffusion Tensor MRI Caused by Background Noise, Magnetic Resonace in Medicine, Vol. 44, pp. 41–50, 2000.

[61] Ulug, A. and van Zijl, P., Orientation-independent diffusion imaging without tensor diagonalization: anisotropy definitions based on physical attributes of the diffusion ellipsoid, Journal of Magnetic Resonace Imaging, Vol. 9, pp. 804–813, 1999.

[62] Laidlaw, D., Ahrens, E., Kremers, D., Avalos, M., Jacobs, R., and Readhead, C., Visualizing Diffusion Tensor Images of the Mouse Spinal Cord, In: Proceedings IEEE Visualization '98, pp. 127–134, 1998.

[63] Kindlmann, G. and Weinstein, D., Hue-Balls and Lit-Tensors for Direct Volume Rendering of Diffusion Tensor Fields, In: Proceedings IEEE Visualization '99, pp. 183–189, 1999.

[64] Zhukov, L., , Museth, K., Breen, D., Whitaker, R., and Barr, A., Level Set Modeling and Segmentation of DT-MRI Brain Data, Journal of Electronic Imaging, Vol. 12, no. 1, pp. 125–133, January 2003.

[65] Basser, P., Pajevic, S., Pierpaoli, C., Duda, J., and Aldroubi, A., In Vivo Fiber Tractography Using DT-MRI Data, Magnetic Resonace in Medicine, Vol. 44, pp. 625–632, 2000.

[66] Poupon, C., Clark, C., Frouin, V., Regis, J., Bloch, I., Bihan, D. L., and Mangin, J.-F., Regularization of Diffusion-Based Direction Maps for the Tracking of Brain White Matter Fascicles, Neuroimage, Vol. 12, pp. 184–195, 2000.

[67] Singh, A., Goldgof, D., and Terzopoulos, D., eds., Deformable Models in Medical Image Analysis, IEEE Computer Society Press, Los Alamitos, CA, 1998.

[68] Kindlmann, G. and Durkin, J., Semi-Automatic Generation of Transfer Functions for Direct Volume Rendering, In: Proceedings IEEE Symposium on Volume Visualization, pp. 79–86, 1998.

[69] Zhukov, L., Weinstein, D., and Johnson, C., Independent Component Analysis for EEG Source Localization in Realistic Head Model, IEEE Engineering in Medicine and Biology, Vol. 19, pp. 87–96, 2000.

[70] Gibson, S. et al., Volumetric Object Modeling for Surgical Simulation, Medical Image Analysis, Vol. 2, no. 2, pp. 121–132, 1998.

[71] Bailey, M., Manufacturing Isovolumes, In: Volume Graphics, Chen, M., Kaufman, A., and Yagel, R., eds., pp. 79–83, Springer Verlag, London, 2000.

[72] Lorensen, W. and Cline, H., Marching Cubes: A High Resolution 3D Surface Construction Algorithm, In: Proc. SIGGRAPH '87, pp. 163–169, Jul. 1987.

[73] Ramm, A. G. and Katsevich, A. I., The Radon Transform and Local Tomography, CRC Press, Inc., 2000, Corporate Blvd., N.W., Boca Raton, Florida 33431, 1996.

[74] From Sinograms to Surfaces: A Direct Approach to the Segmentation of Tomographic Data, Vol. 2208 of Lecture Notes in Computer Science, Springer, 2001.

[75] Herman, G. T., Image Reconstruction From Projections, The Fundamentals of Computerized Tomography, Academic Press, New York, 1980.

[76] Roerdink, J. B. T. M., Computerized tomography and its applications: a guided tour, Nieuw Archief voor Wiskunde, Vol. 10, no. 3, pp. 277–308, nov 1992.

[77] Wang, G., Vannier, M., and Cheng, P., Iterative X-ray cone-beam tomography for metal artifact reduction and local region reconstruction, Microscopy and Microanalysis, Vol. 5, pp. 58–65, 1999.

[78] Inouye, T., Image Reconstruction with Limited Angle Projection Data, IEEE Transactions on Nuclear Science, Vol. NS-26, pp. 2666–2684, 1979.

[79] Prince, J. L. and Willsky, A. S., Hierarchical Reconstruction Using Geometry and Sinogram Restoration, IEEE Transactions on Image Processing, Vol. 2, no. 3, pp. 401–416, jul 1993.

[80] Herman, G. T. and Kuba, A., eds., Discrete Tomography: Foundations, Algorithms, and Applications, Birkhauser, Boston, 1999.

[81] Thirion, J. P., Segmentation of Tomographic Data without Image Reconstruction, IEEE Transactions on Medical Imaging, Vol. 11, pp. 102–110, mar 1992.

[82] Sullivan, S., Noble, A., and Ponce, J., On Reconstructing Curved Object Boundaries from Sets of X-Ray Images, In: Proceedings of the 1995 Conference on Computer Vision, Virtual Reality, and Robotics in Medicine, Ayache, N., ed., Lecture Notes in Computer Science 905, pp. 385–391, Springer-Verlag, 1995.

[83] Hanson, K., Cunningham, G., Jr., G. J., and Wolf, D., Tomographic reconstruction based on flexible geometric models, In: IEEE Int. Conf. on Image Processing (ICIP 94), pp. 145–147, 1994.

[84] Battle, X. L., Cunningham, G. S., and Hanson, K. M., 3D tomographic reconstruction using geometrical models, In: Medical Imaging: Image Processing, Hanson, K. M., ed., Vol. 3034, pp. 346–357, SPIE, 1997.

[85] Battle, X. L., Bizais, Y. J., Rest, C. L., and Turzo, A., Tomographic reconstruction using free-form deformation models, In: Medical Imaging: Image Processing, Hanson, K. M., ed., Vol. 3661, pp. 356–367, SPIE, 1999.

[86] Battle, X. L., LeRest, C., Turzo, A., and Bizais, Y., Three-Dimensional Attenuation Map Reconstruction Using Geometrical Models and Free-Form Deformations, IEEE Transactions on Medical Imaging, Vol. 19, no. 5, pp. 404–411, 2000.

[87] Mohammad-Djafari, A., Sauer, K., Khayi, Y., and Cano, E., Reconstruction of the shape of a compact object from a few number of projections, In: IEEE International Conference on Image Processing (ICIP), Vol. 1, pp. 165–169, 1997.

[88] Caselles, V., Kimmel, R., and Sapiro, G., Geodesic Active Contours, In: Fifth Int. Conf. on Comp. Vision, pp. 694–699, IEEE, IEEE Computer Society Press, 1995.

[89] Santosa, F., A level-set approach for inverse problems involving obstacles, European Series in Applied and Industrial Mathematics: Control Optimization and Calculus of Variations, Vol. 1, pp. 17–33, jan 1996.

[90] Dorn, O., Miller, E. L., and Rappaport, C., A shape reconstruction method for electromagnetic tomography using adjoint fields and level

sets, Inverse problems: Special issue on Electromagnetic Imaging and Inversion of the Earth's Subsurface (Invited Paper), Vol. 16, pp. 1119–1156, oct 2000.

[91] Dorn, O., Miller, E. L., and Rappaport, C., Shape reconstruction in 2D from limited-view multi-frequency electromagnetic data, Contemporary mathematics, to appear 2001.

[92] Chan, T. F. and Vese, L. A., A Level Set Algorithm for Minimizing the Mumford-Shah Functional in Image Processing, Tech. Rep. CAM 00-13, UCLA, Department of Mathematics, 2000.

[93] Tsai, A., Yezzi, A., and Willsky, A., A Curve Evolution Approach to Smoothing and Segmentation Using the Mumford-Shah Functional, In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Vol. 1, pp. 119–124, 2000.

[94] Debreuve, E., Barlaud, M., Aubert, G., and Darcourt, J., Attenuation Map Segmentation Without Reconstruction Using a Level Set Method in Nuclear Medicine Imaging, In: IEEE International Conference on Image Processing (ICIP), Vol. 1, pp. 34–38, 1998.

[95] Yu, D. and Fessler, J., Edge-preserving tomographic reconstruction with nonlocal regularization, In: Proceedings of IEEE Intl. Conf. on Image Processing, pp. 29–33, 1998.

[96] Whitaker, R. and Gregor, J., A Maximum Likelihood Surface Estimator For Dense Range Data, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 24, no. 10, pp. 1372–1387, October 2002.

[97] Sapiro, G., Geometric Partial Differential Equations and Image Analysis, Cambridge University Press, 2001.

[98] Lorigo, L., Faugeras, O., Grimson, E., Keriven, R., Kikinis, R., Nabavi, A., and Westin, C.-F., Co-Dimension 2 Geodesic Active Contours for the Segmentation of Tubular Strucures, In: Proceedings of IEEE Conf. on Comp. Vision and Pattern Recognition, pp. 444–452, June 2000.

[99] Koenderink, J. J., Solid Shape, MIT press, Cambridge, Mass, 1990.

[100] do Carmo, M., Differential Geometry of Curves and Surfaces, Prentice-Hall, Englewood Cliffs, 1976.

[101] Rudin, L., Osher, S., and Fatemi, C., Nonlinear total variation based noise removal algorithms, Physica D, Vol. 60, pp. 259–268, 1992.

[102] Whitaker, R. and Xue, X., Variable-Conductance, Level-Set Curvature for Image Denoising, In: Proc. IEEE International Conference on Image Processing, pp. 142–145, October 2001.

Physically Based Modeling and Animation of Fire

Duc Quang Nguyen
Stanford University
Industrial Light & Magic
dqnguyen@stanford.edu

Ronald Fedkiw
Stanford University
Industrial Light & Magic
fedkiw@cs.stanford.edu

Henrik Wann Jensen
Stanford University
henrik@graphics.stanford.edu

Abstract

We present a physically based method for modeling and animating fire. Our method is suitable for both smooth (laminar) and turbulent flames, and it can be used to animate the burning of either solid or gas fuels. We use the incompressible Navier-Stokes equations to independently model both vaporized fuel and hot gaseous products. We develop a physically based model for the expansion that takes place when a vaporized fuel reacts to form hot gaseous products, and a related model for the similar expansion that takes place when a solid fuel is vaporized into a gaseous state. The hot gaseous products, smoke and soot rise under the influence of buoyancy and are rendered using a blackbody radiation model. We also model and render the blue core that results from radicals in the chemical reaction zone where fuel is converted into products. Our method allows the fire and smoke to interact with objects, and flammable objects can catch on fire.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Ray Tracing;

Keywords: flames, fire, smoke, chemical reaction, blackbody radiation, implicit surface, incompressible flow, stable fluids, vorticity confinement

1 Introduction

The modeling of natural phenomena such as fire and flames remains a challenging problem in computer graphics. Simulations of fluid behavior are in demand for special effects depicting smoke, water, fire and other natural phenomena. Fire effects are especially in demand due to the dangerous nature of this phenomenon. Fire simulations are also of interest for virtual reality effects, for example to help train fire fighters or to determine proper placement of exit signs in smoke filled rooms (i.e. so they can be seen). The interested reader is referred to [Rushmeier 1994].

Combustion processes can be loosely classified into two rather distinct types of phenomena: detonations and deflagrations. In both of these processes, chemical reactions convert fuel into hot gaseous products. Deflagrations are low speed events such as the fire and flames we address in this paper, while detonations are high speed events such as explosions where shock waves and other compressible effects are important, see e.g. [Yngve et al. 2000] and [Neff and Fiume 1999]. As low speed events, deflagrations can be modeled using the equations for incompressible flow (as opposed to those for compressible flow). Furthermore, since viscous effects are small, we use the incompressible inviscid Euler equations similar to [Fedkiw et al. 2001]. As noted therein, these equations can be solved efficiently using a semi-Lagrangian stable fluid approach, see e.g. [Staniforth and Cote 1991] and [Stam 1999].

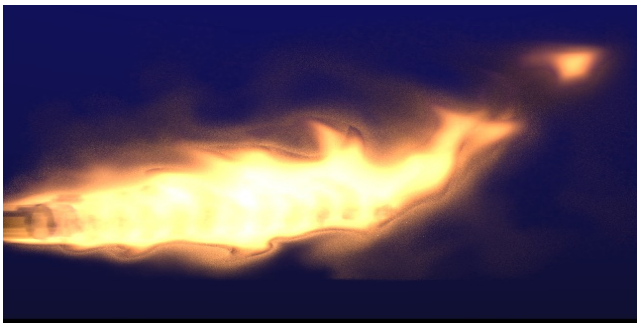


Figure 1: **A turbulent gas flame model of a flamethrower.**

An important, often neglected aspect of fire and flame modeling concerns the expansion of the fuel as it reacts to form hot gaseous products. This expansion is the reason for the visual fullness observed in many flames and is partly responsible for the visual turbulence as well. Since the incompressible equations do not account for expansion, we propose a simple thin flame model for capturing these effects. This is accomplished by using an implicit surface to represent the reaction zone where the gaseous fuel is converted into hot gaseous products. Although real reaction zones have a nonzero (but small) thickness, the thin flame approximation works well for visual modeling and has been used by scientists as well, see for example [Markstein 1964] who first proposed this methodology.

Our implementation of the thin flame model is as follows. First, a dynamic implicit surface is used to track the reaction zone where the gaseous fuel is converted into hot gaseous products. Then both the gaseous fuel and the hot gaseous products are separately modeled using independent sets of incompressible flow equations. Finally, these incompressible flow equations are updated together in a coupled fashion using the fact that both mass and momentum must be conserved as the gas reacts at the interface. While this gives rather pleasant looking laminar (smooth) flames, we include a vorticity confinement term, see [Steinhoff and Underhill 1994] and [Fedkiw et al. 2001], to model the larger scale turbulent flame structures that are difficult to capture on the relatively coarse grids used for efficiency reasons in computer graphics simulations. We also include other features important for visual simulation, such as the buoyancy effects generated by hot gases and the interaction of fire with flammable and nonflammable objects. We render the fire as a participating medium with blackbody radiation using a stochastic ray marching algorithm. In our rendering we pay careful attention

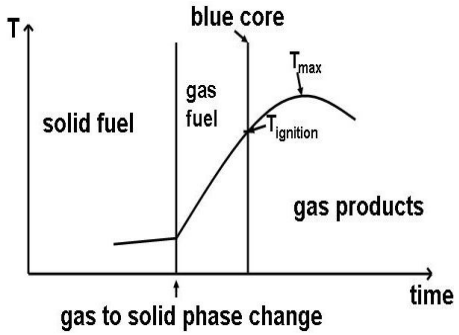


Figure 2: **Flame temperature profile for a solid (or gaseous) fuel.**

to the chromatic adaptation of the observer in order to get the correct colors of the fire.

2 Previous Work

A simple laminar flame was texture mapped onto a flame-like implicit primitive and then volume-traced by [Inakage 1989]. [Perry and Picard 1994] applied a velocity spread model from combustion science to propagate flames. [Chiba et al. 1994] computed the exchange of heat between objects by projecting the environment onto a plane. The spread of flame was a function of both the temperature and the concentration of fuel. [Stam and Fiume 1995] present a similar model in three spatial dimensions for the creation, extinguishing and spread of fire. The spread of the fire is controlled by the amount of fuel available, the geometry of the environment and the initial conditions. Their velocity field is predefined, and then the temperature and density fields are advected using an advection-diffusion type equation. They render the fire using a diffusion approximation which takes into account multiple scattering. [Bukowski and Sequin 1997] integrated the Berkeley Architectural Walkthrough Program with the National Institute of Standards and Technology's CFAST fire simulator. The integrated system creates a simulation based design environment for building fire safety systems.

Although we do not consider high-speed combustion phenomena such as detonations in this paper, there has been some notable work on this subject. [Musgrave 1997] concentrated on the explosive cloud portion of the explosion event using a fractal noise approach. [Neff and Fiume 1999] model and visualize the blast wave portion of an explosion based on a blast curve approach. [Mazarak et al. 1999] discuss the elementary blast wave equations, which were used to model exploding objects. They also show how to incorporate the blast wave model with a rigid body motion simulator to produce realistic animation of flying debris. Most recently, [Yngve et al. 2000] model the propagation of an explosion through the surrounding air using a computational fluid dynamics based approach to solve the equations for compressible, viscous flow. Their system includes two way coupling between solid objects and surrounding fluid, and uses the spectacular brittle fracture technology of [O'Brien and Hodgins 1999]. While the compressible flow equations are useful for modeling shock waves and other compressible phenomena, they introduce a very strict time step restriction associated with the acoustic waves. We use the incompressible flow equations instead to avoid this restriction making our method more computationally efficient.

3 Physically Based Model

We consider three distinct visual phenomena associated with flames. The first of these is the blue or bluish-green core seen in many flames. These colors are emission lines from intermediate chemical species, such as carbon radicals, produced during the chemical reaction. In the thin flame model, this thin blue core is located adjacent to the implicit surface. Therefore, in order to track this blue core, we need to track the movement of the implicit surface. The second visual phenomenon is the blackbody radiation emitted by the hot gaseous products, in particular the carbon soot. This is characterized by the yellowish-orange color we associate with fire. In order to model this with visual accuracy we need to track the temperatures associated with a flame as depicted in figure 2 (read from left to right). If the fuel is solid or liquid, the first step is the heating of the solid until it undergoes a phase change to the gaseous state. (Obviously, for gas fuels, we start in this gaseous state region in the figure.) Then the gas heats up until it reaches its ignition temperature corresponding to our implicit surface and the beginning of the thin blue core region. The temperature continues to increase as the reaction proceeds reaching a maximum before radiative cooling and mixing effects cause the temperature to decrease. As the temperature decreases, the blackbody radiation falls off until the yellowish-orange color is no longer visible. The third and final visual effect we address is the smoke or soot that is apparent in some flames after the temperature cools to the point where the blackbody radiation is no longer visible. We model this effect by carrying along a density variable in a fashion similar to the temperature. One could easily add particles to represent small pieces of soot, but our focus in this paper is the fire, not the smoke. For more details on smoke, see [Foster and Metaxas 1997], [Stam 1999] and [Fedkiw et al. 2001]. Figure 3 shows smoke coupled to our gas flame.

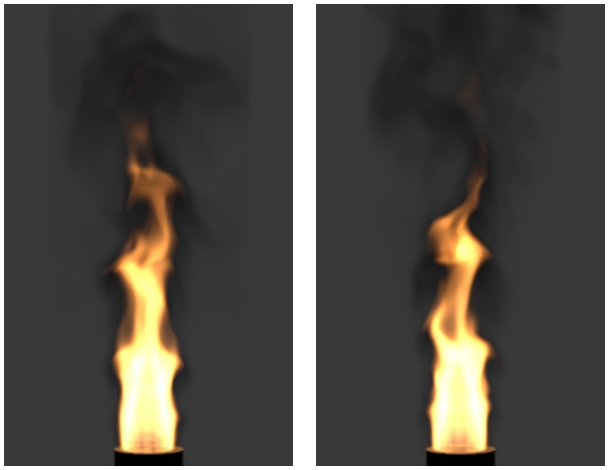


Figure 3: **The hot gaseous products and soot emit blackbody radiation that illuminates the smoke.**

3.1 Blue Core

Our implicit surface separates this gaseous fuel from the hot gaseous products and surrounding air. Consider for example the injection of gaseous fuel from a cylindrically shaped tube. If the fuel were not burning, then the implicit surface would simply move at the same velocity as the gaseous fuel being injected. However, when the fuel is reacting, the implicit surface moves at the velocity of the unreacted fuel plus a flame speed S that indicates how fast fuel is being converted into gaseous products. S indicates how fast the unreacted gaseous fuel is crossing over the implicit surface turning into hot gaseous products. The approximate surface area of the blue core, A_S, can be estimated with the following equation

$$v_f A_f = S A_S, \tag{1}$$

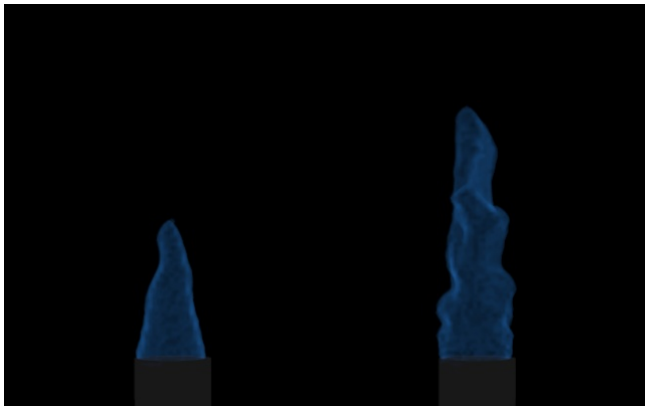


Figure 4: **Blue reaction zone cores for large (left) and small (right) values of the flame reaction speed S. Note the increased turbulence on the right.**

where v_f is the speed the fuel is injected across the injection surface with area A_f, e.g. A_f is the cross section of the cylindrical tube. This equation results from canceling out the density in the equation for conservation of mass. The left hand side is the fuel being injected into the region bounded by the implicit surface, and the right hand side is the fuel leaving this region crossing over the implicit surface as it turns into gaseous products. From this equation, we see that injecting more (less) gas is equivalent to increasing (decreasing) v_f resulting in a larger (smaller) blue core. Similarly, increasing (decreasing) the reaction speed S results in a smaller (larger) blue core. While we can turn the velocity up or down on our cylindrical jet, the reaction speed S is a property of the fuel. For example, S is approximately $.44m/s$ for a propane fuel that has been suitably premixed with oxidizer [Turns 1996]. (We use $S = .5m/s$ for most of our examples.) Figure 4 shows the effect of varying the parameter S. The smaller value of S gives a blue core with more surface area as shown in the figure.

This thin flame approximation is fairly accurate for premixed flames where the fuel and oxidizer are premixed so that the injected gas is ready for combustion. Non-premixed flames, commonly referred to as diffusion flames, behave somewhat differently. In a diffusion flame, the injected fuel has to mix with a surrounding oxidizer before it can combust. Figure 5 shows the injection of fuel out of a cylindrically shaped pipe. The cone shaped curve is the predicted location of the blue core for a premixed flame while the larger rounded curve is the predicted location of the blue core for a diffusion flame. As can be seen in the figure, diffusion flames tend to have larger cores since it takes a while for the injected fuel and surrounding oxidizer to mix. This small-scale molecular diffusion process is governed by a second order partial differential equation that is computationally costly model. Thus for visual purposes,

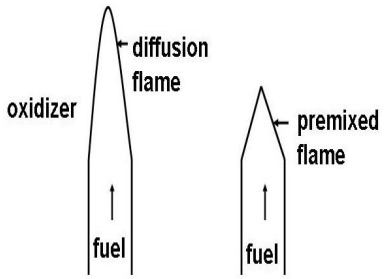


Figure 5: **Location of the blue reaction zone core for a premixed flame versus a diffusion (non-premixed) flame**

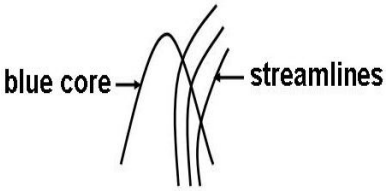


Figure 6: **Streamlines illustrating the path of individual fluid elements as they across the blue reaction zone core. The curved path is caused by the expansion of the gas as it reacts.**

we model diffusion flames with larger blue cores simply by using a smaller value of S than that used for a corresponding premixed flame.

3.2 Hot Gaseous Products

In order to get the proper visual look for our flames, it is important to track individual elements of the flow and follow them through their temperature histories given by figure 2. This is particularly difficult because the gas expands as it undergoes reaction from fuel to hot gaseous products. This expansion is important to model since it changes the trajectories of the gas and the subsequent look and feel of the flame as individual elements go through their temperature profile. Figure 3.2 shows some sample trajectories of individual elements as they cross over the reaction front. Note that individual elements do not go straight up as they pass through the reaction front, but instead turn outward due to the effects of expansion. It is difficult to obtain visually full turbulent flames without modeling this expansion effect. In fact, many practitioners resort to a number of low level hacks (and lots of random numbers) in an attempt to sculpt this behavior, while we obtain the behavior intrinsically by using the appropriate model. The expansion parameter is usually given as a ratio of densities, ρ_f/ρ_h where ρ_f is the density of the gaseous fuel and ρ_h is the density of the hot gaseous products. Figure 7 shows three flames side by side with increasing amounts of expansion from left to right. Note how increasing the expansion makes the flames appear fuller. We used $\rho_f = 1kg/m^3$ (about the density of air) for all three flames with $\rho_h = .2kg/m^3$, $.1kg/m^3$ and $.05kg/m^3$ from left to right.

We use one set of incompressible flow equations to model the fuel and a separate set of incompressible flow equations to model the hot gaseous products and surrounding airflow. We require a model for coupling these two sets of incompressible flow equations together across the interface in a manner that models the expansion that takes place across the reaction front. Given that mass and mo-

Figure 7: **Comparison of flame shapes for differing degrees of gaseous expansion. The amount of expansion increases from left to right making the flame appear fuller and more turbulent.**

mentum are conserved we can derive the following equations for the coupling across the thin flame front:

$$\rho_h(V_h - D) = \rho_f(V_f - D), \quad (2)$$

$$\rho_h(V_h - D)^2 + p_h = \rho_f(V_f - D)^2 + p_f, \quad (3)$$

where V_f and V_h are the normal velocities of the fuel and the hot gaseous products, and p_f and p_h are their pressures. Here, $D = V_f - S$ is the speed of the implicit surface in the normal direction. These equations indicate that both the velocity and the pressure are discontinuous across the flame front. Thus, we will need to exercise caution when taking derivatives of these quantities as is required when solving the incompressible flow equations. (Note that the tangential velocities are continuous across the flame front.)

3.3 Solid Fuels

When considering solid fuels, there are two expansions that need to be accounted for. Besides the expansion across the flame front, a similar expansion takes place when the solid is converted to a gas. However, S is usually relatively small for this reaction (most solids burn slowly in a visual sense), so we can use the boundary of the solid fuel as the reaction front. Since we do not currently model the pressure in solids, only equation 2 applies. We rewrite this equation as

$$\rho_f(V_f - D) = \rho_s(V_s - D), \quad (4)$$

where ρ_s and V_s are the density and the normal velocity of the solid fuel. Substituting $D = V_s - S$ and solving for V_f gives

$$V_f = V_s + \left(\rho_s/\rho_f - 1\right) S \quad (5)$$

indicating that the gasified solid fuel moves at the velocity of the solid fuel plus a correction that accounts for the expansion. We model this phase change by injecting gas out of the solid fuel at the appropriate velocity. This can be used to set arbitrary shaped solid objects on fire as long as they can be voxelized with a suitable surface normal assigned to each voxel indicating the direction of gaseous injection.

In figure 8, we simulate a campfire using two cylindrically shaped logs as solid fuel injecting gas out of the logs in a direction consistent with the local unit surface normal. Note the realistic rolling of the fire up from the base of the log. The ability to inject (or not inject) gaseous fuel out of individual voxels on the surface of a complex solid object allows us to animate objects catching on fire, burn different parts of an object at different rates or not at all (by using spatially varying injection velocities), and extinguish solid fuels simply by turning off the injection velocity. While building an animation system that allows the user to hand paint temporally and spatially varying injection velocities on the surface of solid objects is beyond the scope of this paper, it is the subject of future work.

4 Implementation

We use a uniform discretization of space into N^3 voxels with uniform spacing h. The implicit surface, temperature, density and pressure are defined at the voxel centers and are denoted $\phi_{i,j,k}$, $T_{i,j,k}$, $\rho_{i,j,k}$ and $p_{i,j,k}$ where $i,j,k = 1,\cdots,N$. The velocities are defined at the cell faces and we use half-way index notation: $u_{i+1/2,j,k}$ where $i = 0,\cdots,N$ and $j,k = 1,\cdots,N$; $v_{i,j+1/2,k}$ where $j = 0,\cdots,N$ and $i,k = 1,\cdots,N$; $w_{i,j,k+1/2}$ where $k = 0,\cdots,N$ and $i,j = 1,\cdots,N$.

Figure 8: **Two burning logs are placed on the ground and used to emit fuel. The crossways log on top is not lit so the flame is forced to flow around it.**

4.1 Level Set Equation

We track our reaction zone (blue core) using the level set method of [Osher and Sethian 1988] to track the moving implicit surface. We define ϕ to be positive in the region of space filled with fuel, negative elsewhere and zero at the reaction zone.

The implicit surface moves with velocity $\mathbf{w} = \mathbf{u}_f + S\mathbf{n}$ where $\mathbf{u}_f$ is the velocity of the gaseous fuel and the $S\mathbf{n}$ term governs the conversion of fuel into gaseous products. The local unit normal, $\mathbf{n} = \nabla\phi/|\nabla\phi|$ is defined at the center of each voxel using central differencing to approximate the necessary derivatives, e.g. $\phi_x \approx (\phi_{i+1,j,k} - \phi_{i-1,j,k})/2h$. Standard averaging of voxel face values is used to define $\mathbf{u}_f$ at the voxel centers, e.g. $u_{i,j,k} = (u_{i-1/2,j,k} + u_{i+1/2,j,k})/2$. The motion of the implicit surface is defined through

$$\phi_t = -\mathbf{w}\cdot\nabla\phi \quad (6)$$

and solved at each grid point using

$$\phi^{new} = \phi^{old} - \Delta t\left(w_1\phi_x + w_2\phi_y + w_3\phi_z\right) \quad (7)$$

and an upwind differencing approach to estimate the spatial derivatives. For example, if $w_1 > 0$, $\phi_x \approx (\phi_{i,j,k} - \phi_{i-1,j,k})/h$. Otherwise if $w_1 < 0$, $\phi_x \approx (\phi_{i+1,j,k} - \phi_{i,j,k})/h$. This simple approach is efficient and produces visually appealing blue cores.

To keep the implicit surface well conditioned, we occasionally adjust the values of ϕ in order to keep ϕ a signed distance function with $|\nabla\phi| = 1$. First, interpolation is used to reset the values of ϕ at voxels adjacent to the $\phi = 0$ isocontour (which we don't want to

move since it is the visual location of the blue core). Then we march out from the zero isocontour adjusting the values of ϕ at the other grid points as we cross them. [Tsitsiklis 1995] showed that this could be accomplished in an accurate, optimal and efficient manner solving quadratic equations and sorting points with a binary heap data structure. Later, [Sethian 1996] proposed the finite difference formulation of this algorithm that we currently use.

4.2 Incompressible Flow

We model the flow of the gaseous fuel and the hot gaseous products using a separate set of incompressible Euler equations for each. Incompressibility is enforced through conservation of mass (or volume), i.e. $\nabla \cdot \mathbf{u} = 0$ where $\mathbf{u} = (u, v, w)$ is the velocity field. The equations for the velocity

$$\mathbf{u}_t = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \nabla p/\rho + \mathbf{f} \tag{8}$$

are solved for in two parts. First, we use this equation to compute an intermediate velocity $\mathbf{u}^*$ ignoring the pressure term, and then we add the pressure (correction) term using

$$\mathbf{u} = \mathbf{u}^* - \Delta t \nabla p/\rho. \tag{9}$$

The key idea to this splitting method is illustrated by taking the divergence of equation 9 to obtain

$$\nabla \cdot \mathbf{u} = \nabla \cdot \mathbf{u}^* - \Delta t \nabla \cdot (\nabla p/\rho) \tag{10}$$

and then realizing that we want $\nabla \cdot \mathbf{u} = 0$ to enforce mass conservation. Thus the left hand side of equation 10 should vanish leaving a Poisson equation of the form

$$\nabla \cdot (\nabla p/\rho) = \nabla \cdot \mathbf{u}^*/\Delta t \tag{11}$$

that can be solved to find the pressure needed for updating equation 9.

We use a semi-Lagrangian stable fluids approach for finding the intermediate velocity $\mathbf{u}^*$ and refer the reader to [Stam 1999] and [Fedkiw et al. 2001] for the details. Since we use two sets of incompressible flow equations, we need to address the stable fluid update when a characteristic traced back from one set of incompressible flow equations crosses the implicit surface and queries the velocities from the other set of incompressible flow equations. Since the normal velocity is discontinuous across the interface, the straightforward stable fluids approach fails to work. Instead, we need to use the balance equation 2 for conservation of mass to correctly interpolate a velocity.

Suppose we are solving for the hot gaseous products and we interpolate across the interface into a region where a velocity from the gaseous fuel might incorrectly be used. Instead of using this value, we compute a ghost value as follows. First, we compute the normal velocity of the fuel, $V_f = \mathbf{u}_f \cdot \mathbf{n}$. Then we use the balance equation 2 to find a ghost value for V_h^G as

$$V_h^G = V_f + (\rho_f/\rho_h - 1)\, S. \tag{12}$$

Since the tangential velocities are continuous across the implicit surface, we combine this new normal velocity with the existing tangential velocity to obtain

$$\mathbf{u}_h^G = V_h^G \mathbf{n} + \mathbf{u}_f - (\mathbf{u}_f \cdot \mathbf{n})\mathbf{n} \tag{13}$$

as a ghost value for the velocity of the hot gaseous products in the region where only the fuel is defined. This ghost velocity can then be used to correctly carry out the stable fluids update. Since both $\mathbf{n}$ and $\mathbf{u}_f$ are defined throughout the region occupied by the fuel, and ρ_f, ρ_h and S are known constants, a ghost cell value for the hot gaseous products, $\mathbf{u}_h^G$, can be found anywhere in the fuel region (even quite far from the interface) by simply algebraically evaluating the right hand side of equation 13. [Nguyen et al. 2001] showed that this ghost fluid method, invented in [Fedkiw et al. 1999], could be used to compute physically accurate engineering simulations of deflagrations.

After computing the intermediate velocity $\mathbf{u}^*$ for both sets of incompressible flow equations, we solve equation 11 for the pressure and finally use equation 9 to find our new velocity field. Equation 11 is solved by assembling and solving a linear system of equations for the pressure as discussed in more detail in [Foster and Fedkiw 2001] and [Fedkiw et al. 2001]. Once again, we need to exercise caution here since the pressure is discontinuous across the interface. Using the ghost fluid method and equation 3, we can obtain and solve a slightly modified linear system incorporating this jump in pressure. We refer the reader to [Nguyen et al. 2001] for explicit details and a demonstration of the physical accuracy of this approach in the context of deflagration waves.

The temperature affects the fluid velocity as hot gases tend to rise due to buoyancy. We use a simple model to account for these effects by defining external forces that are directly proportional to the temperature

$$\mathbf{f}_{buoy} = \alpha (T - T_{air})\, \mathbf{z}, \tag{14}$$

where $\mathbf{z} = (0, 0, 1)$ points in the upward vertical direction, T_{air} is the ambient temperature of the air and α is positive constant with the appropriate units.

Fire, smoke and air mixtures contain velocity fields with large spatial deviations accompanied by a significant amount of rotational and turbulent structure on a variety of scales. Nonphysical numerical dissipation damps out these interesting flow features, so we aim to add them back on the coarse grid. We use the vorticity confinement technique invented by Steinhoff (see e.g. [Steinhoff and Underhill 1994]) and used by [Fedkiw et al. 2001] to generate the swirling effects for smoke. The first step in generating the small scale detail is to identify the vorticity $\boldsymbol{\omega} = \nabla \times \mathbf{u}$ as the source of this small scale structure. Each small piece of vorticity can be thought of as a paddle wheel trying to spin the flow field in a particular direction. Normalized vorticity location vectors, $\mathbf{N} = \nabla|\boldsymbol{\omega}|/|\nabla|\boldsymbol{\omega}||$ simply point from lower concentrations of vorticity to higher concentrations. Using these, the magnitude and direction of the vorticity confinement (paddle wheel) force is computed as

$$\mathbf{f}_{conf} = \varepsilon h (\mathbf{N} \times \boldsymbol{\omega}), \tag{15}$$

where $\varepsilon > 0$ and is used to control the amount of small scale detail added back into the flow field. The dependence on h guarantees that as the mesh is refined the physically correct solution is still obtained. All these quantities can be evaluated in a straightforward fashion as outlined in [Fedkiw et al. 2001].

Usually a standard CFL time step restriction dictates that the time step $\triangle t$ should be limited by $\triangle t < h/|\mathbf{u}|_{max}$ where $|\mathbf{u}|_{max}$ is the maximum velocity in the flow field. While this is true for our level set equation 6 with $\mathbf{u}$ replaced by $\mathbf{w}$, the combination of the semi-Lagrangian discretization and the ghost fluid method allows us to take a much larger time step for the incompressible flow equations. We choose our incompressible flow time step to be about five times bigger than that dictated by applying the CFL condition to the level set equation, and then stably update ϕ using substeps. This reduces the number of times one needs to solve for the pressure, which is the most expensive part of the calculation, by a factor of five.

4.3 Temperature and Density

The temperature profile has great effect on how we visually perceive flames, and we need to generate a temperature time history for fluid elements that behaves as shown in figure 2. Since this figure depicts a time history of the temperature of fluid elements, we

need a way to track individual fluid elements as they cross over the blue core and rise upward due to buoyancy. In particular, we need to know how much time has elapsed since a fluid element has passed through the blue core so that we can assign an appropriate temperature to it. This is easily accomplished using a reaction coordinate variable Y governed by the equation

$$Y_t = -(\mathbf{u}\cdot\nabla)Y - k, \tag{16}$$

where k is a positive constant which we take to be 1 (larger or smaller values can be used to get a good numerical variation of Y in the flame). Ignoring the convection term, $Y_t = -1$ can be solved exactly to obtain $Y(t) = -t + Y(0)$. If we set $Y(0) = 1$ in the region of space occupied by the gaseous fuel and solve equation 16 for Y, then the local value of $1-Y$ is equal to the total time elapsed since a fluid element crossed over the blue reaction core.

We solve equation 16 using the semi-Lagrangian stable fluids method to first update the convection term obtaining an intermediate value Y^*. Then we separately integrate the source term analytically so it too is stable for large time steps, i.e. $Y^{new} = -k\Delta t + Y^*$.

We can now use the values of Y to assign temperature values to the flow. Since $T_{ignition}$ is usually below the visual blackbody emission threshold, the temperature we set inside the blue core is usually not important. Therefore, we can set $T = T_{ignition}$ for the points inside the blue core. The region between the blue core and the maximum temperature in figure 2 is important since it models the rise in temperature due to the progress of a complex chemical reaction (which we do not model for the sake of efficiency). Here the animator has a lot of freedom to sculpt temperature rise curves and adjust how the mapping corresponds to the local Y values. For example, one could use $T = T_{ignition}$ at $Y = 1$, $T = T_{max}$ at $Y = .9$ and use a linear temperature function for the in between values of $Y \in (.9, 1)$. For large flames, this temperature rise interval will be compressed too close to the blue core for our grid to resolve. In these instances we use the ghost fluid method to set $T = T_{max}$ for any characteristic that looks across the blue core into the gaseous fuel region. The blue core then "spits" out gas at the maximum temperature that immediately starts to cool off, i.e. there is no temperature rise region. In fact, we did not find it necessary to use the temperature rise region in our examples as we are interested in larger scale flames, but this temperature rise region would be useful, for example, when modeling candle.

The animator can also sculpt the temperature falloff region to the right of figure 2. However, there is a physically correct, viable (i.e. computationally cheap) alternative. For the values of Y in the temperature falloff region, we simply solve

$$T_t = -(\mathbf{u}\cdot\nabla)T - c_T\left(\frac{T - T_{air}}{T_{max} - T_{air}}\right)^4 \tag{17}$$

which is derived from conservation of energy. Similar to equation 16, we solve this equation by first using the semi-Lagrangian stable fluids method to solve for the convection term. Then we integrate the fourth power term analytically to cool down the flame at a rate governed by the cooling constant c_T.

Similar to the temperature curve in figure 2, the animator can sculpt a density curve for smoke and soot formation. The density should start low and increase as the reaction proceeds. In the temperature falloff region, the animator can switch from the density curve to a physically correct equation

$$\rho_t = -(\mathbf{u}\cdot\nabla)\rho \tag{18}$$

that can (once again) be solved using the semi-Lagrangian stable fluids method. Again, we did not find it necessary to sculpt densities for our particular examples.

5 Rendering of Fire

Fire is a participating medium. It is more complex than the types of participating media (e.g. smoke and fog) that are typically encountered in computer graphics since fire emits light. The region that creates the light-energy typically has a complex shape, which makes it difficult to sample. Another complication with fire is that the fire is bright enough that our eyes adapt to its color. This chromatic adaptation is important to account for when displaying fire on a monitor. In this section, we will first describe how we simulate the scattering of light within a fire-medium. Then, we will detail how to properly integrate the spectral distribution of power in the fire and account for chromatic adaptation.

5.1 Light Scattering in a Fire Medium

Fire is a blackbody radiator and a participating medium. The properties of a participating medium are described by the scattering, absorption and emission properties. Specifically, we have the scattering coefficient, σ_s, the absorption coefficient, σ_a, and the extinction coefficient, $\sigma_t = \sigma_a + \sigma_s$. These coefficients specify the amount of scattering, absorption and extinction per unit-distance for a beam of light moving through the medium. The spherical distribution of the scattered light at a location is specified by a phase-function, p. We use the Henyey-Greenstein phase-function [Henyey and Greenstein 1941]

$$p(\vec{\omega}\cdot\vec{\omega}') = \frac{1-g^2}{4\pi(1+g^2-2g\vec{\omega}\cdot\vec{\omega}')^{1.5}}. \tag{19}$$

Here, $g \in [-1, 1]$ is the scattering anisotropy of the medium, $g > 0$ is forward scattering, $g < 0$ is backward scattering, while $g = 0$ is isotropic scattering. Note that the distribution of the scattered light only depends on the angle between the incoming direction, $\vec{\omega}$, and the outgoing direction, $\vec{\omega}'$.

Light transport in participating media is described by an integro-differential equation, the radiative transport equation [Siegel and Howell 1981]:

$$\begin{aligned}(\vec{\omega}\cdot\nabla)L_\lambda(x,\vec{\omega}) \quad = \quad & -\sigma_t(x)L_\lambda(x,\vec{\omega}) + \\ & \sigma_s(x)\int_{4\pi} p(\vec{\omega},\vec{\omega}')L_\lambda(x,\vec{\omega}')d\vec{\omega}' + \\ & \sigma_a(x)L_{e,\lambda}(x,\vec{\omega}).\end{aligned} \tag{20}$$

Here, L_λ is the spectral radiance, and $L_{e,\lambda}$ is the emitted spectral radiance. Note that σ_s, σ_a, and σ_t vary throughout the medium and therefore depend on the position x.

We solve Equation 20 to estimate the radiance distribution in the medium by using a stochastic adaptive ray marching algorithm which recursively samples multiple scattering. In highly scattering media this approach is costly; however, we are concerned about fire which is a blackbody radiator (no scattering, only absorption) that creates a low-albedo smoke (the only scattering part of the fire-medium). This makes the Monte Carlo ray tracing approach practical.

To estimate the radiance along a ray traversing the medium, we split the ray into short segments. For a given segment, n, the scattering properties of the medium are assumed constant, and the radiance, L_n, at the start of the segment is computed as:

$$\begin{aligned}L_{n,\lambda}(x,\vec{\omega}) \quad = \quad & e^{-\sigma_t\Delta x}L_{(n-1),\lambda}(x+\Delta x,\vec{\omega}) + \\ & L_\lambda(x,\vec{\omega}')p(\vec{\omega}\cdot\vec{\omega}')\sigma_s\Delta x + \\ & \sigma_a L_{e,\lambda}(x)\Delta x.\end{aligned} \tag{21}$$

This equation is evaluated recursively to compute the total radiance at the origin of the ray. Δx is the length of the segment, L_{n-1} is the radiance at the beginning of the next segment, and $\vec{\omega}'$ is a sample

Figure 9: **A metal ball passes through and interacts with a gas flame.**

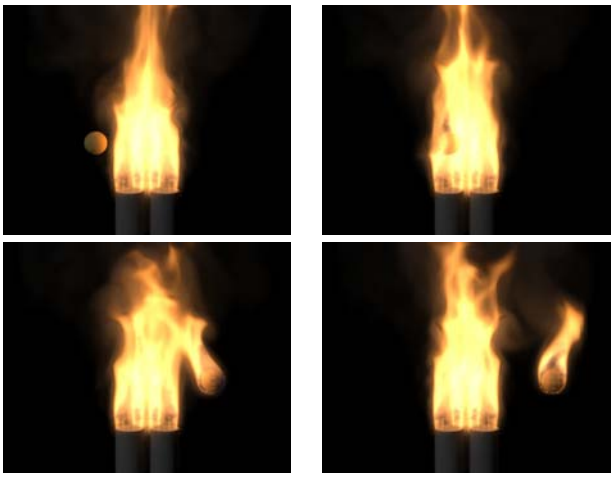


Figure 10: **A flammable ball passes through a gas flame and catches on fire.**

direction for a new ray that evaluates the indirect illumination in a given direction for the segment. We find the sample direction by importance sampling the Henyey-Greenstein phase function. Note that we do not explicitly sample the fire volume; instead we rely on the Monte Carlo sampling to pick up energy as sample rays hit the fire. This strategy is reasonably efficient in the presence of the low-albedo smoke generated by the fire.

The emitted radiance is normally ignored in graphics, but for fire it is an essential component. For a blackbody we can compute the emitted spectral radiance using Planck's formula:

$$L_{e,\lambda}(x) = \frac{2C_1}{\lambda^5(e^{C_2/(\lambda T)} - 1)}, \tag{22}$$

where T is the temperature, $C_1 \approx 3.7418 \cdot 10^{-16} W m^2$, and $C_2 \approx 1.4388 \cdot 10^{-2} m^{\circ} K$ [Siegel and Howell 1981]. In the next section, we will describe how to properly render fire taking this spectral distribution of emitted radiance into account.

5.2 Reproducing the Color of Fire

Accurately reproducing the colors of fire is critical for a realistic fire rendering. The full spectral distribution can be obtained directly by using Planck's formula for spectral radiance when performing the ray marching. This spectrum can then be converted to RGB before being displayed on a monitor. To get the right colors of fire out of this process it is necessary to take into account the fact that our eyes adapt to the spectrum of the fire.

To compute the chromatic adaptation for fire, we use a von Kries transformation [Fairchild 1998]. We assume that the eye is adapted to the color of the spectrum for the maximum temperature present in the fire. We map the spectrum of this white point to the LMS cone responsivities (L_w, M_w, S_w). This enables us to map a spectrum to the monitor as follows. We first integrate the spectrum to find the raw XYZ tristimulus values (X_r, Y_r, Z_r). We then find the adapted XYZ tristimulus values (X_a, Y_a, Z_a) as:

$$\begin{bmatrix} X_a \\ Y_a \\ Z_a \end{bmatrix} = \mathbf{M}^{-1} \begin{bmatrix} 1/L_w & 0 & 0 \\ 0 & 1/M_w & 0 \\ 0 & 0 & 1/S_w \end{bmatrix} \mathbf{M} \begin{bmatrix} X_r \\ Y_r \\ Z_r \end{bmatrix}. \tag{23}$$

Here, $\mathbf{M}$ maps the XYZ colors to LMS (consult [Fairchild 1998] for the details). Finally, we map the adapted XYZ tristimulus values to the monitor RGB space using the monitor white point.

In our implementation, we integrate the spectrum of the blackbody at the source (e.g. when emitted radiance is computed); we then map this spectrum to RGB before using it in the ray marcher. This is much faster than doing a full spectral participating media simulation, and we found that it is sufficiently accurate, since we already assume that the fire is the dominating light source in the scene when doing the von Kries transformation.

6 Results

Figure 1 shows a frame from a simulation of a flamethrower. We used a domain that was 8 meters long with 160 grid cells in the horizontal direction ($h = .05$). The flame was injected at $30m/s$ out of a cylindrical pipe with diameter $.4m$. We used $S = .5m/s$, $rho_f = 1kg/m^3$, $rho_h = .01kg/m^3$, $c_T = 3000K/s$ and $\alpha = .15m/(Ks^2)$. The vorticity confinement parameter was set to $\varepsilon = 16$ for the gaseous fuel and to $\varepsilon = 60$ for the hot gaseous products. The simulation cost was approximately 3 minutes per frame using a Pentium IV.

Solid objects are treated by first tagging all the voxels inside the object as occupied. Then all the occupied voxel cell faces have their velocity set to that of the object. The temperature at the center of occupied voxels is set to the object's temperature and the (smoke) density is set to zero. Figure 9 shows a metal sphere as it passes through and interacts with a gas fire. Note the reflection of the fire on the surface of the sphere. For more details on object interactions with liquids and gases see [Foster and Fedkiw 2001] and [Fedkiw et al. 2001].

Since we have high temperatures (i.e. fire) in our flow field, we allow our objects to heat up if their temperature is lower than that of their surroundings. We use a simple conduction model where we increase the local temperature of an object depending on the surrounding air temperature and object temperature as well as the time step Δt. Normally, the value of the implicit surface is set to a negative value of h at the center of all voxels occupied by objects indicating that there is no available fuel. However, we can easily model ignition for objects we designate as flammable. Once the temperature of a voxel inside an object increases above a predefined threshold indicating ignition, we change the value of the implicit surface in that voxel from $-h$ to h indicating that it contains fuel. In addition, those voxel's faces have their velocities augmented above the object velocity by an increment in the direction normal to the object surface indicating that gaseous fuel is being injected according to the phase change addressed earlier for solid fuels. In figure 10, we illustrate this technique with a spherical ball that heats up and subsequently catches on fire as it passes through the flame. Both this flammable ball and the metal ball were com-

puted on a $120 \times 120 \times 120$ grid at approximately 5 minutes per frame.

7 Conclusion

We have presented a physically based model for animating and rendering fire and flames. We demonstrated that this model could be used to produce realistic looking turbulent flames from both solid and gaseous fuels. We showed plausible interaction of our fire and smoke with objects, including ignition of objects by the flames.

8 Acknowledgment

Research supported in part by an ONR YIP and PECASE award N00014-01-1-0620, NSF DMS-0106694, NSF ACI-0121288, and the DOE ASCI Academic Strategic Alliances Program (LLNL contract B341491).

References

BUKOWSKI, R., AND SEQUIN, C. 1997. Interactive Simulation of Fire in Virtual Building Environments. In *Proceedings of SIGGRAPH 1997*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 35–44.

CHIBA, N., MURAOKA, K., TAKAHASHI, H., AND MIURA, M. 1994. Two dimensional Visual Simulation of Flames, Smoke and the Spread of Fire. *The Journal of Visualization and Computer Animation 5*, 37–53.

FAIRCHILD, M. 1998. *Color Appearance Models*. Addison Wesley Longman, Inc.

FEDKIW, R., ASLAM, T., MERRIMAN, B., AND OSHER, S. 1999. A Non-oscillatory Eulerian Approach to Interfaces in Multimaterial Flows (The Ghost Fluid Method). *J. Comput. Phys. 152*, 457.

FEDKIW, R., STAM, J., AND JENSEN, H. W. 2001. Visual Simulation of Smoke. In *Proceedings of SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, E. Fiume, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 15–22.

FOSTER, N., AND FEDKIW, R. 2001. Practical Animation of Liquids. In *Proceedings of SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 23–30.

FOSTER, N., AND METAXAS, D. 1997. Modeling the Motion of a Hot, Turbulent Gas. In *Proceedings of SIGGRAPH 97*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 181–188.

HENYEY, L., AND GREENSTEIN, J. 1941. Diffuse radiation in the galaxy. *Astrophysics Journal 93*, 70–83.

INAKAGE, M. 1989. A Simple Model of Flames. In *Proceedings of Computer Graphics International 89*, Springer-Verlag, 71–81.

MARKSTEIN, G. H. 1964. *Nonsteady Flame Propagation*. Pergamon, Oxford.

MAZARAK, O., MARTINS, C., AND AMANATIDES, J. 1999. Animating Exploding Objects. In *Proceedings of Graphics Interface 99*, 211–218.

MUSGRAVE, F. K. 1997. Great Balls of Fire. In *SIGGRAPH 97 Animation Sketches, Visual Procedings*, ACM SIGGRAPH, 259–268.

NEFF, M., AND FIUME, E. 1999. A Visual Model for Blast Waves and Fracture. In *Proceedings of Graphics Interface 99*, 193–202.

NGUYEN, D., FEDKIW, R., AND KANG, M. 2001. A Boundary Condition Capturing Method for Incompressible Flame Discontinuities. *J. Comput. Phys. 172*, 71–98.

O'BRIEN, J. F., AND HODGINS, J. K. 1999. Graphical Modeling and Animation of Brittle Fracture. In *Proceedings of SIGGRAPH 1999*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 137–146.

OSHER, S., AND SETHIAN, J. A. 1988. Fronts Propagating with Curvature Dependent Speed: Algorithms Based on Hamilton-Jacobi Formualtions. *J. Comput. Phys. 79*, 12.

PERRY, C., AND PICARD, R. 1994. Synthesizing Flames and their Spread. *SIGGRAPH 94 Technical Sketches Notes* (July).

RUSHMEIER, H. 1994. Rendering Participating Media: Problems and Solutions from Application Areas. In *Proceedings of the 5th Eurographics Workshop on Rendering*, 35–56.

SETHIAN, J. 1996. A Fast Marching Level Set Method for Monotonically Advancing Fronts. *Proc. Nat. Acad. Sci. 93*, 1591–1595.

SIEGEL, R., AND HOWELL, J. 1981. *Thermal Radiation Heat Transfer*. Hemisphere Publishing Corp., Washington, DC.

STAM, J., AND FIUME, E. 1995. Depicting Fire and Other Gaseous Phenomena Using Diffusion Process. In *Proceedings of SIGGRAPH 95*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 129–136.

STAM, J. 1999. Stable Fluids. In *SIGGRAPH 99 Conference Proceedings, Annual Conference Series*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 121–128.

STANIFORTH, A., AND COTE, J. 1991. Semi-Lagrangian Integration Schemes for Atmospheric Models: A Review. *Monthly Weather Review 119*, 2206–2223.

STEINHOFF, J., AND UNDERHILL, D. 1994. Modification of the Euler Equations for "Vorticity Confinement": Application to the Computation of Interacting Vortex Rings. *Physics of Fluids 6*, 8, 2738–2744.

TSITSIKLIS, J. 1995. Efficient Algorithms for Globally Optimal Trajectories. *IEEE Transactions on Automatic Control 40*, 1528–1538.

TURNS, S. R. 1996. *An Introduction to Combustion*. McGraw-Hill, Inc.

YNGVE, G. D., O'BRIEN, J. F., AND HODGINS, J. K. 2000. Animating Explosions. In *Proceedings of SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 29–36.

Practical Animation of Liquids

Nick Foster*
PDI/DreamWorks

Ronald Fedkiw**
Stanford University

Abstract

We present a general method for modeling and animating liquids. The system is specifically designed for computer animation and handles viscous liquids as they move in a 3D environment and interact with graphics primitives such as parametric curves and moving polygons. We combine an appropriately modified semi-Lagrangian method with a new approach to calculating fluid flow around objects. This allows us to efficiently solve the equations of motion for a liquid while retaining enough detail to obtain realistic looking behavior. The object interaction mechanism is extended to provide control over the liquid's 3D motion. A high quality surface is obtained from the resulting velocity field using a novel adaptive technique for evolving an implicit surface.

Keywords: animation, computational fluid dynamics, implicit surface, level set, liquids, natural phenomena, Navier-Stokes, particles, semi-Lagrangian.

1. Introduction

The desire for improved physics-based animation tools has grown hand in hand with the advances made in computer animation on the whole. It is natural then, that established engineering techniques for simulating and modeling the real world have been modified and applied to computer graphics more frequently over the last few years. One group of methods that have resisted this transition are those used to model the behavior of liquids from the field of computational fluid dynamics (CFD). Not only are such techniques generally complex and computationally intensive, but they are also not readily adaptable to what could be considered the basic requirements of a computer animation system.

One of the key difficulties encountered when using these methods for animation directly characterizes the trade off between simulation and control. Physics-based animations usually rely on direct numerical simulation (DNS) to achieve realism. In engineering terms, this means that initial conditions and boundary conditions are specified and the process is left to run freely with only minor influence on the part of the animator. The majority of engineering techniques for liquid simulation assume this model.

From an animation viewpoint, we are interested in using numerical techniques to obtain behaviors that would be prohibitive to model by hand. At the same time we want control over the global, low frequency motion so we can match it to the behavior we are trying to create. This then becomes the goal when transitioning between engineering and computer animation; preserve as much of the realistic behavior as feasible while allowing for control over motion on both a local and global scale. This has to be achieved without compromising the overall requirement of a visually coherent and realistic look.

* nickf@pdi.com, ** fedkiw@cs.stanford.edu

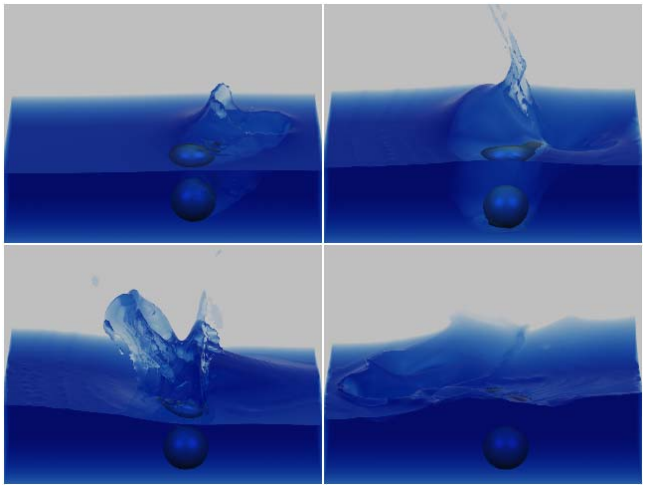


***Figure 1**: A ball splashes into a tank of water.*

This paper specifically addresses these issues for liquid animation. The method presented is for animating viscous liquids ranging from water to thick mud. These liquids can freely mix, move arbitrarily within a fixed three-dimensional grid and interact realistically with stationary or moving polygonal objects. This is achieved for animation by trading off engineering correctness for computational efficiency.

We start with the Navier-Stokes equations for incompressible flow and solve for liquid motion using an adaptation of a semi-Lagrangian method introduced recently to graphics for solving fluid flows [25]. These methods usually result in mass dissipation. While not an issue for gas or smoke, this is visually unacceptable for modeling liquids. We correct for this by tracking the motion of the liquid surface using a novel hybrid combination of inertialess particles and an implicit surface called a level set. The level set prevents mass dissipation while the particles allow the liquid to still splash freely. A useful consequence is that this combined surface can be rendered in a highly believable way.

The next innovation involves taking account of the effects of moving polygonal objects within the liquid. We develop a new technique that, while not accurate in an engineering sense, satisfies the physics of object/liquid interactions and looks visually realistic. This method is efficient and robust, and we show that it can be adapted to provide low frequency directional control over the liquid volume. This allows us to efficiently calculate liquid behavior that would be impossible to get by hand,

while at the same time allowing us to "dial-in" specific motion components.

When the techniques described above are applied together, the result is a comprehensive system for modeling and animating liquids for computer graphics. The main contributions of the system are a numerical method that takes the minimal computational effort required for visual realism combined with tailor-made methods for handling moving objects and for maintaining a smooth, temporally coherent liquid surface.

2. Previous Work

The behavior of a volume of liquid can be described by a set of equations that were jointly developed by Navier and Stokes in the early eighteen hundreds (see next section). The last fifty years has seen an enormous amount of research by the CFD community into solving these equations for a variety of engineering applications. We direct the interested reader to Abbot and Basco [1] which covers some of the important principles without being too mathematically dense.

Early graphics work concentrated on modeling just the surface of a body of water as a parametric function that could be animated over time to simulate wave transport [12, 22, 23]. Kass and Miller [17] approximated the 2D shallow water equations to get a dynamic height field surface that interacted with a static ground "object". Chen and Lobo [4] extended the height field approach by using the pressure arising from a 2D solution of the Navier-Stokes equations to modulate surface elevation. O'Brien and Hodgins [20] simulated splashing liquids by combining a particle system and height field, while Miller and Pearce [19] used viscous springs between particles to achieve dynamic flow in 3D. Terzopoulos, Platt and Fleischer [27] simulated melting deformable solids using a molecular dynamics approach to simulate the particles in the liquid phase.

Surface or particle based methods are relatively fast, especially in the case of large bodies of water, but they don't address the full range of motion exhibited by liquids. Specifically, they don't take advantage of the realism inherent in a full solution to the Navier-Stokes equations. They are also not easily adapted to include interaction with moving objects. Foster and Metaxas [11] modified an original method by Harlow and Welch [15] (later improved by others, see e.g. [5]) to solve the full equations in 3D with arbitrary static objects and extended it to include simple control mechanisms [9]. Foster and Metaxas also applied a similar technique to model hot gases [10]. Stam [25] replaced their finite difference scheme with a semi-Lagrangian method to achieve significant performance improvements at the cost of increased rotational damping. Yngve et al. used a finite difference scheme to solve the compressible Navier-Stokes equations to model shock wave and convection effects generated by an explosion [28].

3. Method Outline

The Navier-Stokes equations for describing the motion of a liquid consist of two parts. The first, enforces incompressibility by saying that mass should always be conserved, i.e.

$$\nabla \cdot \mathbf{u} = 0 , \qquad (3.1)$$

where $\mathbf{u}$ is the liquid velocity field, and

$$\nabla = \left(\partial / \partial \mathrm{x}, \partial / \partial \mathrm{y}, \partial / \partial \mathrm{z} \right)$$

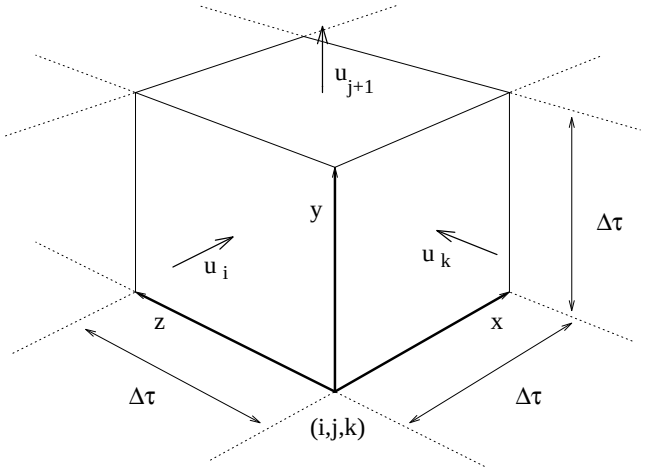


Figure 2: *A single grid cell with three of its six face velocities shown.*

is the gradient operator. The second equation couples the velocity and pressure fields and relates them through the conservation of momentum, i.e.

$$\mathbf{u}_t = \nu \nabla \cdot \left(\nabla \mathbf{u} \right) - \left(\mathbf{u} \cdot \nabla \right) \mathbf{u} - \frac{1}{\rho} \nabla \mathrm{p} + \mathbf{g} . \qquad (3.2)$$

This equation models the changes in the velocity field over time due to the effects of viscosity (ν), convection, density (ρ), pressure (p), and gravity (**g**). By solving (3.1) and (3.2) over time, we can model the behavior of a volume of liquid. The new algorithm we are proposing to do this consists of six straightforward steps.

I. Model the static environment as a voxel grid.

II. Model the liquid volume using a combination of particles and an implicit surface.

Then, for each simulation time step

III. Update the velocity field by solving (3.2) using finite differences combined with a semi-Lagrangian method.

IV. Apply velocity constraints due to moving objects.

V. Enforce incompressibility by solving a linear system built from (3.1).

VI. Update the position of the liquid volume (particles and implicit surface) using the new velocity field.

These steps are described in detail in the following sections. Steps IV and V are presented in reverse order for clarity.

4. Static Environment

Equations (3.1) and (3.2) model a liquid as two coupled dynamic fields, velocity and pressure. The motion of the liquid we are modeling will be determined by evolving these fields over time. We start by representing the environment that we want the liquid to move in as a rectangular grid of voxels with side length Δτ. The grid does not have to be rectangular, but the overhead of unused (non-liquid containing) cells will be low and so it is convenient. Each cell has a pressure variable at its center and shares a velocity variable with each of its adjacent neighbors (see figure 2). This velocity is defined at the center of the face shared

by the two neighboring cells and represents the magnitude of the flow normal to that face. This is the classic "staggered" MAC grid [15]. Each cell is then either tagged as being empty (available to be filled with liquid) or filled completely with an impermeable static object. Despite the crude voxelized approximation of both objects and the liquid volume itself, we'll show that we can still obtain and track a smooth, temporally coherent liquid surface.

5. Liquid Representation

The actual distribution of liquid in the environment is represented using an implicit surface. The implicit function is derived from a combination of inertialess particles and a dynamic isocontour. The isocontour provides a smooth surface to regions of liquid that are well resolved compared to our grid, whereas the particles provide detail where the surface starts to splash.

5.1 Particles

Particles are placed (or introduced via a source) into the grid according to some initial liquid distribution. Their positions then evolve over time by simple convection. Particle velocity is computed directly from the velocity grid using tri-linear interpolation and each particle is moved according to the inertialess equation $d\mathbf{x}_p/dt = \mathbf{v}_x$, where $\mathbf{v}_x$ is the fluid velocity at $\mathbf{x}_p$. Particles have a low computational overhead and smoothly integrate the changing liquid velocity field over time. The obvious drawback to using them, however, is that there is no straightforward way to extract a smooth polygonal (or parametric) description of the actual liquid surface. This surface is preferred because we want to render the liquid realistically using traditional computer graphics techniques. It is possible to identify it by connecting all the particles together into triangles, although deducing both the connectivity and set of surface triangles is difficult. In addition, since the particles do not generally form a smooth surface, the resulting polygonal mesh suffers from temporal aliasing as triangles "pop" in or out.

5.2 Isocontour

An alternative technique for representing the liquid surface is to generate it from an isocontour of an implicit function. The function is defined on a high resolution Eulerian sub-grid that sits inside the Navier-Stokes grid. Let each particle represent the center of an implicitly defined volumetric object (see Bloomenthal et al. [3] for a survey of implicit surfaces). Specifically, an implicit function centered at the particle location $\mathbf{x}_p$ with radius r is given by

$$\phi_p(\mathbf{x}) = \sqrt{(x_i - x_{pi})^2 + (x_j - x_{pj})^2 + (x_k - x_{pk})^2} - r .$$

The surface of that particle is defined as the spherical shell around $\mathbf{x}_p$ where $\phi_p(\mathbf{x})=0$. An implicit function, $\phi(\mathbf{x})$, is then defined over all the particles by taking the value of $\phi_p(\mathbf{x})$ from the particle closest to $\mathbf{x}$. If we sample $\phi(\mathbf{x})$ at each sub-grid point we can use a marching cubes algorithm [18] to tessellate the $\phi(\mathbf{x})=0$ isocontour with polygons. More sophisticated blend functions could be used to create an implicit function, however, we are going to temporally and spatially smooth $\phi(\mathbf{x})$ so it isn't necessary. We refer those interested in wrapping implicit surfaces around particles to the work of Desbrun and Cani-Gascuel [7].

The first step towards smoothing the surface is to normalize ϕ so that $|\phi(\mathbf{x})|$ equals the distance from $\mathbf{x}$ to the closest point on the zero isocontour. The sign of ϕ is set negative inside the liquid and positive outside. This signed distance function can be created quickly using the Fast Marching Method [24] starting from the initial guess of $\phi(\mathbf{x})$ defined by the particles as outlined above.

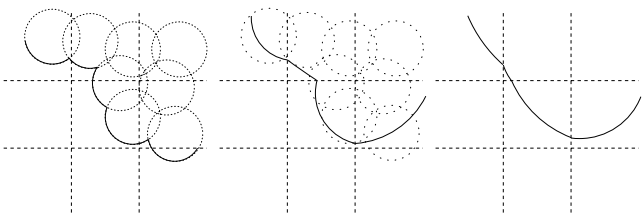


Figure 3: *The isocontour due to the implicit function around the particles, interpolated ϕ values, and smoothed ϕ values, respectively.*

In order to smooth out ϕ to reduce unnatural "folds" or "corners" in the surface (see figure 3), a smoothing equation of the form

$$\phi_\eta = -S\left(\phi^{\eta=0}\right)\left(\left|\nabla\phi\right| - 1\right), \qquad (5.1)$$

is used to modify values of ϕ close to the $\phi(\mathbf{x})=0$ isocontour. $S(\phi)$ is a smoothed sign function given by

$$S(\phi) = \frac{\phi}{\sqrt{\phi^2 + \Delta\tau^2}} .$$

If applied for a few relaxation steps in fictitious time η (everything else remains constant), (5.1) smooths out the $\phi(\mathbf{x})=0$ isocontour while maintaining overall shape. Once smoothed, the isocontour can be ray traced directly using a root finding algorithm to identify the zero values of ϕ. A fast root finder can be built easily because at any sub-grid point the value of ϕ explicitly gives the minimum step to take to get closer to the surface. Note that the surface normal is given by $\mathbf{n} = \nabla\phi / \left|\nabla\phi\right|$.

By creating a smooth isocontour for each frame of animation, we get an improved surface representation compared to using particles alone. There are still drawbacks however. A high density of particles is required at the $\phi(\mathbf{x})=0$ isocontour before the surface looks believably flat. Particles are also required throughout the entire liquid volume even when it's clear that they make no contribution to the visible surface. The solution is to create ϕ once using the particles, and then track how it moves using the same velocity field that we're using to move the particles. This leads to a temporally smoothed dynamic isosurface known in the CFD literature as a level set.

5.3 Dynamic Level Set

An obvious way to track the evolution of the surface of a volume of liquid would be to attach particles directly to the surface in its initial position and then just move them around in the velocity field. This would require adding extra particles when the surface becomes too sparsely resolved, and removing them as the surface folds, or "splashes" back over itself. An alternative method which is intuitively similar, but that doesn't use particles, was developed by Osher and Sethian [21] and is called the level set method.

We want to evolve ϕ directly over time using the liquid velocity field $\mathbf{u}$. We have a smooth surface but need to conform, visually at least, to the physics of liquids. It has been shown [21] that the

equation to update ϕ under these circumstances has the following structure,

$$\phi_t + \mathbf{u} \cdot \nabla \phi = 0 \,. \qquad (5.2)$$

Using (5.2), the surface position is evolved over time by tracking $\phi(\mathbf{x})=0$. The $(\mathbf{u} \cdot \nabla \phi)$ term is a convection term similar to the $(\mathbf{u} \cdot \nabla)\mathbf{u}$ term in (3.2) implying that we could use a semi-Lagrangian method to solve this equation. However, since this equation represents the mass evolution of our liquid, the semi-Lagrangian method tends to be too inaccurate. Instead we use a higher order upwind differencing procedure [1] on the $(\mathbf{u} \cdot \nabla \phi)$ term. Fedkiw et al. [8] used this methodology to track a fluid surface and give explicit details on solving (5.2). This method can suffer from severe volume loss especially on the relatively coarse grids commonly used in computer graphics. This is clearly visible when regions of liquid break away during splashing and then disappear because they are too small to be resolved by the level set. We require visual coherency for this to be a useful graphics technique and so the level set method needs to be modified to preserve volume.

5.4 Hybrid Surface Model

Particle evolution is a fully Lagrangian approach to the mass motion while level set evolution is a fully Eulerian approach. Since they tend to have complementary strengths and weakness, a combined approach gives superior results under a wider variety of situations. Level set evolution suffers from volume loss near detailed features while particle evolution suffers from visual artifacts in the surface when the number of particles is small. Conversely, the level set is always smooth, and particles retain detail regardless of flow complexity. Therefore we suggest a novel combination of the two approaches.

At each time step we evolve the particles and the level set ϕ forward in time. Next, we use the updated value of the level set function to decide how to treat each particle. If a particle is more than a few grid cells away from, and inside the surface, as indicated by the locally interpolated value of ϕ, then that particle is deleted. This increases efficiency since particles are only needed locally near the surface of the liquid as opposed to throughout the entire liquid volume. In addition, for cells close to $\phi(\mathbf{x})=0$ that are sparsely populated, extra particles can be introduced "within" the isocontour. Thus, for a bounded number of particles, we get improved surface resolution.

Next, for each particle near the surface, the locally interpolated curvature of the interface, calculated as

$$k = \nabla \cdot \left(\nabla\phi \Big/ \left|\nabla\phi\right| \right),$$

is used to indicate whether or not the surface is smooth. Smooth regions have low curvature and the particles are ignored allowing the level set function to give a very smooth representation of the liquid surface. On the other hand, regions of high curvature indicate splashing. In these regions, the particles are a better indicator of the rough surface topology. Particles in these regions are allowed to modify the local values of ϕ. At grid points where the implicit basis function for the particle would give a smaller value of ϕ (i.e. a particle is "poking" out of the zero level set), this smaller value is used to replace the value obtained from the time evolution of ϕ.

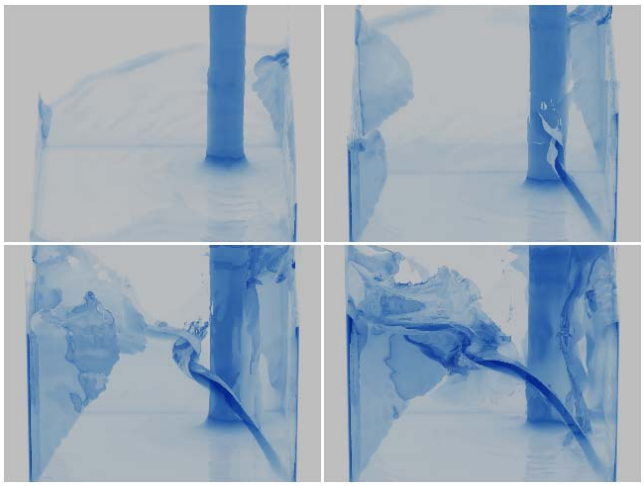


Figure 4: *Water pours into a container causing a complex surface to develop.*

Even with the tight coupling between the particles and the level set, some particles will escape the inside of the liquid layer since the grid is too coarse to represent them individually. These particles can be rendered directly as small liquid drops. In addition, these stray particles could be used as control particles to indicate the presence of fine spray or mist.

6. Updating the Velocity Field

We have a representation of the graphics environment and a way of tracking the surface of a volume of liquid. We can now apply (3.2) to the existing velocity field to advance it through an Euler-integration time step Δt. The equation is solved in three stages. First we compute Δt using the CFL condition (see Appendix A). Next, we update the convective component, i.e. $(\mathbf{u} \cdot \nabla)\mathbf{u}$, using a first order semi-Lagrangian method, as per Courant et al. [6] and by Stam [25]. We use the same formulation as Stam and refer readers to his description. Standard central differencing is then used on the viscous terms of (3.2) as described by Foster and Metaxas [11]. The results from this and the preceding calculation are added together to get an updated (though not mass conserving) velocity field for time $t+\Delta t$.

Semi-Lagrangian methods allow us to take large time steps without regard for the sometimes overly restrictive CFL condition [26]. Unfortunately, these large time steps come at the cost of added dissipation. This is visually acceptable for gases such as smoke where it appears realistic. For liquids however, mass dissipation ruins the visual effect. Therefore, even though we use a semi-Lagrangian method to update (3.2), the time step for evolving the particles and the level set still needs to be limited according to a plausible CFL condition. Updating the surface position isn't particularly expensive computationally, and so we alternate between a large time step for updating the Navier-Stokes equations and a series of small time steps (that add up to the large time step) for the particles and the level set. Our experience suggests that the velocity field time step can only be a few (around five) times bigger than that dictated by the usual CFL criterion. However, even this gives tremendous computational savings, since enforcing incompressibility (step V, discussed in section 8) is the most expensive part of the algorithm.

We caution the reader that using a particle only evolution with the semi-Lagrangian method introduces noise into the surface, and that using a level set only evolution with the semi-Lagrangian method gives noticeable volume loss. The key to making the semi-Lagrangian method work for liquids is the mixed Eulerian-Lagrangian formulation that uses *both* particles and level sets to evolve the surface position over time.

7. Boundary Conditions

When solving (3.2) within the grid, we need to specify pressure and velocity values for certain cells. We want stationary object cells to resist liquid motion and cells that represent the boundary between air and liquid to behave appropriately.

7.1.1 Non-liquid Cells

Cells in the grid that don't contain particles and aren't contained within the isosurface are either considered empty (open air) or are part of an object. If a cell is empty, its pressure is set to atmospheric pressure, and the velocity on each of its faces shared with another empty cell is set to zero. This assumes that air dynamics has a negligible effect. An object cell, on the other hand, can have velocities and pressures set using many different combinations to approximate liquid flowing into or out of the environment, or to approximate different object material properties. Foster and Metaxas [10] summarize and discuss methods to do this.

7.1.2 Liquid Surface

Other grid cells that require special attention are those that contain part of the $\phi(\mathbf{x})=0$ isocontour. Such cells represent what we know about the location of the liquid surface within the grid. The movement of the isocontour will determine how the surface evolves, but we need to set velocities on faces between empty and liquid cells so that normal and tangential stresses are zero. Intuitively, we need to make sure that the "air" doesn't mix with or inhibit the motion of the liquid, while allowing it to flow freely into empty cells. This is done by explicitly enforcing incompressibility within each cell that contains part of the liquid surface. Velocities adjacent to a liquid filled cell are left alone, whereas the others are set directly so (3.1) is satisfied for that cell. The pressure in a surface cell is set to atmospheric pressure.

8. Conservation of Mass

The velocity field generated after evolving the Navier-Stokes equations (steps III and IV) has rotation and convection components that are governed by (3.2) (excluding the pressure term). However, (3.1), conservation of mass, is only satisfied in surface cells where we have explicitly enforced it. The best we can do to preserve mass within our grid is to ensure that the incompressibility condition is satisfied for every grid cell (at least to some tolerance). Foster and Metaxas [11] achieved this using a technique called Successive Over Relaxation.

A more efficient method for enforcing incompressibility comes from solving the linear system of equations given by using the Laplacian operator to couple local pressure changes to the divergence in each cell. Specifically, this gives

$$\nabla^2 p = \rho \nabla \cdot \mathbf{u} / \Delta t \,, \qquad (8.1)$$

where $\nabla^2 p$ is the spatial variation (Laplacian) of the pressure and $\mathbf{u}$ is the velocity field obtained after solving (3.2). Applied at the center of a cell, (8.1) can be discretized as

$$\sum_{n=\{ijk\}} (p_{n+1} + p_{n-1}) - 6p = \rho \frac{\Delta \tau}{\Delta t} \sum_{n=\{ijk\}} (u_{n+1} - u_n), \qquad (8.2)$$

where $p_{n \pm 1}$ is the pressure from the cell ahead (+) or behind (-) in the n direction, and the u values are taken directly from the grid cell faces. Using (8.2), we form a linear system AP = b where P is the vector of unknown pressures needed to make the velocity field divergence free, b is the RHS of (8.2), and A has a regular but sparse structure. The diagonal coefficients of A, a_{ii}, are equal to the negative number of liquid cells adjacent to $cell_i$ (e.g., -6 for a fully "submerged" cell) while the off diagonal elements are simply $a_{ij}=a_{ji}=1$ for all liquid $cells_j$ adjacent to $cell_i$.

Conveniently, the system described above is symmetric and positive definite (as long as there is at least one surface cell as part of each volume). Static object and empty cells don't disrupt this structure. In that case pressure and velocity terms can disappear from both sides of (8.2), but the system remains symmetric. Because of this, it can be solved quickly and efficiently using a Preconditioned Conjugate Gradient (PCG) method. Further efficiency gains can be made by using an Incomplete Choleski preconditioner to accelerate convergence. There is a wealth of literature available regarding PCG techniques and we recommend any of the standard implementations, see Barret et al. [2] for some basic templates. Once the new pressures have been determined, the velocities in each cell are updated according to

$$u_{\{ijk\}}^{t+\Delta t} = u_{\{ijk\}} - \frac{\Delta t}{\rho \Delta \tau} (p_n - p_{n-1})$$

The resulting velocity field conserves mass (is divergence free) and satisfies the Navier-Stokes equations.

9. Moving Objects

Previous techniques proposed for liquid animation could deal with static objects that have roughly the same resolution as the grid, but they have difficulty dealing with moving objects. Unfortunately, the CFD literature has little to offer to help resolve the effects of moving objects on a liquid in terms of animation. There are sophisticated methods available for handling such interactions, but they typically require highly resolved computational grids or a grid mechanism that can adapt itself to the moving object surface. Neither approach is particularly well suited to animation because of the additional time complexity involved. Therefore, we propose the following method for handling interactions between moving objects and the liquid.

Consider an object (or part of an object) moving within a cell that contains liquid. There are two basic conditions that we want to enforce with respect to the computational grid, and an additional condition with respect to the surface tracking method. These are

1. Liquid should not flow into the object. At any point of contact, the relative velocity between the liquid and object along the object's surface normal should be greater than or equal to zero.
2. Tangential to the surface, the liquid should flow freely without interference.
3. Neither the particles nor the level set surface should pass through any part of the surface of the object.

The last of these is relatively straightforward. We know where the polygons that comprise the object surface are and in what direction they are moving. We simply move the particles so that they are always outside the surface of the object. As long as we accurately take account of the velocity field within the grid then the isocontour will remain in the correct position relative to the object.

To prevent liquid from flowing into the object we directly set the component of liquid velocity normal to the object. We know the object surface normal, $\mathbf{n}_s$, and can calculate the liquid velocity relative to that surface, $\mathbf{v}_r$, in a given cell. If $\mathbf{v}_r.\mathbf{n}_s < 0$ then liquid is flowing through the surface. In such cases we manipulate $\mathbf{u}$ in the cell so that $\mathbf{v}_r.\mathbf{n}_s = 0$ leaving the tangential ("slip") part of the velocity unchanged.

These velocities need to be applied without introducing visual artifacts into the flow. The following method solves for both normal and tangential velocity components. It's relatively intuitive, and it seems to work well in practice. The steps are

1. As a boundary condition, any cell within a solid object has its velocities set to that of the moving object.
2. The velocity field is updated using (3.2). No special consideration is given to cells containing an object, i.e. they are all allowed to change freely as if they contain liquid.
3. Each cell that intersects an object surface gets the component of the object velocity along its normal set explicitly as outlined above.
4. Cells internal to the object have their velocities set back to the object velocity.
5. During the mass conservation step (section 8) the velocity for any grid cell that intersects the object is held fixed.

The result of this approach is that liquid is both pushed along by an object while being allowed to flow freely around it, causing realistic-looking behavior in the mean time. Obviously it's only possible to accurately account for one polygon face per grid cell. Objects that are more detailed with respect to the grid can still be handled by determining an average object surface normal and velocity for each intersecting cell, but grid resolution remains the limiting factor.

10. Control

Animation is all about control. Having things behave according to some arbitrary aesthetic is the goal of most production software. The difficulty is in providing this level of control over a physics-based animation while still maintaining a realistic behavioral component. The nature of the governing equations of motion of liquids means that they will always swirl, mix, and splash unless the applied forces are identical everywhere. This necessarily limits the level of control that we can have over the final motion and comes with the territory of non-linear simulation.

Gates [13] has shown that mass conserving flow fields can be blended with calculated fields to get good non-dynamic results. The Navier-Stokes equations allow for the body force term, $\mathbf{g}$, to be manipulated directly [9] much like a traditional particle system. Forces aren't always a very intuitive way of getting motion that we want however. The moving object mechanism on the other hand, is well suited to this. Instead of moving polygons, we can explicitly set velocities anywhere in the grid by introducing "fake" surfaces (a single point even) that have normals and velocities pointing in the direction that we want the liquid to go. Setting the normal *and* tangential velocities in individual cells is also possible if it is done before the mass conservation calculation. This allows the solver to smooth out any lack of physical correctness in applied velocities before passing them into (3.2).

As a brief example, consider a set of 3D parametric space curves that define the desired path for the liquid to follow. We instance a set of points along each curve giving each point a parametric coordinate φ_p. A point's spatial position is then given simply by the curve definition, i.e. $\mathbf{x}_p=F(\varphi_p)$. The velocity of the point can then be described as

$$\mathbf{v}_p = C(t)\left(dF(\varphi_p)/d\varphi_p\right)$$

where $C(t)$ is a monotonic key framed scaling function. $C(t)$ is also used to update φ_p over time according to $d\varphi_p/dt = C(t)$. The "fake" surface normal of the point is then simply $\mathbf{n}_p = \mathbf{v}_p/|\mathbf{v}_p|$. By manipulating $\mathbf{x}_p$, $\mathbf{v}_p$, and $\mathbf{n}_p$ over time, we can "trap" small pockets of liquid and control them directly. The governing equations then make sure that neighboring liquid attempts to follow along.

This basic approach can be adapted to surfaces or even volumetric functions as long as they vary smoothly. While still not giving perfect direct control over the liquid motion, when combined with force fields it is good enough to make it a useful animation tool.

11. Results

The animation system described in the preceding sections was used to generate all of the examples in this paper. The basic Navier-Stokes solver and implicit surface are demonstrated by the container-filling example in figure 4. The combination of particles and level set make sure that the resulting surface stays smooth and behaves in a physically believable way. The splashing object examples in figures 1, 5 and 6 show close interaction between the liquid and moving objects. They also show how the hybrid surface can handle extreme splashing without either the particles or level set being apparent. The particles play a large role in both cases by allowing the liquid to "splash" at a higher resolution than would be possible with the level set alone. All of these images were rendered using a ray-tracing algorithm that marches through the implicit surface grid as outlined in section 5.

The final example, figure 7, makes use of just a spherical implicit function around each particle. It shows the interaction between a thick (high viscosity) liquid and a hand animated character. The character surface is sampled at each grid cell and the mechanisms described in section 9 take account of all the motion in the scene. This includes the character filling his mouth with mud. The mud is later ejected using a 3D space curve as a controller as outlined in section 10. The captions to each figure give the static grid size used during calculation along with computation times per frame (for motion, not rendering) on a PentiumII 500MHz.

12. Conclusion

We have presented a method for modeling and animating liquids that is a pragmatic compromise between the numerical care that needs to be taken when simulating non-linear physics and the interaction and control animators require. Where appropriate, we have drawn on techniques from computational fluid dynamics and combined them with recent computer graphics advances as well

as new methods for free surface evolution and interaction between moving objects and the liquid volume. The result is a technique that is very general, efficient, and offers flexible control mechanisms for specifying how the liquid should behave.

13. Acknowledgements

The work of the second author was supported in part by ONR N00014-97-1-0027.

A. Courant-Friedrichs-Levy (CFL) Condition

The CFL condition is a restriction on the size of the time step, Δt, that can be used together with a time-marching numerical simulation. It says that Δt must be less than the minimum time over which "something significant" can occur in the system for the simulation to remain numerically stable. The CFL condition depends both on the physical system being modeled as well as the specifics of the discretization method employed. In the context of the system described in this paper a good CFL condition is that a discrete element of liquid cannot "jump over" a cell in the computational grid, i.e. $\Delta t < \Delta\tau / |\mathbf{u}|$.

Note that the viscosity related terms also impose a CFL type restriction. This can be avoided by locally adjusting the magnitude of the viscosity in cells where the viscous terms would dictate the necessity for a smaller time step.

References

[1] Abbot, M. and Basco, D., "Computational Fluid Dynamics – An Introduction for Engineers", Longman, 1989.

[2] Barrett, R., Berry, M., Chan, T., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C. and van der Vorst, H., "Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods", Society for Industrial and Applied Mathematics, 1993.

[3] Bloomenthal, J., Bajaj, C., Blinn, J., Cani-Gascual, M.-P., Rockwood, A., Wyvill, B. and Wyvill, G., "Introduction to Implicit Surfaces", Morgan Kaufmann Publishers Inc., San Francisco, 1997.

[4] Chen, J. and Lobo, N., "Toward Interactive-Rate Simulation of Fluids with Moving Obstacles Using the Navier-Stokes Equations", Graphical Models and Image Processing 57, 107-116 (1994).

[5] Chen, S., Johnson, D., Raad, P. and Fadda, D., "The Surface Marker and Micro Cell Method", Int. J. Numer. Methods in Fluids 25, 749-778 (1997).

[6] Courant, R., Issacson, E. and Rees, M., "On the Solution of Nonlinear Hyperbolic Differential Equations by Finite Differences", Comm. Pure and Applied Math 5, 243-255 (1952).

[7] Desbrun, M. and Cani-Gascuel, M.P., "Active Implicit Surface for Animation", Graphics Interface 98, 143-150 (1998).

[8] Fedkiw, R., Aslam, T., Merriman, B. and Osher, S., "A Non-Oscillatory Eulerian Approach to Interfaces in Multimaterial Flows (The Ghost Fluid Method)", J. Comput. Phys. 152, 457-492 (1999).

[9] Foster, N. and Metaxas, D., "Controlling Fluid Animation", Computer Graphics International 97, 178-188 (1997).

[10] Foster, N. and Metaxas, D., "Modeling the Motion of a Hot Turbulent Gas", ACM SIGGRAPH 97, 181-188 (1997).

[11] Foster, N. and Metaxas, D., "Realistic Animation of Liquids", Graphical Models and Image Processing 58, 471-483 (1996).

[12] Fournier, A. and Reeves, W.T., "A Simple Model of Ocean Waves", ACM SIGGRAPH 86, 75-84 (1986).

[13] Gates, W.F., "Interactive Flow Field Modeling for the Design and Control of Fluid Motion in Computer Animation", UBC CS Master's Thesis, 1994.

[14] Golub, G.H. and Van Loan, C.F., "Matrix Computations", The John Hopkins University Press, 1996.

[15] Harlow, F.H. and Welch, J.E., "Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with a Free Surface", The Physics of Fluids 8, 2182-2189 (1965).

[16] Kang, M., Fedkiw, R. and Liu, X.-D., "A Boundary Condition Capturing Method For Multiphase Incompressible Flow", J. Sci. Comput. 15, 323-360 (2000).

[17] Kass, M. and Miller, G., "Rapid, Stable Fluid Dynamics for Computer Graphics", ACM SIGGRAPH 90, 49-57 (1990).

[18] Lorenson, W.E. and Cline, H.E., "Marching Cubes: A High Resolution 3D Surface Construction Algorithm", Computer Graphics 21, 163-169 (1987).

[19] Miller, G. and Pearce, A., "Globular Dynamics: A Connected Particle System for Animating Viscous Fluids", Computers and Graphics 13, 305-309 (1989).

[20] O'Brien, J. and Hodgins, J., "Dynamic Simulation of Splashing Fluids", Computer Animation 95, 198-205 (1995).

[21] Osher, S. and Sethian, J.A., "Fronts Propagating with Curvature Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations", J. Comput. Phys. 79, 12-49 (1988).

[22] Peachy, D., "Modeling Waves and Surf", ACM SIGGRAPH 86, 65-74 (1986).

[23] Schachter, B., "Long Crested Wave Models", Computer Graphics and Image Processing 12, 187-201 (1980).

[24] Sethian, J.A. "Level Set Methods and Fast Marching Methods", Cambridge University Press, Cambridge 1999.

[25] Stam, J., "Stable Fluids", ACM SIGGRAPH 99, 121-128 (1999).

[26] Staniforth, A. and Cote, J., "Semi-Lagrangian Integration Schemes for Atmospheric Models – A Review", Monthly Weather Review 119, 2206-2223 (1991).

[27] Terzopoulos, D., Platt, J. and Fleischer, K., "Heating and Melting Deformable Models (From Goop to Glop)", Graphics Interface 89, 219-226 (1995).

[28] Yngve, G., O'Brien, J. and Hodgins, J., "Animating Explosions", ACM SIGGRAPH 00, 29-36 (2000).

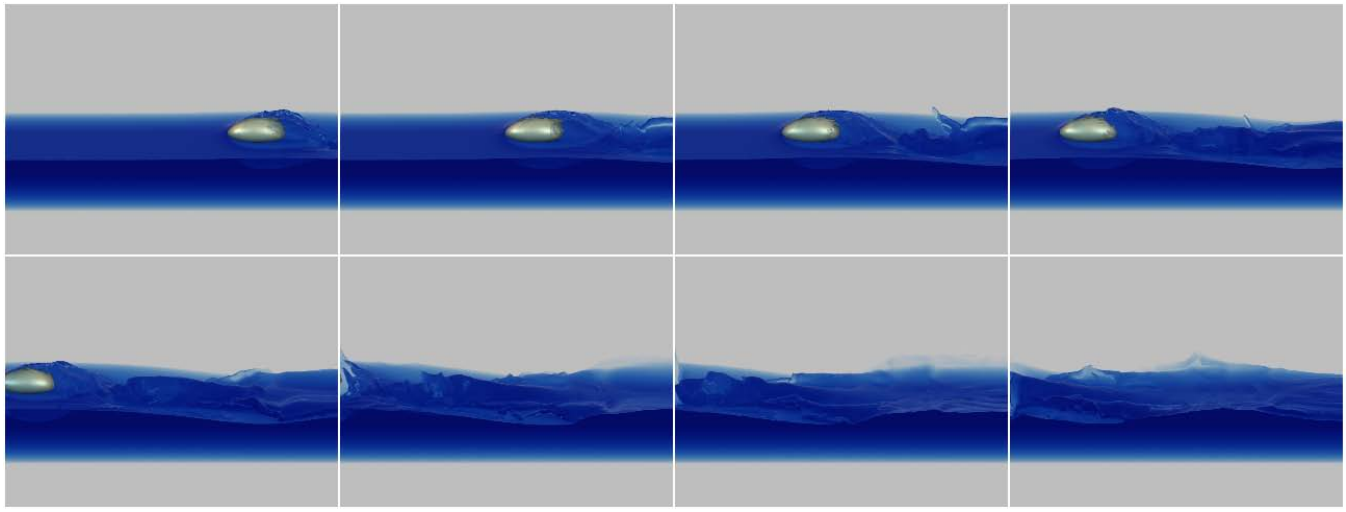


Figure 5: *An ellipsoid slips along through shallow water. The combination of particle and level set tracking allows water to flow over the object without any visual loss of volume. The environment for this example was 250x75x90 cells. It took approximately seven minutes to calculate the liquid motion (including surface evolution) per frame.*

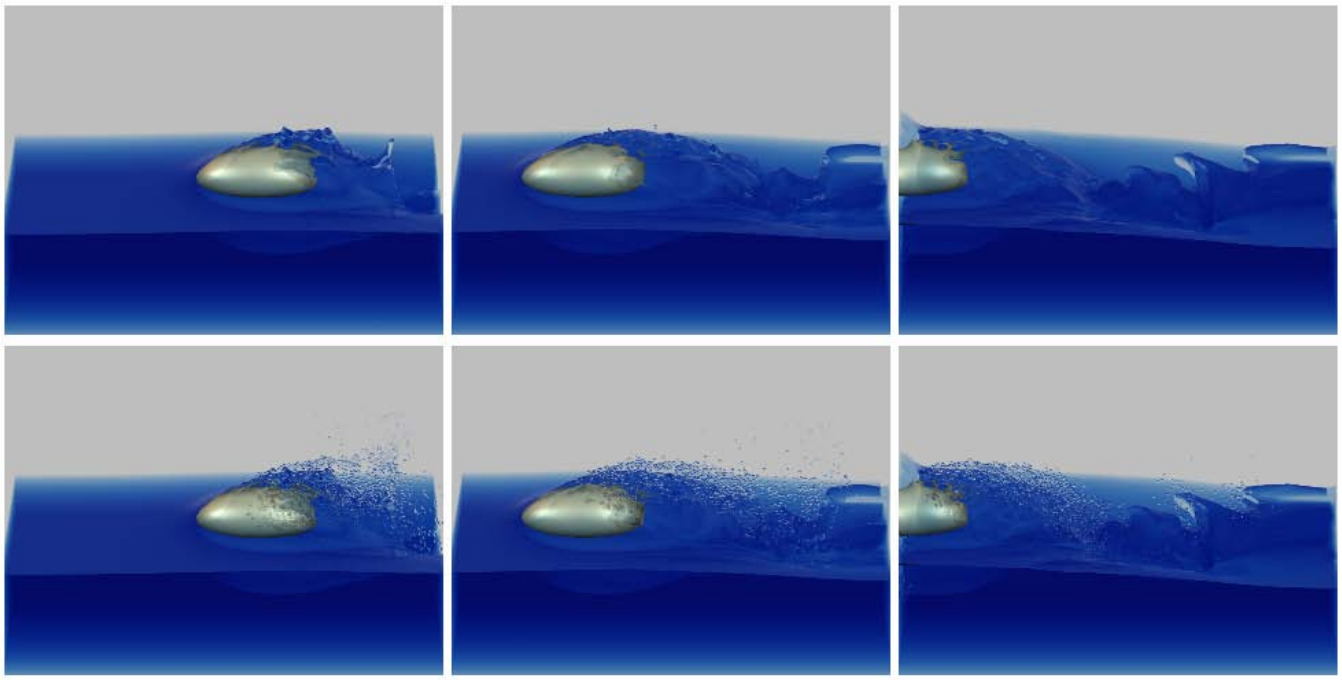


Figure 6: *A close up of the ellipsoid from figure 5 showing the implicit surface derived from combining the particle basis functions and level set (top), and with the addition of the freely splashing particles raytraced as small spheres (bottom). The environment for this example was 150x75x90 cells. Calculation times were approximately four minutes per frame.*

Figure 7: *A fully articulated animated character interacts with viscous mud. The environment surrounding the character is 150x200x150 cells. That resolution is sufficient to accurately model the character filling his mouth with mud. A 3D control curve is used to eject (spit) the mouthful of mud later in the sequence. This example runs at three minutes per frame.*

Animation and Rendering of Complex Water Surfaces

Douglas Enright
Stanford University
Industrial Light & Magic
enright@stanford.edu

Stephen Marschner
Stanford University
srm@graphics.stanford.edu

Ronald Fedkiw
Stanford University
Industrial Light & Magic
fedkiw@cs.stanford.edu

Abstract

We present a new method for the animation and rendering of *photorealistic* water effects. Our method is designed to produce visually plausible three dimensional effects, for example the pouring of water into a glass (see figure 1) and the breaking of an ocean wave, in a manner which can used in a computer animation environment. In order to better obtain photorealism in the behavior of the simulated water surface, we introduce a new "thickened" front tracking technique to accurately represent the water surface and a new velocity extrapolation method to move the surface in a smooth, water-like manner. The velocity extrapolation method allows us to provide a degree of control to the surface motion, e.g. to generate a wind-blown look or to force the water to settle quickly. To ensure that the photorealism of the simulation carries over to the final images, we have integrated our method with an advanced physically based rendering system.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing;

Keywords: computational fluid dynamics, implicit surfaces, natural phenomena, physically based animation, rendering, volume rendering

1 Introduction

Water surrounds us in our everyday lives. Given the ubiquity of water and our constant interaction with it, the animation and rendering of water poses one of the greatest challenges in computer graphics. The difficulty of this challenge was underscored recently through the use of water effects in a variety of motion pictures including the recent feature film "Shrek" where water, mud, beer and milk effects were seen. In response to a question concerning what was the single hardest shot in "Shrek", DreamWorks SKG principal and producer of "Shrek", Jeffrey Katzenberg, stated, *It's the pouring of milk into a glass.* [Hiltzik and Pham 2001].

The above quote illustrates the need for *photorealistic* simulation and rendering of water (and other liquids such as milk), especially in the case of complex, three dimensional behavior as seen when water is poured into a glass as in figure 1. A key to achieving this goal is the visually accurate treatment of the surface separating the water from the air. The behavior of this surface provides the visual impression that water is being seen. If the numerical simulation method used to model this surface is not robust enough to capture the essence of water, then the effect is ruined for the viewer. A variety of new techniques for treatment of the surface are proposed in this paper in order to provide visually pleasing motion and photorealistic results.

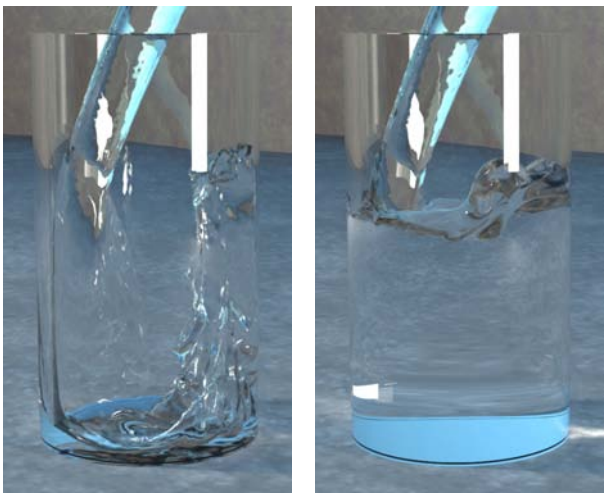


Figure 1: **Water being poured into a glass (55x120x55 grid cells).**

We propose a new "thickened" front tracking approach to modeling the surface, called the "particle level set method". It is a hybrid surface tracking method that uses massless marker particles combined with a dynamic implicit surface. This method was entirely inspired by the hybrid liquid volume model proposed in [Foster and Fedkiw 2001], but exhibits significantly more realistic surface behavior. This effect is achieved by focusing on modeling the surface as opposed to the liquid volume as was done by [Foster and Fedkiw 2001]. This shift in philosophy away from volume modeling and towards surface modeling, is the key idea behind our new techniques, resulting in better photorealistic behavior of the water surface.

We propose a new treatment of the velocity at the surface in order to obtain more visually realistic water surface behavior. The motion of both the massless marker particles and the implicit function representing the surface is dependent upon the velocities contained on the underlying computational mesh. By extrapolating velocities across the water surface and into the region occupied by the air, we obtain more accurate and better visual results. In the limit as the computational grid is refined, the resulting surface condition is identical to the traditional approach of making the velocity divergence free, but it gives more visually appealing and physically plausible results on coarse grids. Furthermore, this velocity extrapolation procedure alows us to add a degree of control to the behavior of the water surface. We can add dampening and/or churning effects forcing it to quiet down or splash up faster than would be

allowed by a straightforward physical simulation.

Our new advances can be easily incorporated into a pre-existing Navier-Stokes solver for water. In fact, we solve the Navier-Stokes equations for liquid water along the lines of [Foster and Fedkiw 2001], in particular using a semi-Lagrangian "stable fluid" approach introduced to the community by Stam [Stam 1999]. Our approach preserves as much of the realistic behavior of the water as possible, while at the same time providing a degree of control necessary for its use in a computer animation environment.

Photorealistic rendering is necessary in order to complete the computational illusion of real water. In some ways water is an easy material to render, because unlike many common materials its optical properties are well understood and are easy to describe. In all but the largest-scale scenes, surface tension prevents water surfaces from exhibiting the microscopic features that make reflection from many other materials so complicated. However, water invariably creates situations in which objects are illuminated through complex refracting surfaces, which means that the light transport problem that is so easy to state is difficult to solve. Most widely used rendering algorithms disregard this sort of illumination or handle it using simple approximations, but because water and its illumination effects are so familiar these approaches fail to achieve realism. There are several rendering algorithms that can properly account for all transport paths, including the illumination through the water surface; some examples are path tracing [Kajiya 1986], bidirectional path tracing [Heckbert 1990; Lafortune and Willems 1993; Veach and Guibas 1994], Metropolis light transport [Veach and Guibas 1997], and photon mapping [Jensen 1995]. In our renderings of clear water for this paper we have chosen photon mapping because it is simple and it makes it easy to avoid the distracting noise that often arises in pure path sampling algorithms from illumination through refracting surfaces.

2 Previous Work

Early (and continuing) work by the graphics community into the modeling of water phenomenon focused on reduced model representations of the water surface, ranging from Fourier synthesis methods [Masten et al. 1987] to parametric representations of the water surface [Schachter 1980; Fournier and Reeves 1986; Peachey 1986; Ts'o and Barsky 1987]. The last three references are notable in the way they attempt to model realistic wave behavior including the change in wave behavior as a function of the depth of the water. Fairly realistic two dimensional wave scenery can be developed using these methods including the *illusion* of breaking waves, but ultimately they are all constrained by the sinusoidal modeling assumption present in each of them. They are unable to easily deal with complex three dimensional behaviors such as flow around objects and dynamically changing boundaries. A summary of the above methods and their application to modeling and rendering of ocean wave environments can be found in [Tessendorf 2001].

In order to obtain water models which could potentially be used in a dynamic animation environment, researchers turned towards using two dimensional approximations to the full 3D Navier-Stokes equations. [Kass and Miller 1990] use a linearized form of the 2D shallow water equations to obtain a height field representation of the water surface. A pressure defined height field formulation was used by [Chen and Lobo 1994] in fluid simulations with moving obstacles. [O'Brien and Hodgins 1995] used a height model combined with a particle system in order to simulate splashing liquids. The use of a height field gives a three dimensional look to a two dimensional flow calculation, but it constrains the surface to be a function, i.e. the surface passes the vertical line test where for each (x,y) position there is at most one z value. The surface of a crashing wave or of water being poured into a glass does not satisfy the vertical line test. Use of particle systems permits the surface to become multivalued. A viscous spring particle representation of a liquid has been proposed by [Miller and Pearce 1989]. An alternative molecular dynamics approach to the simulation of particles in the liquid phase has been developed by [Terzopoulos et al. 1989]. Particle methods, while quite versatile, can pose difficulties when trying to reconstruct a smooth water surface from the locations of the particles alone.

The simulation of complex water effects using the full 3D Navier-Stokes equations has been based upon the large amount of research done by the computational fluid dynamics community over the past 50 years. [Foster and Metaxas 1996] utilized the work of [Harlow and Welch 1965] in developing a 3D Navier-Stokes methodology for the realistic animation of liquids. Further CFD enhancements to the traditional marker and cell method of Harlow and Welch which allow one to place particles only near the surface can be found in [Chen et al. 1997]. A semi-Lagrangian "stable fluids" treatment of the convection portion of the Navier-Stokes equations was introduced to the computer graphics community by [Stam 1999] in order to allow the use of significantly larger time steps without hindering stability. [Foster and Fedkiw 2001] made significant contributions to the simulation and control of three dimensional fluid simulations through the introduction of a hybrid liquid volume model combining implicit surfaces and massless marker particles; the formulation of plausible boundary conditions for moving objects in a liquid; the use of an efficient iterative method to solve for the pressure; and a time step subcycling scheme for the particle and implicit surface evolution equations in order to reduce the amount of visual error inherent to the large semi-Lagrangian "stable fluid" time step used for time evolving the fluid velocity and the pressure. The combination of all of the above advances in 3D fluid simulation technology along with ever increasing computational resources has set the stage for the inclusion of fully 3D fluid animation tools in a production environment.

Most work on simulating water at small scales has not specifically addressed rendering and has not used methods that correctly account for all significant light transport paths. Research on the rendering of illumination through water [Watt 1990; Nishita and Nakamae 1994] has used methods based on processing each polygon of a mesh that represents a fairly smooth water surface, so these methods cannot be used for the very complex implicit surfaces that result from our simulations. For the case of 2D wave fields in the open ocean, approaches motivated by physical correctness have produced excellent results [Premoze and Ashikhmin 2000; Tessendorf 2001].

3 Simulation Method

3.1 Motivation

[Foster and Fedkiw 2001] chose to define the liquid volume being simulated as one side of an isocontour of an implicit function, ϕ. The surface of the water was defined by the $\phi = 0$ isocontour with $\phi \leq 0$ representing the water and $\phi > 0$ representing the air. By using an implicit function representation of the liquid volume, they obtained a smooth, temporally coherent liquid surface. They rejected the use of particles alone to represent the liquid surface because it is difficult to a calculate a visually desirable smooth liquid surface from the discrete particles alone. The implicit surface was dynamically evolved in space and time according to the underlying liquid velocity $\vec{u}$. As shown in [Osher and Sethian 1988], the appropriate equation to do this is

$$\phi_t + \vec{u} \cdot \nabla \phi = 0 \tag{1}$$

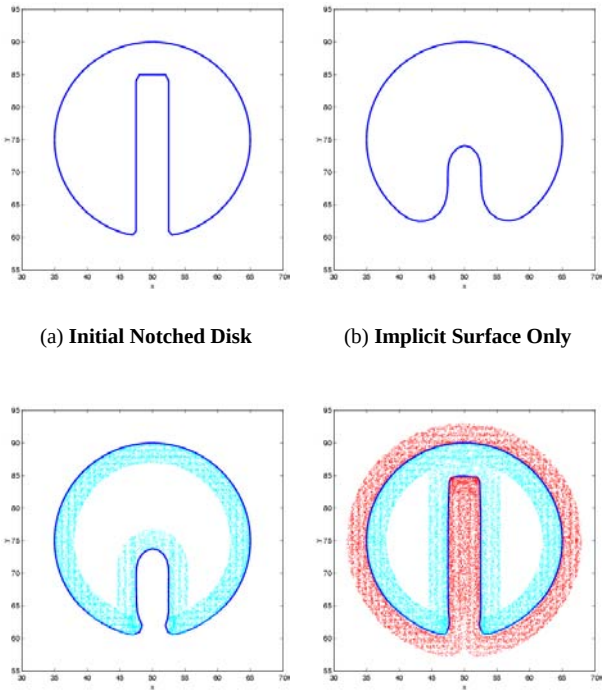
where ϕ_t is the partial derivative of ϕ with respect to time and $\nabla = (\partial/\partial x, \partial/\partial y, \partial/\partial z)$ is the gradient operator.

An implicit function only approach to modeling the surface will not yield realistic surface behavior due to an excessive amount of volume loss on coarse grids. A seminal advance of [Foster and Fedkiw 2001] in creating realistic liquids for computer animation is the hybridization of the visually pleasing smooth implicit function modeling of the liquid volume with particles that can maintain the liquid volume on coarse grids. The inclusion of particles provides a way for capturing the liveliness of a real liquid with spray and splashing effects. Curvature was used as an indicator for allowing particles to influence the implicit surface representation of the water. This is a natural choice since small droplets of water have very high curvature and dynamic implict surfaces have difficulty resolving features with sharp corners.

Figure 2(a) shows a notched disk that we rotate for one rigid body rotation about the point (50,50). The inside of the disk can be thought of as a volume of liquid. Figure 2(b) shows the result of using an implicit surface only approach (after one rotation) where both the higher and lower corners of the disk are erroneously shaved off causing both loss of visual features and an artificially viscous look for the liquid. This numerical result was obtained using a highly accurate fifth order WENO discretization of equation 1 (see e.g. [Jiang and Peng 2000; Osher and Fedkiw 2002]). For the sake of comparison, we note that [Sethian 1999] only proposes second order accurate methods for discretizing this equation. Figure 2(c) shows the result obtained with our implementation of the method from [Foster and Fedkiw 2001]. The particles inside the disk do not allow the implicit surface to cross over them and help to preserve the two corners near the bottom. However, there is little they can do to stop the implicit surface from drifting away from them near the top corners. This represents loss of air or bubbles as the method erroneously gains liquid volume. This is not desirable since many complex water motions such as wave phenomenon are due in part to differing heights of water columns adjacent to each other. Loss of air in a water column reduces the pressure forces from neighboring columns destroying many of the dynamic splashing effects as well as the overall visually stimulating liveliness of the liquid.

While the hybrid liquid volume model of Foster and Fedkiw attempts to maintain the volume of the liquid accurately, it fails to model the air or more generally the opposite side of the liquid surface. We shift the focus away from maintaining a liquid volume towards maintaining the liquid surface itself. An immediate advantage of this approach is that it leads to symmetry in particle placement. We place particles on both sides of the surface and use them to maintain an accurate representation of the surface itself regardless of what may be on one side or the other. The particles are not meant to track volume, they are intended to correct errors in the surface representation by the implicit function. In [Enright et al. 2002] we showed that this surface only approach leads to the most accurate 3D results for surface tracking ever achieved (in both CFD and CG). This was done for analytical "test" velocity fields. Figure 2(d) shows that this new method correctly computes the rigid body rotation for the notched disk preserving both the water and the air volumes so that more realistic water motion can be obtained.

In this current paper, we couple this new method to real velocity fields and fluid dynamics calculations for the first time. Representative results of this new method can be seen in figure 3. A ball is thrown into a tank of water with the same tank geometry, grid spacing and ball speed as seen in figure 4 (courtesy of [Foster and Fedkiw 2001]). The resulting splash after the ball impacts the surface of the water is dramatically different between the two figures. Our new method produces the well formed, thin sheet one would visually expect. Note that the distorted look of the ball in our figure is due to the correct calculation of the refraction of light when it passes through the surface of the water. To give an indication of the additional computational cost incurred using our new method,



(a) **Initial Notched Disk** (b) **Implicit Surface Only**

(c) **Particles Inside Only** (d) **Our New Method**

Figure 2: **Rigid Body Rotation Test**

figure 3 took about 11 minutes per frame, figure 4 took about 7 minutes per frame and a level set only solution takes about 3 minutes per frame.

3.2 Particle Level Set Method

3.2.1 Initialization of Particles

Two sets of particles are randomly placed in a "thickened" surface region (we use three grid cells on each side of the surface) with *positive* particles in the $\phi > 0$ region and *negative* particles in the $\phi \leq 0$ region. There is no need to place particles far away from the surface since the sign of the level set function readily identifies these regions gaining large computational savings. The number of particles placed in each cell is an adjustable parameter that can be used to control the amount of resolution, e.g. we use 64 partlces per cell for most of our examples. Each particle possesses a radius, r_p, which is constrained to take a minimum and maximum value based upon the size of the underlying computational cells used in the simulation. A minimum radius of $.1\min(\Delta x, \Delta y, \Delta z)$ and maximum radius of $.5\min(\Delta x, \Delta y, \Delta z)$ appear to work well. The radius of a particle changes dynamically throughout the simulation, since a particle's location relative to the surface changes. The radius is set according to:

$$r_p = \begin{cases} r_{max} & \text{if } s_p\phi(\vec{x}_p) > r_{max} \\ s_p\phi(\vec{x}_p) & \text{if } r_{min} \leq s_p\phi(\vec{x}_p) \leq r_{max} \\ r_{min} & \text{if } s_p\phi(\vec{x}_p) < r_{min} \end{cases}, \qquad (2)$$

where s_p is the sign of the particle (+1 for positive particles and -1 for negative particles). This radius adjustment keeps the boundary of the spherical particle tangent to the surface whenever possible. This fact combined with the overlapping nature of the particle spheres allows for an enhanced reconstruction capability of the liquid surface.

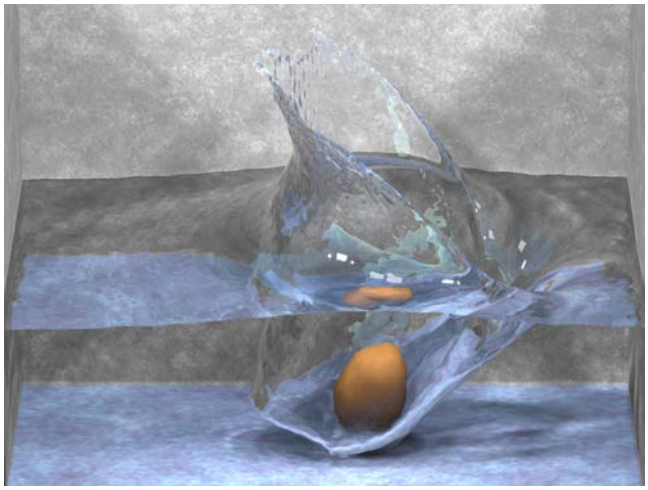


Figure 3: **Our New Method (140x110x90 grid cells).**

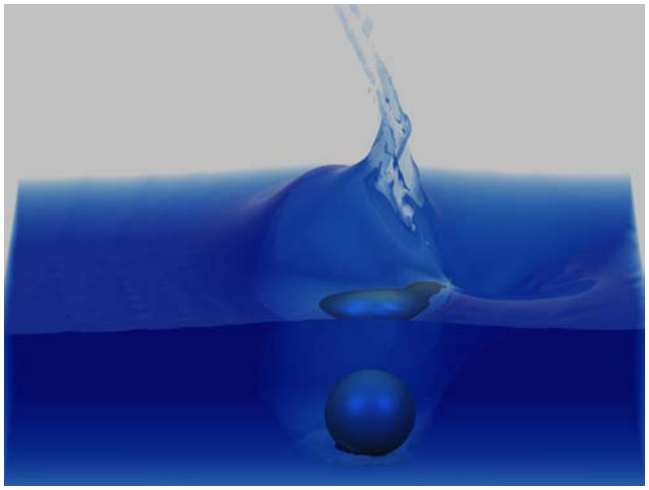


Figure 4: **Foster and Fedkiw 2001 (140x110x90 grid cells).**

3.2.2 Time Integration

The marker particles and the implicit function are separately integrated forward in time using a forward Euler time integration scheme. The implicit function is integrated forward using equation 1, while the particles are passively advected with the flow using $d\vec{x}_p/dt = \vec{u}_p$, where $\vec{u}_p$ is the fluid velocity interpolated to the particle position $\vec{x}_p$.

3.2.3 Error Correction of the Implicit Surface

Identification of Error: The main role of the particles is to detect when the implicit surface has suffered inaccuracies due to the coarseness of the computational grid in regions with sharp features. Particles that are on the wrong side of the interface by more than their radius, as determined by a locally interpolated value of ϕ at the particle position $\vec{x}_p$, are considered to have *escaped* their side of the interface. This indicates errors in the implicit surface representation. In smooth, well resolved regions of the interface, our dynamic implicit surface is highly accurate and particles do not drift a non-trivial distance across the interface.

Quantification of Error: We associate a spherical implicit function , designated ϕ_p, with each particle p whose size is determined by the particle radius, i.e.

$$\phi_p(\vec{x}) = s_p(r_p - |\vec{x} - \vec{x}_p|). \tag{3}$$

Any difference in ϕ from ϕ_p indicates errors in the implicit function representation of the surface. That is, the implicit version of the surface and the particle version of the surface disagree.

Error Correction: We use escaped positive particles to rebuild the $\phi > 0$ region and escaped negative particles to rebuild the $\phi \leq 0$ region as defined by the implicit function. The reconstruction of the implicit surface occurs locally within the cell that each escaped particle currently occupies. Using equation 3, the ϕ_p values of escaped particles are calculated for the eight grid points on the boundary of the cell containing the particle. This value is compared to the current value of ϕ for each grid point and we take the smaller value (in magnitude) which is the value closest to the $\phi = 0$ isocontour defining the surface. We do this for all escaped positive and escaped negative particles. The result is an improved representation of the surface of the liquid.

3.2.4 When To Apply Error Correction

We apply the error correction method discussed above after any computational step in which ϕ has been modified in some way. This occurs when ϕ is integrated forward in time and when the implicit function is smoothed to obtain a visually pleasing surface. We smooth the implicit surface with an equation of the form

$$\phi_\tau = -S(\phi_{\tau=0})(|\nabla\phi| - 1), \tag{4}$$

where τ is a fictitious time and $S(\phi)$ is a smoothed signed distance function given by

$$S(\phi) = \frac{\phi}{\sqrt{\phi^2 + (\Delta x)^2}}. \tag{5}$$

More details on this are given in [Foster and Fedkiw 2001].

3.2.5 Particle Reseeding

In complex flows, a liquid interface can be stretched and torn in a dynamic fashion. The use of only an initial seeding of particles will not capture these effects well, as regions will form that lack a sufficient number of particles to adequately perform the error correction step. Periodically, e.g. every 20 frames, we randomly reseed particles about the "thickened" interface to avoid this dilema. This is done by randomly placing particles near the interface, and then using geometric information contained within the implicit function (e.g. the direction of the shortest possible path to the surface is given by $\vec{N} = \nabla\phi/|\nabla\phi|$) to move the particles to their respective domains, $\phi > 0$ or $\phi \leq 0$. The goal of this reseeding step is to preserve the initial particle resolution of the interface, e.g. 64 particles per cell. Thus, if a given cell has too few or too many particles, some can be added or deleted respectively.

3.2.6 A Note on Alternative Methods

If we felt that preserving the volume of the fluid was absolutely necessary in order to obtain visually pleasing fluid behavior, we would have chosen to use a volume of fluid (VOF) [Hirt and Nichols 1981] representation of the fluid. Although VOF methods explicitly conserve volume, they produce visually disturbing artifacts allowing thin liquid sheets to artificially break up and form "blobbies" and "flotsam" of liquid. Also, a visually desirable smooth fluid interface is difficult to construct when using these methods.

Another alternative is to explicitly discretize the free surface with particles and maintain a connectivity list between particles, see e.g. [Unverdi and Tryggvason 1992]. This connectivity list is difficult to maintain when parts of the free surface break apart or merge together as is often seen in complex flows of water and other liquids. Our approach avoids the especially difficult issues associated with maintaining particle connectivity information.

3.3 Velocities at the Surface

Although the Navier-Stokes equations can be used to find the velocity within the liquid volume, boundary conditions are needed for the velocity on the air side near the free surface. These boundary condition velocities are used in updating the Navier-Stokes equations, moving the surface, and moving the particles placed near the surface. The velocity at the free surface of the water can be determined through the usual enforcement of the conservation of mass (volume) constraint, $\nabla \cdot \vec{u} = 0$, where $\vec{u} = (u, v, w)$ is the velocity of the liquid. This equation allows us to determine the velocities on all the faces of computational cells that contain the $\phi = 0$ isocontour. Unfortunately, the procedure for doing this is not unique when more than one face of a cell needs a boundary condition velocity. A variety of methods have been proposed, e.g. see [Chen et al. 1995] and [Foster and Metaxas 1996].

We propose a different approach altogether, the extrapolation of the velocity across the surface into the surrounding air. As the computational grid is refined, this method is equivalent to the usual method, but it gives a smoother and more visually pleasing motion of the surface on coarser (practical sized) grids. We extrapolate the velocity out a few grid cells into the air, obtaining boundary condition velocities in a band of cells on the air side of the surface. This allows us to use higher order accurate methods and obtain better results when moving the implicit surface using equation 1 and also provides velocities for updating the position of particles on the air side of the surface. Velocity extrapolation also assists in the implementation of the semi-Lagrangian "stable fluid" method, since there are times when characteristics generated by this approach look back across the interface (a number of cells) into the air region for valid velocities.

3.3.1 Extrapolation Method

The equation modeling this extrapolation for the x component of the velocity, u, is given by

$$\frac{\partial u}{\partial \tau} = -\vec{N} \cdot \nabla u, \tag{6}$$

where $\vec{N} = (n_x, n_y, n_z)$ is a unit vector perpendicular to the implicit surface and τ is fictitious time. A similar equation holds for the v and w components of velocity field. Since we use an implicit surface to describe the fluid, $\vec{N} = \nabla\phi / |\nabla\phi|$. Fast methods exist for solving this equation in $O(n \log n)$ time, where n is the number of grid points that one needs to extrapolate over, in our case a five grid cell thick band on the air side of the interface. The fast method is based upon the observation that information in equation 6 propagates in only one direction away from the surface. This implies that we do not have to revisit previously computed values of $\vec{u}_{ext}$ (the extrapolated velocity) if we perform the calculation in the correct order. The order is determined by the value of ϕ allowing us to do an $O(n \log n)$ sorting of the points before beginning the calculation. The value of u itself is determined by enforcing the condition at steady state, namely $\nabla\phi \cdot \nabla u = 0$ where the derivatives are determined using previously calculated values of ϕ and u. From this scalar equation, a new value of u can be determined, and then we proceed to the next point corresponding to the next smallest value of ϕ, etc. Further details of this method are discussed in [Adalsteinsson and Sethian 1999].

3.3.2 Velocity Advection

The momentum portion of the Navier-Stokes equations is:

$$\vec{u}_t = -\vec{u} \cdot \nabla\vec{u} + \nu\nabla \cdot (\nabla\vec{u}) - \frac{1}{\rho}\nabla p + \vec{g}, \tag{7}$$

where ν is the kinematic viscosity of the fluid, ρ is the density of the fluid, p is the pressure, and $\vec{g}$ is an externally applied gravity field. We use the semi-Lagrangian "stable fluids" method [Stam 1999] to update the convective portion of this equation, i.e. the $\vec{u} \cdot \nabla\vec{u}$ term. This method calculates the first term on the left hand side of equation 7 by following the fluid characteristics backwards in time to determine from which computational cell the volume of fluid came, and then taking an average of the appropriate velocities there. This allows one to stably take much larger time steps than would be allowed using other time advancement schemes for velocity advection. A consequence of the now allowed large semi-Lagrangian time step is that near the surface, we might look across the interface as many as a few grid cells into the air region to find velocities. In a standard approach, valid velocities are not defined in this region. However, our velocity extrapolation technique easily handles this case ensuring that physically plausible velocities exist a few grid cells into the air region. In fact, we extrapolate the velocity the maximum distance from the surface that would be allowed during a semi-Lagrangian "stable fluids" time step guaranteeing that a smooth, physically plausible and visually appealing velocity can be found there.

3.3.3 Control

Our velocity extrapolation method enables us to apply a newly devised method for controlling the nature of the surface motion. This is done simply by modifying the extrapolated velocities on the air side of the surface. For example, to model wind-blown water as a result of air drag, we take a convex combination of the extrapolated velocities with a pre-determined wind velocity field

$$\vec{u} = (1 - \alpha)\vec{u}_{ext} + \alpha\vec{u}_{wind}, \tag{8}$$

where $\vec{u}_{ext}$ is the extrapolated velocity, $\vec{u}_{wind}$ a desired air-like velocity, and $0 \leq \alpha \leq 1$ the mixing constant. This can be applied throughtout the surface or only locally in select portions of the computational domain as desired. Note that setting $\vec{u}_{wind} = 0$ forces churning water to settle down faster with the fastest settling resulting from $\alpha = 1$. All of the figures shown in the paper used $\alpha = 0$, but we demonstrate how α can be used to force a poured glass of water to settle more quickly in the accompanying video.

3.4 Summary

We divide up the computational domain into voxels with the components of $\vec{u}$ stored on the appropriate faces and p, ρ and ϕ stored at the center of each cell. This arrangement of computational variables is the classic staggered MAC-style arrangement [Harlow and Welch 1965]. The density of a given cell is given by the value of ϕ at the center of the cell. The evolution of $\vec{u}$, p, ρ and ϕ in a given time step is performed in a series of three steps as outlined below:

1. The current surface velocity is smoothly extrapolated across the surface into the air region as discussed in section 3.3.1. Appropriate control behavior modifications to the velocity field are made.

2. The water surface and marker particles are integrated forward in time via an explicit time step subcycling method with the appropriate corrections to ϕ as described in section 3.2.3.

3. The velocity field is updated with equation 7 using the updated values for ρ. This is done by first using the semi-Lagrangian "stable fluid" method to find an estimate for the velocity. This estimate is further augmented by the viscous and forcing terms. The spatial derivatives used in calculating these terms are calculated using a standard centered second order accurate finite difference scheme. Then a system of linear equations is assembled and solved for the pressure in order to make this intermediate velocity field divergence free. The newly calculated pressure is applied as a correction to the intermediate velocity in order to fully update the water's velocity field. Interaction of the liquid with objects, walls, etc. is treated here as well. For this step, we follow exactly the method of [Foster and Fedkiw 2001] and refer the reader there for more details.

This sequence of steps is repeated until a user defined stopping point is reached. The time step for each iteration of the above steps is determined using the water's velocity to calculate an appropriate CFL condition which is approximately five times larger than the traditional CFL condition used in fluid simulations (the semi-Lagrangian "stable fluids" method allows this). Also, any viscosity related CFL restrictions are locally dealt with by reducing the viscosity in the offending cells in order to allow for our larger time step.

3.5 Rendering

Our results are rendered using a physically based Monte Carlo ray tracer capable of handling all types of illumination using photon maps and irradiance caching [Jensen 2001]. To integrate our simulation system with the renderer we implemented a geometry primitive that intersects rays with the implicit surface directly by solving for the root of the signed distance along the ray. Depending on the accuracy required by a particular scene we use either a trilinear or a tricubic filter [Marschner and Lobb 1994] to reconstruct ϕ. The normal is computed using trilinearly interpolated central differences for the trilinear surface, or simply using the derivative of the reconstructed tricubic surface.

The properties of ϕ have two advantages in the rendering context. First, intersecting a ray with the surface is much more efficient than it would be for an isosurface of a general function. The signed distance function provides a lower bound on the distance to the intersection along the ray, allowing us to take large steps when the current point is far from the surface. Second, it is easy to provide for motion blur in the standard distribution ray tracing framework. To compute the surface at an intermediate time between two frames we simply interpolate between the two signed distance volumes and use the same intersection algorithm unchanged. For the small motions that occur between frames the special properties of ϕ are not significantly compromised.

4 Results

4.1 Pouring Water Into A Glass

Figures 1 and 7 show the high degree of complexity in the water surface. Note the liveliness of the surface of the water when the water is initially being poured. The ability to maintain the visually pleasing thin sheets of water during the turbulent mixing phase is a consequence of our new method. We do not loose any of the fine detail with regards to the air pockets formed, since we model both sides of the water surface. Even though the calculation was performed on a Cartesian computational grid, the glass was shaped as a cylinder on the grid, with the grid points outside the cylinder treated as an object which does not permit the fluid to interpenetrate it. The glass was modeled as smooth and clear in order to highlight the action of the water being poured into the glass. The computational cost was approximately 8 minutes per frame.

To our knowledge, the only other complex, three dimensional simulation of a liquid being poured into a glass for computer animation is from the Gingerbread Man torture scene in the feature film "Shrek". Milk lacks the transparency of water making it difficult to clearly view the dynamic behavior of the milk surface. Also, the modeling of a thick polygonal glass with a rough surface does not provide a clear view of the milk making a direct comparison difficult.

The scene used to render our result contains just a simple cylindrical glass, the simulated water surface, and a few texture mapped polygons for the background. Illumination comes from a physically based sky model and two area sources. Because water only reflects and refracts other objects, the scene including the sky is very important to the appearance. The glass and the water together create three dielectric interfaces: glass-air, water-air, and air-water, so the inside surface of the glass has two sets of material properties depending on whether it is inside or outside the implicitly defined surface (which is easy to determine by checking the sign of ϕ). The illumination in the shadow of the glass is provided by a photon map, and illumination from the sky on diffuse surfaces in the scene is computed using Monte Carlo integration. We also note that milk would not present as difficult a global illumination problem due to its lack of transparency. Motion blur was included for these renderings.

4.2 Breaking Wave

As a second example, we have performed a breaking wave calculation in a numerical wave tank as shown in figure 8. To begin to model this pheonmenon, we needed to chose an initial condition for the wave. We chose to use the theoretical solution of a solitary wave of finite amplitude propagating without shape change [Radovitzky and Ortiz 1998]. The initial velocities u and v in the x and y directions respectively and surface height η are given by:

$$u = \sqrt{gd}\frac{H}{d}\operatorname{sech}^2\left[\sqrt{\frac{3H}{4d^3}}x\right] \quad (9)$$

$$v = \sqrt{3gd}\left(\frac{H}{d}\right)^{3/2}\frac{y}{d}\operatorname{sech}^2\left[\sqrt{\frac{3H}{4d^3}}x\right]\tanh\left[\sqrt{\frac{3H}{4d^3}}x\right] \quad (10)$$

$$\eta = d + H\operatorname{sech}^2\left[\sqrt{\frac{3H}{4d^3}}x\right], \quad (11)$$

where g is the gravitational constant 9.8 m/s. For our simulations, we set $H = 7$ and $d = 10$. We used the same initial conditions in three spatial dimensions, replicating the two dimensional initial conditions along the z-axis.

Next, we needed to introduce a model of a sloping underwater shelf in order cause the propagating pulse to actually pile up on itself and form a breaking wave. We performed a variety of two dimensional tests to determine the best underwater shelf geometry to generate a visually pleasing breaking wave. Figure 5 is a sequence of frames from one of the two dimensional tests we ran with the same x-y cross section as can be found in our 3D example. We found this prototyping technique to be a fast and easy way to explore possible breaking wave behaviors in a fraction of the time it would take to run a fully three dimensional test case.

After determining the best submerged shelf geometry to generate a breaking wave, we generated a slight tilt in the geometry in order

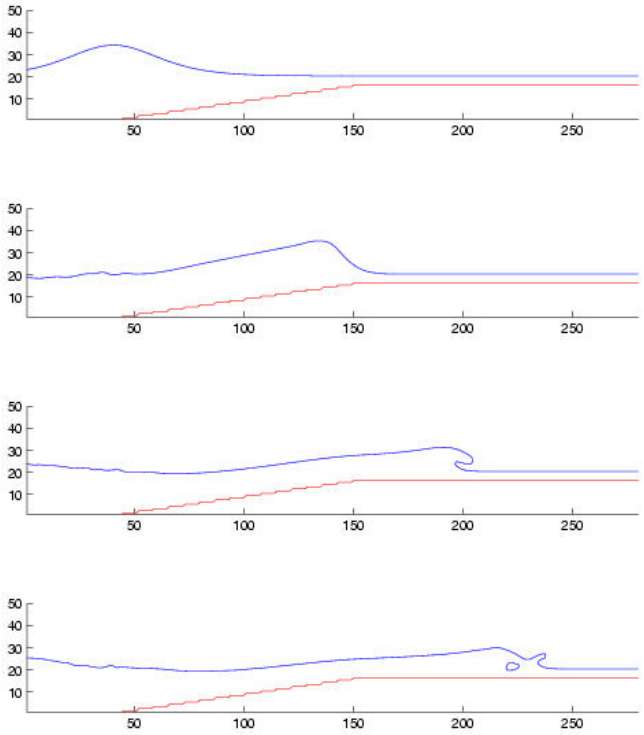


Figure 5: **Two Dimensional Breaking Wave**

to induce a curl in the break of the wave and enhance the three dimensional look. A sketch of the shelf geometry used in our three dimensional calculations is shown in figure 6. An incline of slope 1:7 rises up from the seabottom to a depth of 2 m below the surface of the water. Instead of allowing the incline to go all the way to the surface we chose to have it flatten out at this depth in order to illustrate the splash up effects after the initial breaking of the wave.

Next we ran a fully three dimensional simulation, the results of which can be seen in figures 8 and 9. The wave has the intended curling effect. The formation of a tube of air is clearly seen after the wave splashes down, with the "air" particles maintaining the tube even after the wave begins a secondary splash up. We observe some solely three dimensional effects including the fingering of the breaking wave. The computational cost was approximately 3 minutes per frame.

Because of the very different scale of this simulation as compared to the water glass, different optical effects are important. We model the water volume under the surface, which in surf is densely populated with scattering particles, as a gray diffuse reflector. The color of the water comes from a cloudy blue sky. Because the features that can be resolved are so fine compared to the grid resolution, we used cubic interpolation for this scene.

The simulation shown captures the basic phenomena of the breaking wave on a very coarse grid, but in real waves there are small-scale features that, while not important to the *behavior* of the large wave, are very important for its *appearance*. Coupling a 2D simulation for the small scale features to the wave simulation is a promising avenue for future work. Once those waves are incorporated it will become important to treat diffusion of light through the water more correctly. Also, no texture mapping was performed, e.g. use of a Philips spectrum [Tessendorf 2001], or bump mapping technique [Fournier and Reeves 1986]. Another challenge will be to augment our method to naturally handle spray and foam.

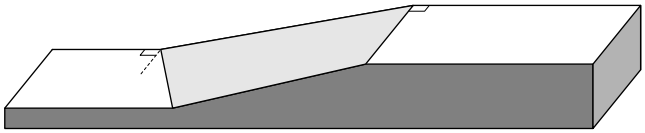


Figure 6: **Submerged Shelf**

5 Conclusion and Future Work

We have presented some novel computational methods for enhanced surface tracking and modeling of the surface motion. Combining these leads to the possibility of creating photorealistic behavior in 3D water simulations for the purpose of computer graphics animation. We discussed some advances in rendering the simulated photorealistic behavior in order to complete the illusion that real water is being seen. The computational methods presented can easily be included into an existing three dimensional fluid simulation animation tool. While we have not done any texture mapping of the water surfaces, we believe that our "thickened" front tracking approach should enable texture mapping of an implicitly defined fluid surface, and we will pursue this as future work.

6 Acknowledgment

Research supported in part by an ONR YIP and PECASE award N00014-01-1-0620, NSF DMS-0106694, NSF ACI-0121288, NSF IIS-0085864, and the DOE ASCI Academic Strategic Alliances Program (LLNL contract B341491).

The authors acknowledge Henrik Wann Jensen for the use of his *dali* rendering system and for his help with the extensions that were added to it for this paper.

The third author would like to acknowledge the fruitful collaboration with Nick Foster regarding the hybridization of particle and level set methods that was published in [Foster and Fedkiw 2001]. Without it, neither [Enright et al. 2002] nor this current work would have been possible.

References

ADALSTEINSSON, D., AND SETHIAN, J. 1999. The fast construction of extension velocities in level set methods. *J. Comp. Phys. 148*, 2–22.

CHEN, J., AND LOBO, N. 1994. Toward interactive-rate simulation of fluids with moving obstacles using the navier-stokes equations. *Computer Graphics and Image Processing 57*, 107–116.

CHEN, S., JOHNSON, D., AND RAAD, P. 1995. Velocity boundary conditions for the simulation of free surface fluid flow. *J. Comp. Phys. 116*, 262–276.

CHEN, S., JOHNSON, D., RAAD, P., AND FADDA, D. 1997. The surface marker and micro cell method. *Int. J. for Num. Meth. in Fluids 25*, 749–778.

ENRIGHT, D., FEDKIW, R., FERZIGER, J., AND MITCHELL, I. 2002. A hybrid particle level set method for improved interface capturing. *J. Comp. Phys.*.

FOSTER, N., AND FEDKIW, R. 2001. Practical animation of liquids. In *Proceedings of SIGGRAPH 2001*, ACM Press / ACM

SIGGRAPH, E. Fiume, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 23–30.

FOSTER, N., AND METAXAS, D. 1996. Realistic animation of liquids. *Graphical Models and Image Processing 58*, 471–483.

FOURNIER, A., AND REEVES, W. T. 1986. A simple model of ocean waves. In *Computer Graphics (Proceedings of SIGGRAPH 86), 20(4)*, ACM, 75–84.

HARLOW, F., AND WELCH, J. 1965. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Phys. Fluids 8*, 2182–2189.

HECKBERT, P. S. 1990. Adaptive radiosity textures for bidirectional ray tracing. In *Computer Graphics (Proceedings of SIGGRAPH 90)*, vol. 24, ACM, 145–154.

HILTZIK, M. A., AND PHAM, A. 2001. Synthetic actors guild. *Los Angeles Times*. May 8, 2001, natl. ed. : A1+.

HIRT, C., AND NICHOLS, B. 1981. Volume of fluid (VOF) method for the dynamics of free boundaries. *J. Comp. Phys. 39*, 201–225.

JENSEN, H. W. 1995. Importance driven path tracing using the photon map. In *Eurographics Rendering Workshop 1995*, 326–335.

JENSEN, H. W. 2001. *Realistic Image Synthesis Using Photon Maps*. A K Peters.

JIANG, G.-S., AND PENG, D. 2000. Weighted eno schemes for hamilton-jacobi equations. *SIAM J. Sci. Comput. 21*, 2126–2143.

KAJIYA, J. T. 1986. The rendering equation. In *Computer Graphics (Proceedings of SIGGRAPH 86)*, vol. 20, 143–150.

KASS, M., AND MILLER, G. 1990. Rapid, stable fluid dynamics for computer graphics. In *Computer Graphics (Proceedings of SIGGRAPH 90)*, vol. 24, ACM, 49–57.

LAFORTUNE, E. P., AND WILLEMS, Y. D. 1993. Bi-directional path tracing. In *Proceedings of Compugraphics '93*, 145–153.

MARSCHNER, S., AND LOBB, R. 1994. An evaluation of reconstruction filters for volume rendering. In *Proceedings of Visualization 94*, IEEE Comput. Soc. Press, 100–107.

MASTEN, G., WATTERBERG, P., AND MAREDA, I. 1987. Fourier synthesis of ocean scenes. *IEEE Computer Graphics and Application 7*, 16–23.

MILLER, G., AND PEARCE, A. 1989. Globular dynamics: A connected particle system for animating viscous fluids. *Computers and Graphics 13*, 3, 305–309.

NISHITA, T., AND NAKAMAE, E. 1994. Method of displaying optical effects within water using accumulation buffer. In *Proceedings of SIGGRAPH 94*, ACM SIGGRAPH / ACM Press, Computer Graphics Proceedings, Annual Conference Series, ACM, 373–381.

O'BRIEN, J., AND HODGINS, J. 1995. Dynamic simulation of splashing fluids. In *Proceedings of Computer Animation 95*, 198–205.

OSHER, S., AND FEDKIW, R. 2002. *The Level Set Method and Dynamic Implicit Surfaces*. Springer-Verlag, New York.

OSHER, S., AND SETHIAN, J. 1988. Fronts propagating with curvature dependent speed: Algorithms based on hamiliton-jacobi formulations. *J. Comp. Phys. 79*, 12–49.

PEACHEY, D. R. 1986. Modeling waves and surf. In *Computer Graphics (Proceedings of SIGGRAPH 86), 20(4)*, ACM, 65–74.

PREMOZE, S., AND ASHIKHMIN, M. 2000. Rendering natural waters. In *The proceedings of Pacific Graphics 2000*, 23–30.

RADOVITZKY, R., AND ORTIZ, M. 1998. Lagrangian finite element analysis of newtonian fluid flows. *Int. J. Numer. Meth. Engng. 43*, 607–619.

SCHACHTER, B. 1980. Long crested wave models. *Computer Graphics and Image Processing 12*, 187–201.

SETHIAN, J. 1999. *Level Set Methods and Fast Marching Methods*. Cambridge University Press.

STAM, J. 1999. Stable fluids. In *Proceedings of SIGGRAPH 99*, ACM SIGGRAPH / Addison Wesley Longman, Computer Graphics Proceedings, Annual Conference Series, ACM, 121–128.

TERZOPOULOS, D., PLATT, J., AND FLEISCHER, K. 1989. Heating and melting deformable models (from goop to glop). In *Graphics Interface '89*, 219–226.

TESSENDORF, J. 2001. Simulating ocean water. In *SIGGRAPH 2001 Course Notes*.

TS'O, P. Y., AND BARSKY, B. A. 1987. Modeling and rendering waves: Wave-tracing using beta-splines and reflective and refractive texture mapping. *ACM Transactions on Graphics 6*, 3, 191–214.

UNVERDI, S.-O., AND TRYGGVASON, G. 1992. A front-tracking method for viscous, incompressible, multi-fluid flows. *J. Comp. Phys. 100*, 25–37.

VEACH, E., AND GUIBAS, L. J. 1994. Bidirectional estimators for light transport. In *Eurographics Rendering Workshop 1994 Proceedings*, 147–162.

VEACH, E., AND GUIBAS, L. J. 1997. Metropolis light transport. In *Proceedings of SIGGRAPH 97*, ACM SIGGRAPH / Addison Wesley, Computer Graphics Proceedings, Annual Conference Series, ACM, 65–76.

WATT, M. 1990. Light-water interaction using backward beam tracing. In *Computer Graphics (Proceedings of SIGGRAPH 90)*, vol. 24, ACM, 377–385.

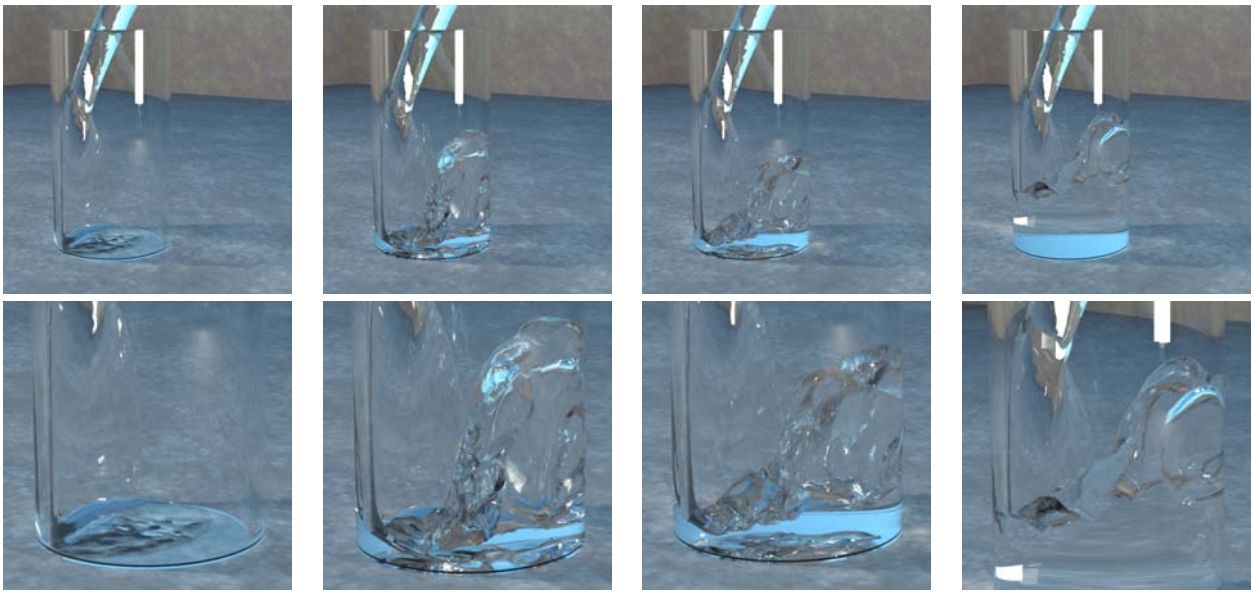


Figure 7: **Water being poured into a clear, cylindrical glass (55x55x120 grid cells). Our method makes possible the fine detail seen in the turbulent mixing of the water and air.**

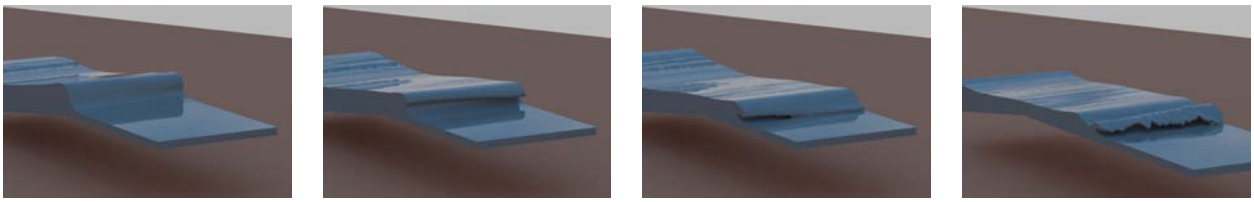


Figure 8: **Abstract view of wave breaking on a submerged shelf (360x50x80 grid cells). Note the ability to properly model the initial breaking (first three frames) and secondary splash up (last frame) phases.**

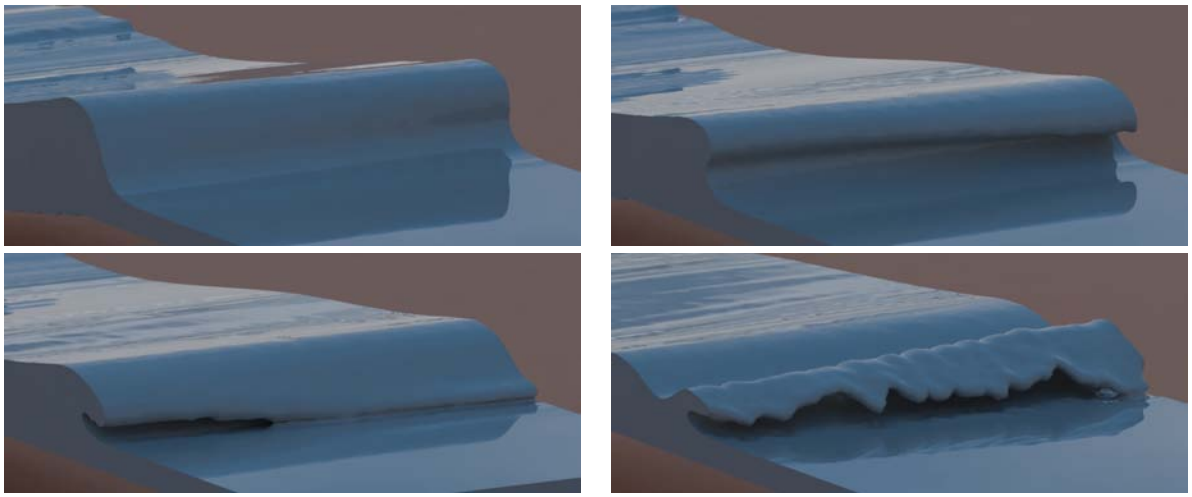


Figure 9: **Closeup of wave action seen in figure 8.**

Simulating Water and Smoke with an Octree Data Structure

Frank Losasso*
Stanford University
Industrial Light + Magic

Frédéric Gibou†
Stanford University

Ron Fedkiw‡
Stanford University
Industrial Light + Magic

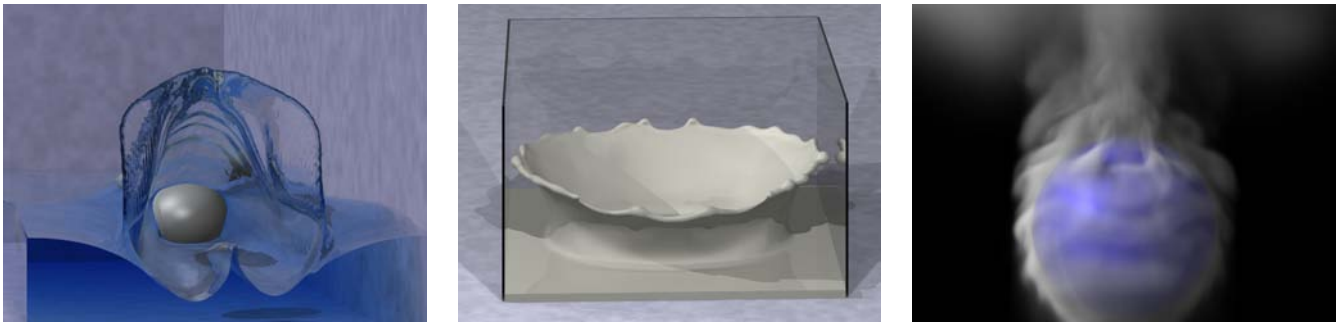


Figure 1: Ellipse traveling through a shallow pool of water (left), formation of a milk crown (center), smoke rising past a sphere (right).

Abstract

We present a method for simulating water and smoke on an *unrestricted* octree data structure exploiting mesh refinement techniques to capture the small scale visual detail. We propose a new technique for discretizing the Poisson equation on this octree grid. The resulting linear system is symmetric positive definite enabling the use of fast solution methods such as preconditioned conjugate gradients, whereas the standard approximation to the Poisson equation on an octree grid results in a non-symmetric linear system which is more computationally challenging to invert. The semi-Lagrangian characteristic tracing technique is used to advect the velocity, smoke density, and even the level set making implementation on an octree straightforward. In the case of smoke, we have multiple refinement criteria including object boundaries, optical depth, and vorticity concentration. In the case of water, we refine near the interface as determined by the zero isocontour of the level set function.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: octree data structure, adaptive mesh refinement, physics-based animation, smoke, water, level set, particles

1 Introduction

Realistic simulations of smoke and water are among the most desired in the special effects industry, since they provide the director with explicit control over the environment enabling the creation of otherwise impossible content. These phenomena contain highly complex motions and rich visual detail, especially when they interact with inanimate objects or the actors themselves. Moreover, a significant portion of the entertainment value and much of the believability relies on an adequate representation and presentation of the small scale visual details such as thin films in water, small rolling vortices in smoke, droplets and sprays, etc. Thus, it is desirable to have both simulation and rendering techniques that can deal with levels of detail.

Recent improvements in simulation techniques have led to impressive simulations of both smoke and water on uniform grids. Empowered by the semi-Lagrangian work of [Stam 1999], [Fedkiw et al. 2001] used vorticity confinement to simulate smoke with visually rich small scale rolling motions. Similarly using both semi-Lagrangian methods and hybridized particle and implicit surface techniques, [Foster and Fedkiw 2001; Enright et al. 2002b] simulated splashing water with both smooth surfaces and thin sheets. While these methods have achieved great success, their application is limited by the computational hardware (i.e. CPU, RAM, disk space) required for the simulations. In an attempt to alleviate this, [Rasmussen et al. 2003] proposed a method that combines interpolation and two-dimensional simulation to obtain highly detailed simulations of large scale smoke-like phenomena. While stunning results were obtained, this method does not faithfully reproduce the three-dimensional Navier-Stokes equations and thus is unable to obtain results for various fully three-dimensional phenomena. Moreover, water was not addressed.

In order to optimize the use of computational resources, we use an adaptive mesh or a level of detail approach where more grid cells are placed in visually interesting regions with rolling smoke or sheeting water. Although adaptive mesh strategies for incompressible flow are quite common, see e.g. [Ham et al. 2002], implementations based on recursive structures, such as the octrees we propose here, are less common. In fact, [Popinet 2003] claims to have the first octree implementation of incompressible flow, although there are certainly similar works such as the nested dyadic grids used for parabolic equations in [Roussel et al. 2003]. We extend the work of [Popinet 2003] in two ways. First, we extend octrees to free surface flows allowing the modeling of a liquid interface. Second, we consider *unrestricted* octrees whereas [Popinet 2003]'s octrees were restricted.

Adaptive meshing strategies lead to nonuniform stencils and thus a nonsymmetric system of linear equations when solving for the pressure, which is needed to enforce the divergence free condition. Although [Popinet 2003] solved this nonsymmetric linear system with a multilevel Poisson solver, [Day et al. 1998] pointed out that these multigrid approaches can be problematic in the presence of

*e-mail: losasso@graphics.stanford.edu
†e-mail: fgibou@math.stanford.edu
‡e-mail: fedkiw@cs.stanford.com

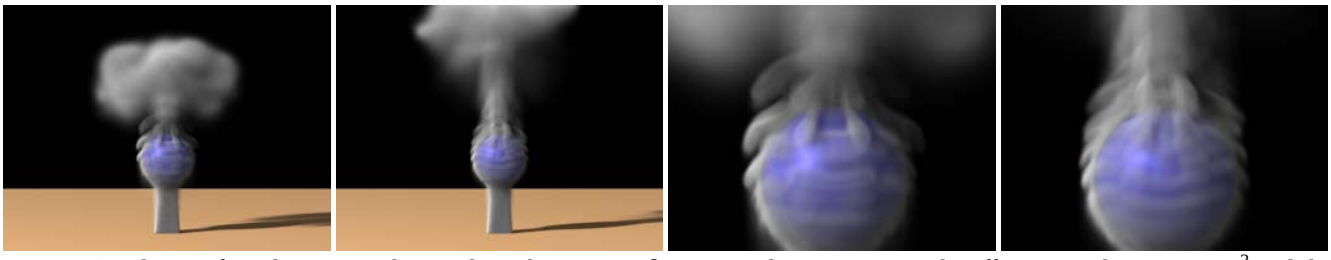


Figure 2: Simulation of smoke past a sphere. The rightmost two figures are close up views. The effective resolution is 1024^3 and the computational time is about 4-5 minutes per frame.

objects with high frequency detail. Moreover, the situation worsens in the presence of interfaces (such as that between water and air), especially since the faithful coarse mesh representation of watertight isosurfaces is a difficult research problem in itself, see e.g. [Lee et al. 2003]. Although multigrid solvers can be efficiently applied if the density is smeared out across the interface as in [Sussman et al. 1999] (resembling a one-phase variable density flow as in [Almgren et al. 1998]), this damps out the surface wave generation that relies on horizontal pressure differences caused by stacking different heights of high-density fluid. That is, damping these high frequency pressure differentials makes multigrid efficient, but also damps the wave motion leading to visually uninteresting overly viscous flows.

More recently, [Sussman 2003] departed from a smeared out density approach and instead solves a free surface problem as in [Enright et al. 2002b]. Moreover, [Sussman 2003] switches from multigrid to a preconditioned conjugate gradient (PCG) method stating that the pressure can be robustly solved for with PCG since the matrix is symmetric. However, the symmetry requirement limits his work to uniform non-adaptive grids. Our new formulation alleviates this restriction by providing a symmetric positive definite discretization of the Poisson equation on an unrestricted octree data structure allowing fast solvers such as PCG to be applied, even in the presence of interfaces.

2 Previous Work

[Kass and Miller 1990] solved a linearized form of the three dimensional Navier-Stokes equations, and [Chen and Lobo 1994] solved the two dimensional Navier-Stokes equations using the pressure to define a height field. The full three dimensional Navier-Stokes equations were solved in [Foster and Metaxas 1996; Foster and Metaxas 1997a; Foster and Metaxas 1997b] for both water and smoke. Large strides in efficiency were made when [Stam 1999] introduced the use of semi-Lagrangian numerical techniques, and [Fedkiw et al. 2001] advocated using vorticity confinement in order to preserve the small scale structure of the flow. [Foster and Fedkiw 2001; Enright et al. 2002b] proposed hybridizing particle and level set methods for water. The incompressible form of the Navier Stokes equations has been used and augmented to model fire [Lamorlette and Foster 2002; Nguyen et al. 2002], clouds [Miyazaki et al. 2002], particle explosions [Feldman et al. 2003], variable viscosity [Carlson et al. 2002], bubbles and surface tension [Hong and Kim 2003], splash and foam [Takahashi et al. 2003], etc. [Treuille et al. 2003] proposed a method for control and used it to make letters out of smoke, and [Stam 2003] solved these equations on surfaces creating beautiful imagery. The compressible version of these equations were used to couple fracture to explosions in [Yngve et al. 2000]. There are also other approaches such as SPH methods [Premoze et al. 2003; Müller et al. 2003].

The representation of implicit surfaces on octree data structures has a long history in the marching cubes community, see the recent papers of [Ju et al. 2002; Ohtake et al. 2003] and the references therein. Moreover, [Frisken et al. 2000; Perry and Frisken 2001] popularized the use of signed distance functions on octree grids. In order to simulate water, we need to solve the partial differential equations that govern the motion of the signed distance function. [Strain 1999b] advocated using quadtrees and semi-Lagrangian methods to solve these equations. Reinitialization for maintaining the signed distance property was addressed in [Strain 1999a], and extrapolation of velocities was considered in [Strain 2000]. One difficulty with semi-Lagrangian methods for solving level set equations is that extreme mass loss (and thus visual artifacts) usually occurs, however [Enright et al. 2004] recently showed that the particles in the particle level set method alleviate this difficulty. A quadtree structure for level set evolution was also proposed in [Sochnikov and Efrima 2003]. However, none of these authors considered level sets in the context of incompressible flows with interfaces such as water.

Starting with the seminal works of [Berger and Oliger 1984; Berger and Colella 1989], adaptive mesh refinement (AMR) typically utilizes uniform overlapping Cartesian grids of various sizes. This is because AMR originally focused on compressible flow with shock waves, and a block structured approach is better able to avoid spurious shock reflections from changing grid levels (since there are less of them). However, in the absence of shocks, a more optimal unrestricted octree approach can be used for incompressible flow.

3 The Octree Data Structure

Figure 3 illustrates our unrestricted octree data structure (see e.g. [Samet 1989]) with a standard MAC grid arrangement [Harlow and Welch 1965], except that all the scalars except the pressure are stored on the *nodes* or corners of the cell. This is convenient since interpolations are more difficult with cell centered data (see e.g. [Strain 1999a]).

Coarsening is performed from the smaller cells to the larger cells, i.e. from the leaves to the root. When coarsening, nodal values are either deleted or unchanged, and the new velocity components at the faces are computed by averaging the old values from that face. Refinement is performed from the larger cells to the smaller cells. The value of a new node on an edge is defined as the average of its two neighbors, and the value of a new node at a face center is defined as the average of the values on the four corners of that face. The velocities on the new faces are defined by first computing the velocities at the nodes, and then averaging back to the face centers. Nodal velocities are computed by averaging the four values from the surrounding cell faces as long as the faces are all the same size. Otherwise, using the coarsest neighboring face as the scale, we compute temporary coarsened velocities on the other faces to be used in the averaging.

For all variables, we constrain T-junction nodes on edges to be linearly interpolated from their neighbors on that edge. Similarly, T-junction nodes on faces are constrained to be the average of the four surrounding corner values. See, e.g. [Westermann et al. 1999] for more details.

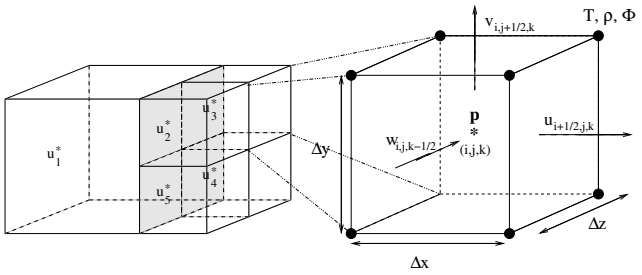


Figure 3: Left: One large cell neighboring four smaller cells. The u_i^* represent the x components of the intermediate velocity $\mathbf{u}^*$ defined at the cell faces. Right: Zoom of one computational cell. The velocity components are defined on the cell faces, while the pressure is defined at the center of the cell. The density, temperature and level set function ϕ are stored at the nodes.

4 Navier Stokes Equations on Octrees

We use the inviscid, incompressible Navier-Stokes equations for the conservation of mass and momentum

$$\begin{aligned} \mathbf{u}_t + \mathbf{u}\cdot\nabla\mathbf{u} &= -\nabla p + \mathbf{f}, & (1)\\ \nabla\cdot\mathbf{u} &= 0, & (2)\end{aligned}$$

where $\mathbf{u} = (u,v,w)$ is the velocity field, $\mathbf{f}$ accounts for the external forces, and the spatially constant density of the mixture has been absorbed into the pressure p. Equation 1 is solved in two steps. First we compute an intermediate velocity $\mathbf{u}^*$ ignoring the pressure term, and then we compute the velocity update via

$$\mathbf{u} = \mathbf{u}^* - \Delta t\nabla p \quad (3)$$

where the pressure is defined as the solution to the Poisson equation,

$$\nabla^2 p = \nabla\cdot\mathbf{u}^*/\triangle t. \quad (4)$$

The external forces are discretized at the cell faces and we postpone the details of their discretization to sections 5 and 6. The convective part of the velocity update is solved using a semi-Lagrangian stable fluids approach as in [Stam 1999]. First we compute nodal velocities, and then we average these values to the cell faces (see section 3). The cell face values are used to trace back characteristics, and trilinear interpolation of nodal values is used to define the new intermediate value of the velocity component on the face in question.

4.1 The Divergence Operator

Equation 4 is solved by first evaluating the right hand side at every cell center in the domain. Then a linear system for the pressure is constructed and inverted. Consider the discretization of equation 4 for a large cell with dimensions $\triangle x$, $\triangle y$ and $\triangle z$ neighboring small cells as depicted in figure 3. Since the discretization is closely related to the second vector form of Green's theorem that relates a volume integral to a surface integral, we first rescale equation 4 by the volume of the large cell to obtain $\mathrm{V}_{\mathrm{cell}}\triangle t\nabla^2 p = \mathrm{V}_{\mathrm{cell}}\nabla\cdot\mathbf{u}^*$. The right hand side now represents the quantity of mass flowing in and out of the large cell within a time step $\triangle t$ in $m^3 s^{-1}$. This can be further rewritten as

$$\mathrm{V}_{\mathrm{cell}}\nabla\cdot(\mathbf{u}^* - \triangle t\nabla p) = 0. \quad (5)$$

This equation implies that the ∇p term is most naturally evaluated at the same location as $\mathbf{u}^*$, namely at the cell faces, and that there is a direct correspondence between the components of ∇p and $\mathbf{u}^*$.

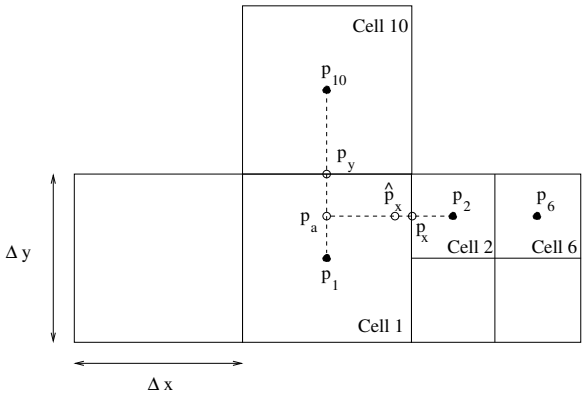


Figure 4: Discretization of the pressure gradient.

Moreover, substituting equation 3 into equation 5 implies $\mathrm{V}_{\mathrm{cell}}\nabla\cdot\mathbf{u} = 0$ or $\nabla\cdot\mathbf{u} = 0$ as desired.

Invoking the second vector form of Green's theorem, one can write

$$\mathrm{V}_{\mathrm{cell}}\nabla\cdot\mathbf{u}^* = \sum_{\mathrm{faces}} (\mathbf{u}^*_{\mathrm{face}}\cdot\mathbf{n})\,\mathrm{A}_{\mathrm{face}},$$

where $\mathbf{n}$ is the *outward* unit normal of the large cell and A_{face} represents the area of a cell face. In the case of figure 3, the discretization of the x component $\partial\mathbf{u}^*/\partial x$ of the divergence reads $\triangle x\triangle y\triangle z\partial\mathbf{u}^*/\partial x = u_2^*\mathrm{A}_2 + u_3^*\mathrm{A}_3 + u_4^*\mathrm{A}_4 + u_5^*\mathrm{A}_5 - u_1^*\mathrm{A}_1$, where the *minus* sign in front of $u_1^*\mathrm{A}_1$ accounts for the fact that the unit normal points to the left. Then $\partial\mathbf{u}^*/\partial x = ((u_2^* + u_3^* + u_4^* + u_5^*)/4 - u_1^*)/\triangle x$. The y and z directions are treated similarly.

Once, the divergence is computed at the cell center, equation 4 is used to construct a linear system of equations for the pressure. Invoking again the second vector form of Green's theorem, one can write

$$\mathrm{V}_{\mathrm{cell}}\nabla\cdot(\triangle t\nabla p) = \sum_{\mathrm{faces}} ((\triangle t\nabla p)_{\mathrm{face}}\cdot\mathbf{n})\,\mathrm{A}_{\mathrm{face}}. \quad (6)$$

Therefore, once the pressure gradient is computed at every face, we can carry out the computation in a manner similar to that of the velocity divergence above. There exist different choices in the discretization of $(\nabla p)_{\mathrm{face}}$, and we seek to discretize the pressure gradient in a fashion that yields a symmetric linear system. Efficient iterative methods such as PCG (see e.g. [Saad 1996]) can be applied to symmetric positive definite matrices offering a significant advantage over methods for nonsymmetric linear systems. Moreover, since data access for the octree is not as convenient as for regular grids, there is a strong benefit in designing a discretization that leads to a symmetric linear system.

4.2 The Pressure Gradient

Consider the configuration in figure 4. In the case where two cells of the same size juxtapose each other, standard central differencing defines the pressure gradient at the face between them, as is the case for $p_y = (p_{10} - p_1)/\triangle y$.

Consider the discretization of the pressure gradient in the x direction at the face between cell 1 and cell 2. A standard approach is to first compute a weighted average value p_a for the pressure, by interpolating between the pressure values p_1 and p_{10}. Then, since standard differencing of $\hat{p}_x = (p_2 - p_a)/(.75\triangle x)$ does not define $\hat{p}_x$ at the cell face but midway between the locations of p_a and p_2, one usually resorts to more complex discretizations. A typical choice is to pass a quadratic interpolant through p_a, p_2 and p_6 and evaluate its derivative at the cell face, see e.g. [Chen et al. 1997]. However, this approach yields a nonsymmetric linear system that is slow to invert. The nonsymmetric nature of the linear system comes from

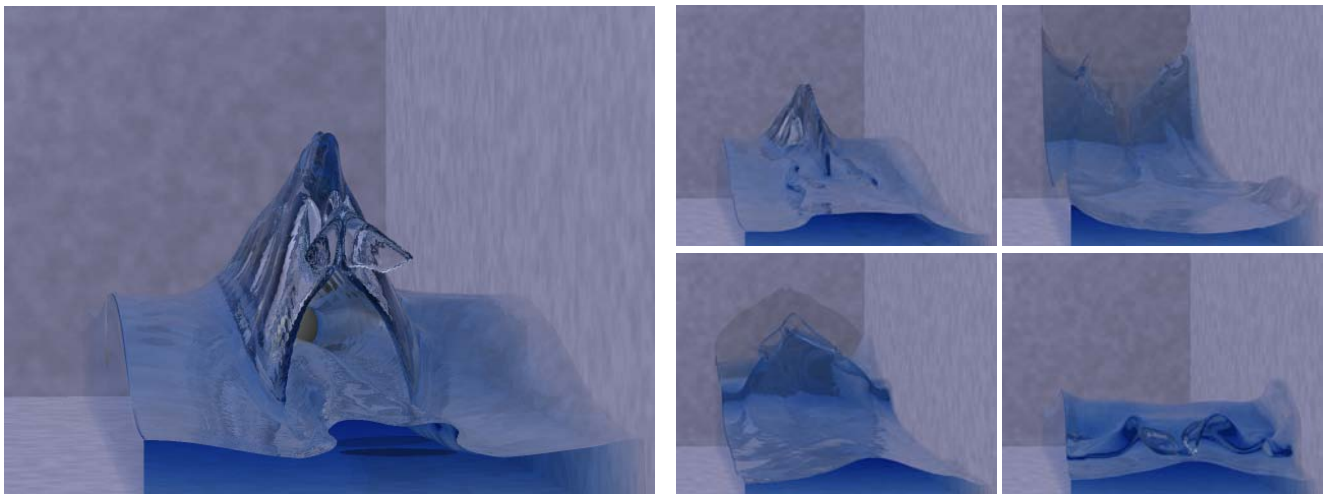


Figure 5: An ellipsoid slips along through shallow water illustrating our method's ability to resolve thin sheets. The effective resolution is 512^3 and the computational time is about 4-5 minutes per frame.

the non-locality of the discretization, i.e. p_a depends on p_{10} and the quadratic interpolation would depend on p_6. Consequently, the equation for cell 1 involves both p_{10} and p_6. It is unlikely that the equation for cell 6 depends on p_1, since cell 6 juxtaposes another cell of the same size, namely cell 2. And even if it did, the coefficients of dependence would not be symmetric.

Our approach is based on the fact that $O(\triangle x)$ perturbations in the pressure location still yield consistent approximations as in [Gibou et al. 2002]. Therefore defining $p_x = (p_2 - p_a)/(.75\triangle x)$ *at* the cell face still yields a convergent approximation, since the location of $\hat{p}_x$ is perturbed by a small amount proportional to a grid cell. Moreover, we can avoid the dependence of p_a on values other than p_1 by simply setting $p_a = p_1$. This corresponds to an $O(\triangle x)$ perturbation of the location of p_1, and therefore still yields a convergent approximation. Thus, our discretization of p_x is simply $p_x = (p_2 - p_1)/(.75\triangle x)$. Moreover, since only p_1 and p_2 are considered, one can define $p_x = (p_2 - p_1)/\triangle$ where $\triangle$ can be defined as the size of the large cell, $\triangle x$, the size of the small cell, $.5\triangle x$, the Euclidean distance between p_1 and p_2, etc. We have carried out numerical tests against known analytic solutions to the Poisson equation demonstrating that all these choices converge. Currently, we are investigating the impact of different $\triangle$ definitions on smoke and water simulations.

In light of equation 6, p_x contributes to both row 1 and row 2 of the matrix representing the linear system of equations, since it is located at the cell face between cell 1 and cell 2. More precisely, the contribution to row 1 occurs through the term

$$\triangle t\, p_x n_1 A_{\text{face}} = \triangle t \frac{p_2 - p_1}{\triangle}(1) A_{\text{face}},$$

since n_1, the x component of the outward normal to cell 1, points to the right (hence $n_1 = 1$). Likewise, the contribution to row 2 occurs through the term

$$\triangle t\, p_x n_1 A_{\text{face}} = \triangle t \frac{p_2 - p_1}{\triangle}(-1) A_{\text{face}},$$

since n_1, the x component of the outward normal to cell 2, points to the left (hence $n_1 = -1$). Therefore, the coefficient for p_2 in row 1 and the coefficient for p_1 for row 2 are identical, namely $\triangle t A_{\text{face}}/\triangle$. The same procedure is applied to all faces, and the discretization of the y and z components of the pressure gradient are carried out in a similar manner. Hence, our discretization yields a symmetric linear system that can be efficiently inverted with a PCG method. The preconditioner we use is based on an incomplete *LU* Cholesky factorization that we modify to ensure that the row sum of *LU* is equal to the row sum of the original matrix (see [Saad 1996]). This yields a significant speed up in the matrix inversion.

The matrix constructed above is negative definite, as is usual when discretizing equation 4. We simply multiply all equations by -1 to make it positive definite. We also note that Dirichlet or Neumann boundary conditions do not disrupt the symmetry. In the case of a Neumann boundary condition, the term $(p_2 - p_1)/\triangle$ disappears from both row 1 and row 2. In the case of a Dirichlet boundary condition, e.g. for p_2, the equation for p_2 drops out of the system and all the terms involving p_2 are moved to the right hand side of the linear system.

4.3 Accuracy

We stress that the dominant errors are due to the first order accurate semi-Lagrangian advection scheme. The velocity averaging is second order accurate and is required in all MAC grid methods in order to define a full velocity vector at a common location for the semi-Lagrangian advection. Dropping the Poisson solver from second to first order accuracy merely puts it on par asymptotically with the semi-Lagrangian scheme. However, we still solve for a fully divergence free velocity field to machine precision just as in a non-adaptive setting. We tested our Poisson solver on many exact solutions and readily obtain several digits of accuracy indicating that the errors from this part of the algorithm are small. See table 1 for a typical result.

5 Smoke

The external forces due to buoyancy and heat convection are modeled as $\mathbf{f}_{\text{buoy}} = -\alpha\rho\mathbf{z} + \beta(T - T_{\text{amb}})\mathbf{z}$, where $\mathbf{z} = (0,0,1)$, T_{amb} is the ambient temperature and α and β are parameters controlling the influence of the density and the temperature. The density and the temperature are passively advected with the flow velocity and are updated with the semi-Lagrangian method using velocities defined at the nodes (see section 3). Both the density and the temperature are then averaged to the faces in order to evaluate the forcing term.

The vorticity confinement force is calculated as follows. First we define velocities at the centers of cells by using area weighted averaging of face values. Then all the derivatives needed to compute the vorticity, $\boldsymbol{\omega} = \nabla \times \mathbf{u}$, are computed on cell faces using the same method used to compute pressure derivatives. Area weighted averaging is used (again) to define all these derivatives at the cell center, and then we compute the vorticity and its magnitude (at

L^1error	order	L^∞error	order
4.083×10^{-2}	$--$	6.332×10^{-2}	$--$
8.713×10^{-3}	2.22	2.203×10^{-2}	1.523
2.952×10^{-3}	1.56	1.292×10^{-2}	.770
9.980×10^{-4}	1.56	7.745×10^{-3}	.739
4.010×10^{-4}	1.31	4.249×10^{-3}	.866
1.820×10^{-4}	1.14	2.287×10^{-3}	.894

Table 1: Poisson solver accuracy on an unrestricted octree grid.

the cell center). Next, the gradients of the vorticity are computed at the cell faces, and averaged back to the cell center to define $\mathbf{N} = \nabla|\omega|/|\nabla|\omega||$. Finally, the unscaled force can be computed at the cell centers as $\mathbf{N}\times\omega$. Cell face values of this term are obtained by averaging the values from the two cells that contain the face. Then this term is scaled by the diagonal of the face h and a tunable parameter ε.

Inside an object, we set the temperature to the object temperature and the density to zero. For velocity, we clip the component normal to the object so that it is guaranteed to be separating. Furthermore, we apply Neumann boundary conditions to the cell faces that intersect the object when solving for the pressure. This keeps these velocities fixed.

In the case of smoke, we utilize three different refinement criteria. First, we refine near objects since their interactions with smoke will introduce small scales features that enhance believability. Second, we refine near concentrations of high vorticity. Third, we refine in a band of density values (for example $.1 < \rho < .3$). This last criteria prunes out both the low densities that cannot be seen as well as the high densities interior to the smoke which are self-occluded.

6 Water

We use the particle level set method of [Enright et al. 2002a] with $\phi \le 0$ designating the water and $\phi > 0$ representing the air. When solving for the pressure, one only needs to consider cells in the water. Dirichlet boundary conditions of $p_I = p_{\text{air}} + \sigma\kappa$ are set in the air cells bordering the water, where σ is a surface tension coefficient and $\kappa = \nabla\cdot(\nabla\phi/|\nabla\phi|)$ is the local interface curvature. We note that [Hong and Kim 2003] considered surface tension in the case of bubbles, but not for films. κ is computed by averaging nodal values of ϕ to the cell center, computing derivatives of ϕ on the cell faces, averaging these back to the cell center, using these cell centered values to obtain the normal, computing derivatives of the normal on the cell faces, averaging these values back to the cell center, and finally using these cell centered values to obtain the curvature. The only external force we account for is gravity via $\mathbf{u} += \triangle t\mathbf{g}$. The interaction with objects is similar to that of smoke. We apply adaptive refinement to a band about the interface (focusing more heavily on the water side), noting that the signed distance property of ϕ makes this straightforward.

Recently, [Enright et al. 2004] showed that the particle level set method relies on particles for accuracy and the level set for connectivity. Moreover, they showed that one could use a simple semi-Lagrangian method on the level set with no significant accuracy penalty as long as the particles are evolved with at least second order Runge-Kutta. Thus, we update ϕ with the semi-Lagrangian method using velocities defined at the nodes (see section 3). The particles are advected using second order Runge-Kutta and trilinearly interpolated nodal velocities.

We use the fast marching method [Tsitsiklis 1995; Sethian 1996] to maintain the signed distance property of ϕ. First, the signed distance is computed at all the nodes around the interface, and they are marked as *updated*. The nodes adjacent to the *updated* nodes are tagged *trial*. Then we compute potential values of ϕ at all *trial* nodes using only *updated* nodes. The smallest of these is tagged as *updated*, and all its non-updated neighbors are tagged *trial*. This process is repeated to fill in a band of values near the interface. For many grid nodes there are neighboring values of ϕ in all six directions, but at T-junctions there are directions where ϕ is missing some of its neighbors. Since we coarsen as we move away from the interface, these directions will generally not contribute to the potential value of ϕ. Thus, we trivially ignore them.

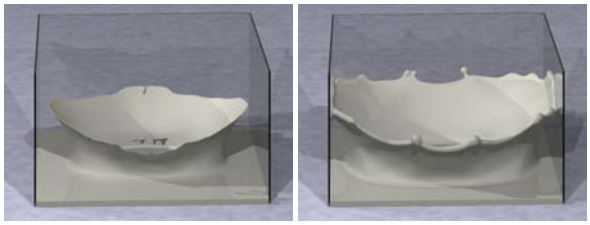


Figure 6: Formation of a milk crown demonstrating the effect of surface tension. We take $\sigma = 0$ (left) and $\sigma = .0005$ (right). The effective resolution is 512^3 and the computational time is about 4-5 minutes per frame.

Velocity extrapolation is carried out by first defining nodal velocities, extrapolating them, and then computing the face velocities. If we perform this algorithm as in [Enright et al. 2002a], we will occasionally encounter grid points that have no neighboring values of velocity and cannot be updated. While this is rare (but not impossible) for uniform grids, T-junctions exacerbate their occurrence on octree grids. When this happens, we skip over these nodes until one of their neighbors is updated (and then we update them in the usual manner).

7 Conclusions

Our new symmetric formulation reduces the pressure solver to approximately 25% of the total simulation time requiring only about 20 iterations to converge to an accuracy of machine precision. This leaves little room for improvement and even a zero cost pressure solver would only make the code 25% faster. On the other hand, nonsymmetric formulations requiring BiCGSTAB or GMRES and nonoptimal preconditioners easily lead to an order of magnitude slowdown, or in the worst case scenario problems with robustly finding a solution at all. The symmetric formulation enables a full octree discretization of the equations that govern both the flow of smoke and water. Moreover, we achieved reasonable computational costs on grids as fine as 1024^3 allowing us to capture fine scale rolling motion in smoke and thin films for water.

8 Acknowledgement

Research supported in part by an ONR YIP award and a PECASE award (ONR N00014-01-1-0620), a Packard Foundation Fellowship, a Sloan Research Fellowship, ONR N00014-03-1-0071, ONR N00014-02-1-0720, ARO DAAD19-03-1-0331, NSF DMS-0106694, NSF ITR-0121288, NSF IIS-0326388, NSF ACI-0323866 and NSF ITR-0205671. In addition, F. G. was supported in part by an NSF Postdoctoral Fellowship (DMS-0102029).

We'd also like to thank Mike Houston for providing computing resources used for rendering.

References

ALMGREN, A., BELL, J., COLELLA, P., HOWELL, L., AND WELCOME, M. 1998. A conservative adaptive projection method for the variable density incompressible navier-stokes equations. *J. Comput. Phys. 142*, 1–46.

BERGER, M., AND COLELLA, P. 1989. Local adaptive mesh refinement for shock hydrodynamics. *J. Comput. Phys. 82*, 64–84.

BERGER, M., AND OLIGER, J. 1984. Adaptive mesh refinement for hyperbolic partial differential equations. *J. Comput. Phys. 53*, 484–512.

CARLSON, M., MUCHA, P., VAN HORN III, R., AND TURK, G. 2002. Melting and flowing. In *ACM SIGGRAPH Symposium on Computer Animation*, 167–174.

CHEN, J., AND LOBO, N. 1994. Toward interactive-rate simulation of fluids with moving obstacles using the navier-stokes equations. *Computer Graphics and Image Processing 57*, 107–116.

CHEN, S., MERRIMAN, B., OSHER, S., AND SMEREKA, P. 1997. A simple level set method for solving stefan problems. 8–29.

DAY, M., COLELLA, P., LIJEWSKI, M., RENDLEMAN, C., AND MARCUS, D. 1998. Embedded boundary algorithms for solving the poisson equation on complex domains. Tech. rep., Lawrence Berkeley National Laboratory (LBNL-41811).

ENRIGHT, D., FEDKIW, R., FERZIGER, J., AND MITCHELL, I. 2002. A hybrid particle level set method for improved interface capturing. *J. Comp. Phys. 183*, 83–116.

ENRIGHT, D., MARSCHNER, S., AND FEDKIW, R. 2002. Animation and rendering of complex water surfaces. *ACM Trans. Graph. (SIGGRAPH Proc.) 21*, 3, 736–744.

ENRIGHT, D., LOSASSO, F., AND FEDKIW, R. 2004. A fast and accurate semi-Lagrangian particle level set method. *Computers and Structures, (in press)*.

FEDKIW, R., STAM, J., AND JENSEN, H. 2001. Visual simulation of smoke. In *Proc. of ACM SIGGRAPH 2001*, 15–22.

FELDMAN, B. E., O'BRIEN, J. F., AND ARIKAN, O. 2003. Animating suspended particle explosions. *ACM Trans. Graph. (SIGGRAPH Proc.) 22*, 3, 708–715.

FOSTER, N., AND FEDKIW, R. 2001. Practical animation of liquids. In *Proc. of ACM SIGGRAPH 2001*, 23–30.

FOSTER, N., AND METAXAS, D. 1996. Realistic animation of liquids. *Graph. Models and Image Processing 58*, 471–483.

FOSTER, N., AND METAXAS, D. 1997. Controlling fluid animation. In *Computer Graphics International 1997*, 178–188.

FOSTER, N., AND METAXAS, D. 1997. Modeling the motion of a hot, turbulent gas. In *Proc. of SIGGRAPH 97*, 181–188.

FRISKEN, S., PERRY, R., ROCKWOOD, A., AND JONES, T. 2000. Adaptively sampled distance fields: a general representation of shape for computer graphics. In *Proc. SIGGRAPH 2000*, 249–254.

GIBOU, F., FEDKIW, R., CHENG, L.-T., AND KANG, M. 2002. A second–order–accurate symmetric discretization of the poisson equation on irregular domains. 205–227.

HAM, F., LIEN, F., AND STRONG, A. 2002. A fully conservative second-order finite difference scheme for incompressible flow on nonuniform grids. *J. Comput. Phys. 117*, 117–133.

HARLOW, F., AND WELCH, J. 1965. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Phys. Fluids 8*, 2182–2189.

HONG, J.-M., AND KIM, C.-H. 2003. Animation of bubbles in liquid. *Comp. Graph. Forum (Eurographics Proc.) 22*, 3, 253–262.

JU, T., LOSASSO, F., SCHAEFER, S., AND WARREN, J. 2002. Dual contouring of Hermite data. In *Proc. of SIGGRAPH 2002*, 339–346.

KASS, M., AND MILLER, G. 1990. Rapid, stable fluid dynamics for computer graphics. In *Computer Graphics (Proc. of SIGGRAPH 90)*, vol. 24, 49–57.

LAMORLETTE, A., AND FOSTER, N. 2002. Structural modeling of natural flames. *ACM Trans. Graph. (SIGGRAPH Proc.) 21*, 3, 729–735.

LEE, H., DESBRUN, M., AND SCHRODER, P. 2003. Progressive encoding of complex isosurfaces. In *Proc. of SIGGRAPH 2003*, 471–476.

MIYAZAKI, R., DOBASHI, Y., AND NISHITA, T. 2002. Simulation of cumuliform clouds based on computational fluid dynamics. *Proc. Eurographics 2002 Short Presentation*, 405–410.

MÜLLER, M., CHARYPAR, D., AND GROSS, M. 2003. Particle-based fluid simulation for interactive applications. In *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 154–159.

NGUYEN, D., FEDKIW, R., AND JENSEN, H. 2002. Physically based modeling and animation of fire. In *ACM Trans. Graph. (SIGGRAPH Proc.)*, vol. 29, 721–728.

OHTAKE, Y., BELYAEV, A., ALEXA, M., TURK, G., AND SEIDEL, H. 2003. Multi-Level Partition of Unity Implicits. In *Proc. of SIGGRAPH 2003*, 463–470.

PERRY, R., AND FRISKEN, S. 2001. Kizamu: a system for sculpting digital characters. *Comput. Graph. (SIGGRAPH Proc.)*, 47–56.

POPINET, S. 2003. Gerris: A tree-based adaptive solver for the incompressible euler equations in complex geometries. *J. Comp. Phys. 190*, 572–600.

PREMOZE, S., TASDIZEN, T., BIGLER, J., LEFOHN, A., AND WHITAKER, R. 2003. Particle–based simulation of fluids. In *Comp. Graph. Forum (Eurographics Proc.)*, vol. 22, 401–410.

RASMUSSEN, N., NGUYEN, D., GEIGER, W., AND FEDKIW, R. 2003. Smoke simulation for large scale phenomena. *ACM Trans. Graph. (SIGGRAPH Proc.) 22*, 703–707.

ROUSSEL, O., SCHNEIDER, K., TSIGULIN, A., AND BOCKHORN, H. 2003. A conservative fully adaptive multiresolution algorithm for parabolic pdes. *J. Comput. Phys. 188*, 493–523.

SAAD, Y. 1996. *Iterative methods for sparse linear systems*. PWS Publishing. New York, NY.

SAMET, H. 1989. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, New York.

SETHIAN, J. 1996. A fast marching level set method for monotonically advancing fronts. *Proc. Natl. Acad. Sci. 93*, 1591–1595.

SOCHNIKOV, V., AND EFRIMA, S. 2003. Level set calculations of the evolution of boundaries on a dynamically adaptive grid. *Int. J. Num. Methods in Eng. 56*, 1913–1929.

STAM, J. 1999. Stable fluids. In *Proc. of SIGGRAPH 99*, 121–128.

STAM, J. 2003. Flows on surfaces of arbitrary topology. *ACM Trans. Graph. (SIGGRAPH Proc.) 22*, 724–731.

STRAIN, J. 1999. Fast tree-based redistancing for level set computations. *J. Comput. Phys. 152*, 664–686.

STRAIN, J. 1999. Tree methods for moving interfaces. *J. Comput. Phys. 151*, 616–648.

STRAIN, J. 2000. A fast modular semi-lagrangian method for moving interfaces. *J. Comput. Phys. 161*, 512–536.

SUSSMAN, M., ALMGREN, A., BELL, J., COLELLA, P., HOWELL, L., AND WELCOME, M. 1999. An adaptive level set approach for incompressible two-phase flows. *J. Comput. Phys. 148*, 81–124.

SUSSMAN, M. 2003. A second order coupled level set and volume-of-fluid method for computing growth and collapse of vapor bubbles. *J. Comp. Phys. 187*, 110–136.

TAKAHASHI, T., FUJII, H., KUNIMATSU, A., HIWADA, K., SAITO, T., TANAKA, K., AND UEKI, H. 2003. Realistic animation of fluid with splash and foam. *Comp. Graph. Forum (Eurographics Proc.) 22*, 3, 391–400.

TREUILLE, A., MCNAMARA, A., POPOVIĆ, Z., AND STAM, J. 2003. Keyframe control of smoke simulations. *ACM Trans. Graph. (SIGGRAPH Proc.) 22*, 3, 716–723.

TSITSIKLIS, J. 1995. Efficient algorithms for globally optimal trajectories. *IEEE Trans. on Automatic Control 40*, 1528–1538.

WESTERMANN, R., KOBBELT, L., AND ERTL, T. 1999. Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces. *The Vis. Comput. 15*, 2, 100–111.

YNGVE, G., O'BRIEN, J., AND HODGINS, J. 2000. Animating explosions. In *Proc. of SIGGRAPH 2000*, 29–36.