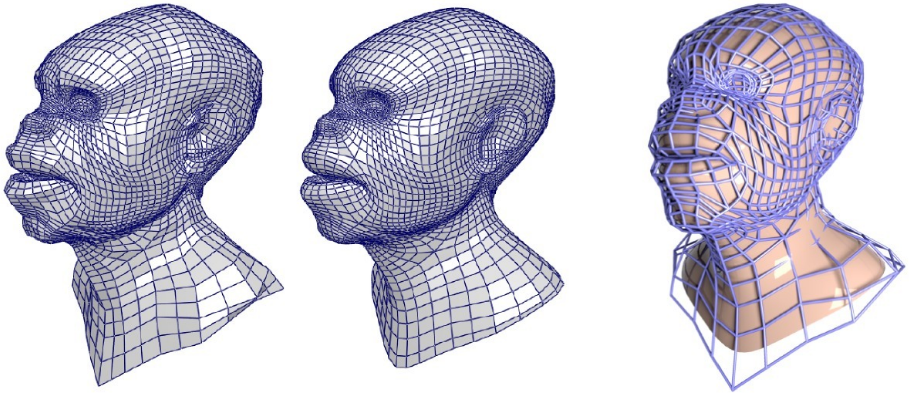
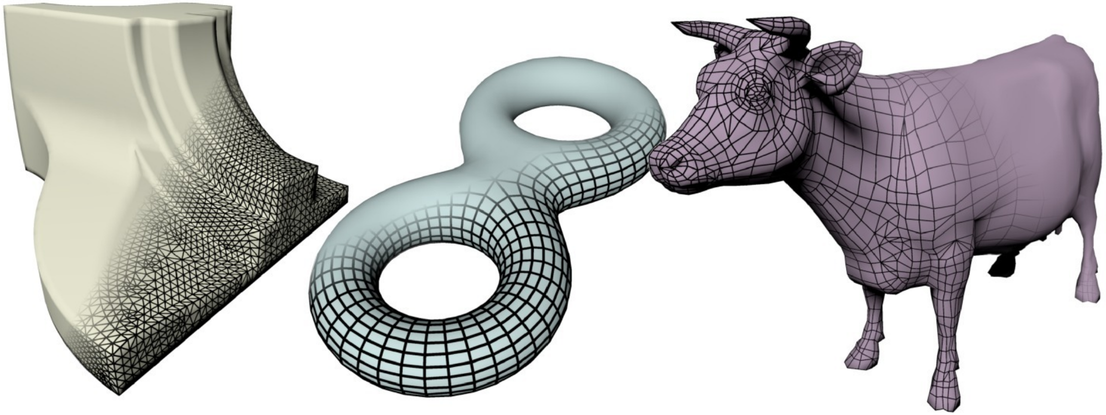
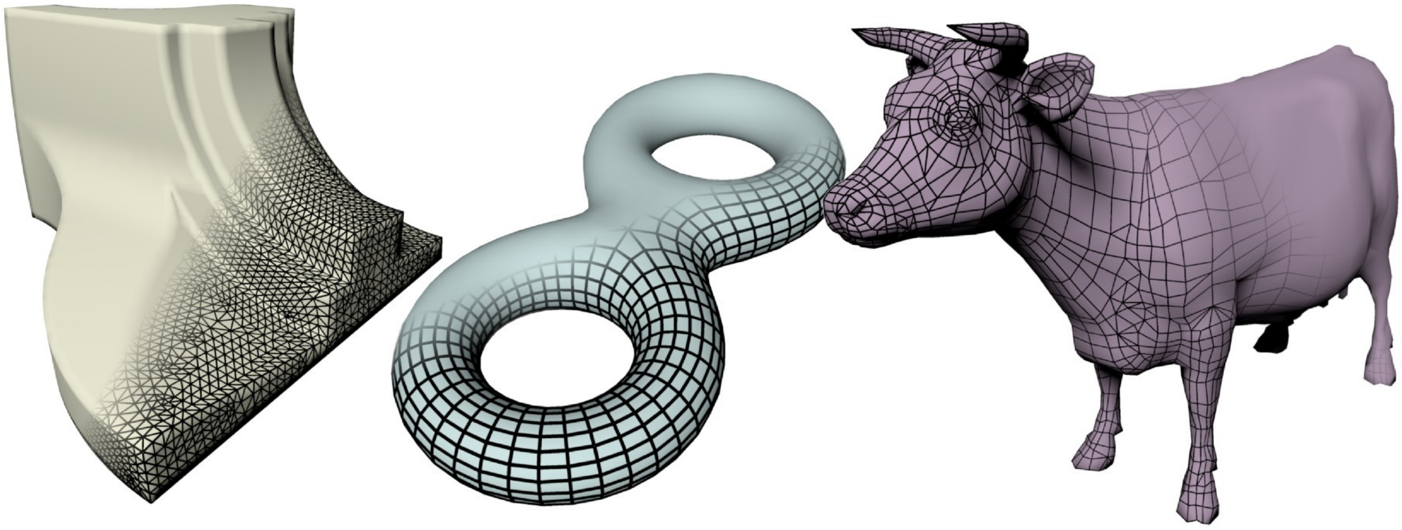


Meshes



Meshes

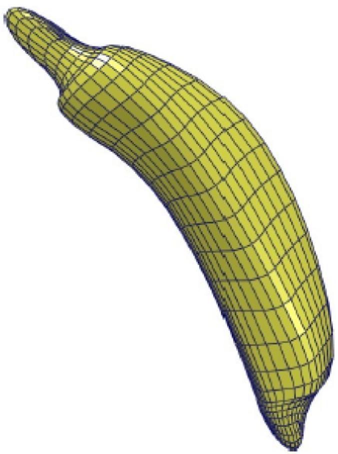
Mesh = Set of **polygons** sharing some edges
$\mathbf{N}_f$ faces, $\mathbf{N}_s$ vertices, $\mathbf{N}_e$ edges.
Triangulation: all faces are triangles
Quad mesh: all faces are quadrangles.
Poly mesh: multiple polygons



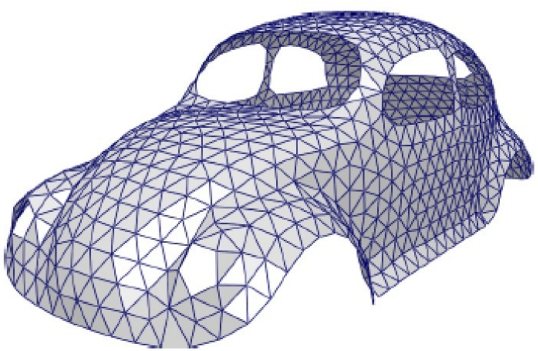
Topology

A surface is a manifold if the neighborhood of every point is homeomorphic to a (half) disc
=> Every edge is shared by at most 2 faces (connectivity)
 + no self-intersection (embeeding)

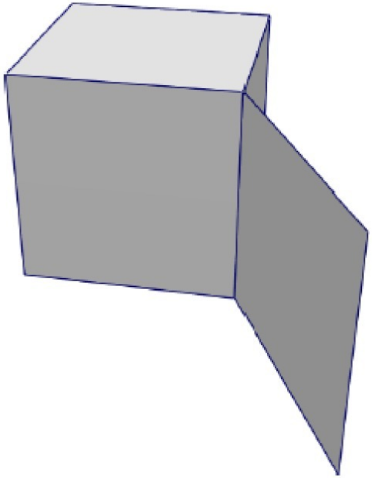
2-manifold



2-manifold
with boundaries



not a 2-manifold



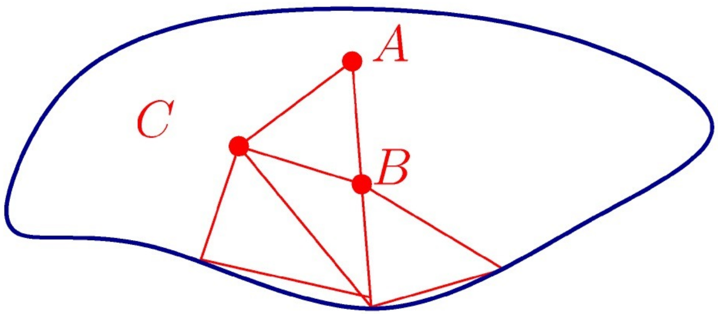
Meshes

Poly-mesh : Special case of triangulation

Triangulation = Linear map S

$$S_i : \begin{cases} \mathcal{D} \subset \mathbb{R}^2 & \to \mathbb{R}^3 \\ (u, v) & \mapsto S_i(u, v) = u\,\overrightarrow{AB} + v\,\overrightarrow{AC} + \overrightarrow{OA} \end{cases}$$

$$\mathcal{D} : (u, v) \in [0, 1]^2, 0 \leq u + v \leq 1$$



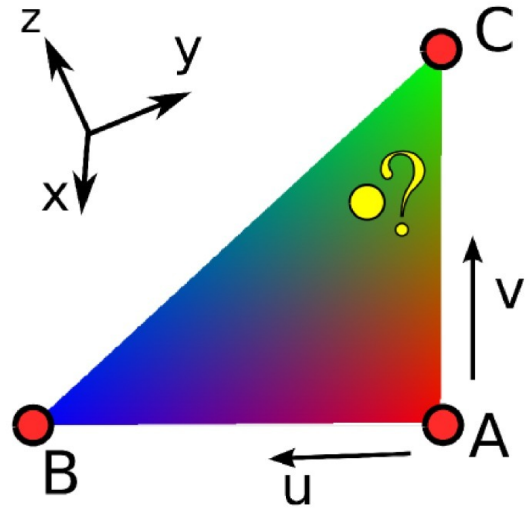
Coordinate within a triangle

Position of point **p** with respect to vertices (A,B,C) ?

$$\overrightarrow{AP} = u\,\overrightarrow{AB} + v\,\overrightarrow{AC}$$

$$\Rightarrow P - A = u\,(B - A) + v\,(C - A)$$

$$\Rightarrow P = \underbrace{(1 - u - v)}_{w}\,A + u\,B + v\,C\ .$$



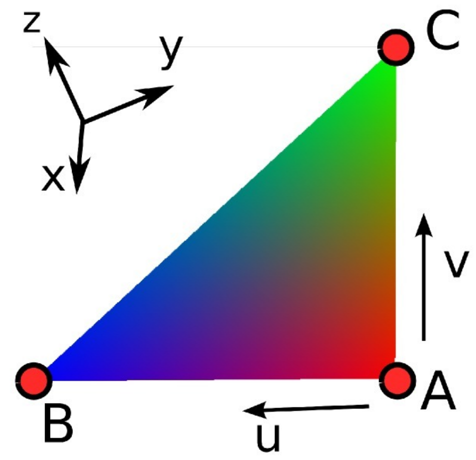
(u,v,w)=Barycentric coordinates

$$\begin{cases} P = w\,A + u\,B + v\,C \\ u + v + w = 1 \\ 0 \leq (u, v, w) \leq 1\ . \end{cases}$$

Linear interpolation

Color interpolation

$$\begin{cases} r(u,v) &= (1-u-v)r_A + ur_B + vr_C \\ g(u,v) &= (1-u-v)g_A + ug_B + vg_C \\ b(u,v) &= (1-u-v)b_A + ub_B + vb_C \end{cases}$$



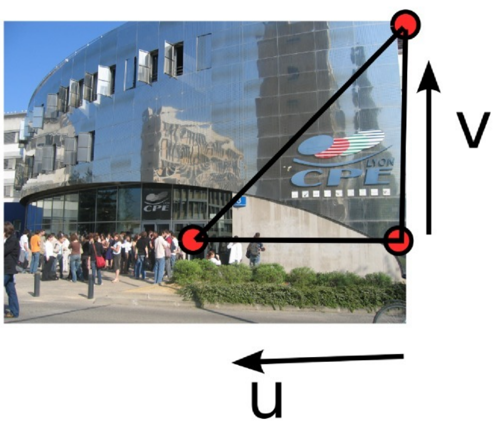
For a general function f defined per vertices

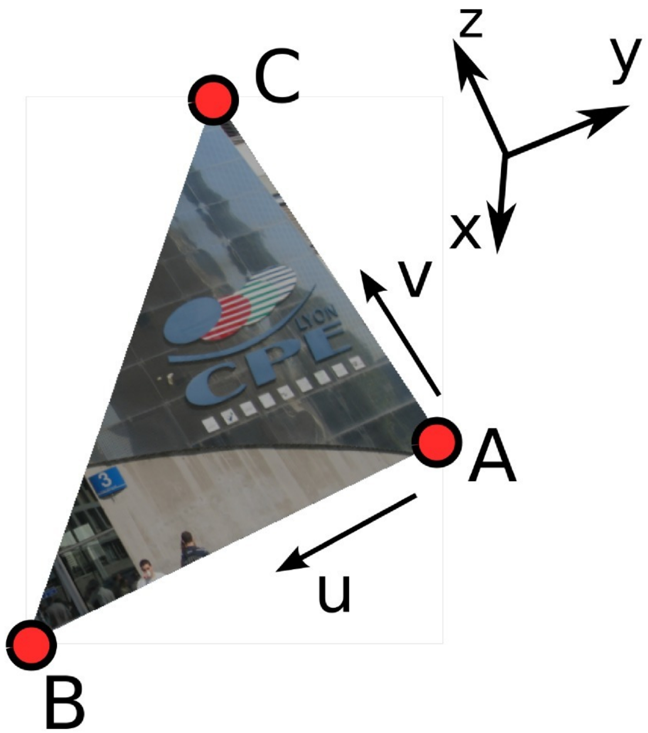
$$f(u,v) = (1-u-v)f_A + uf_B + vf_C$$

Linear interpolation

Can interpolate textures coordinates

$$\begin{cases} t_x = (1-u-v)\,t_x(A) + u\,t_x(B) + v\,t_x(C) \\ t_y = (1-u-v)\,t_y(A) + u\,t_y(B) + v\,t_y(C) \end{cases}$$





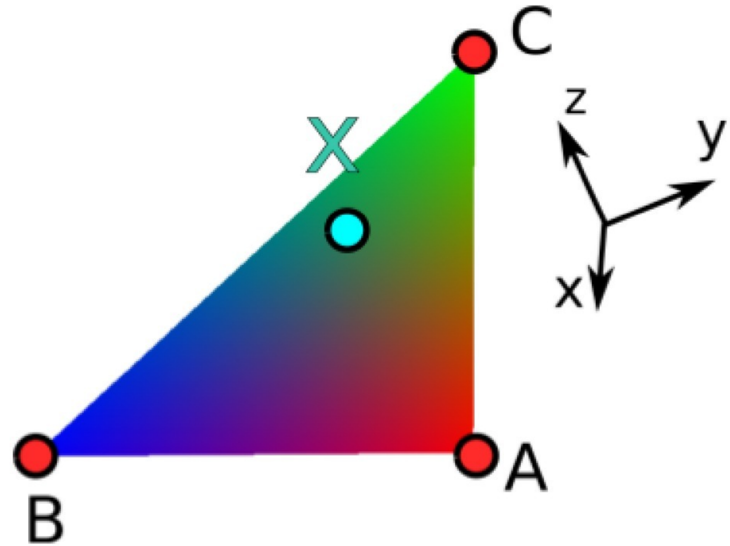
Barycentric coordinates

Given a point $\mathbf{p} = (x, y, z) \in \mathbb{R}^3$

How to know (α, β, γ) such that $\mathbf{p} = \alpha \mathbf{p}_A + \beta \mathbf{p}_B + \gamma \mathbf{p}_C$

$$\alpha + \beta + \gamma = 1$$

$$\begin{cases} A = \text{area}(\mathbf{x}_B - \mathbf{x}_A, \mathbf{x}_C - \mathbf{x}_A) \\ A_1 = \text{area}(\mathbf{x}_C - \mathbf{x}_B, \mathbf{x} - \mathbf{x}_B) \\ A_2 = \text{area}(\mathbf{x}_A - \mathbf{x}_C, \mathbf{x} - \mathbf{x}_C) \\ A_3 = \text{area}(\mathbf{x}_B - \mathbf{x}_A, \mathbf{x} - \mathbf{x}_A) \end{cases}$$



with $\text{area}(\mathbf{v}_0, \mathbf{v}_1) = 1/2 \|\mathbf{v}_0 \times \mathbf{v}_1\|$

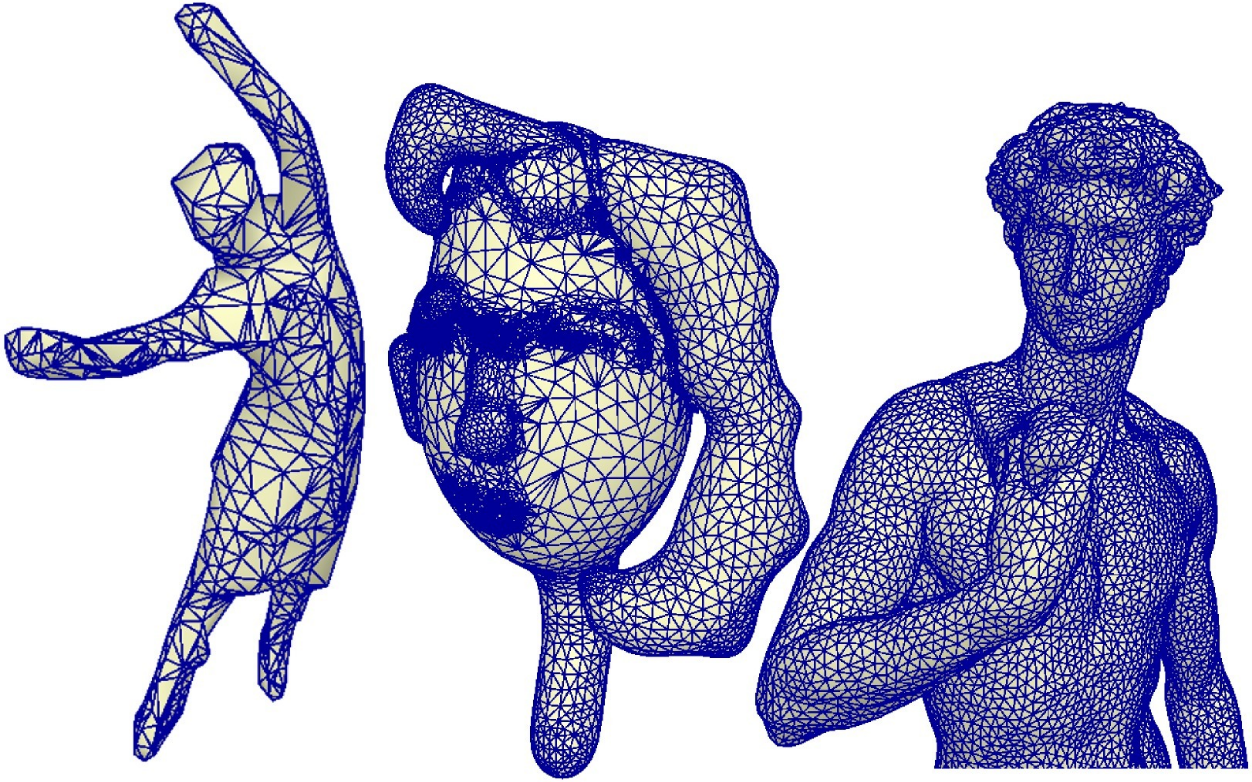
$$\Rightarrow \begin{cases} \alpha = A_1 / A \\ \beta = A_2 / A \\ \gamma = A_3 / A \end{cases}$$

Mesh quality

Triangulation $\theta_{\min} \simeq 60^\circ$

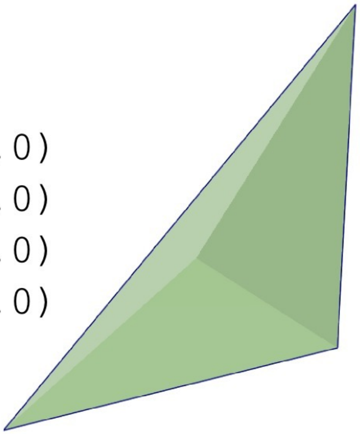
Quads $\theta_{\min} \simeq 90^\circ$

Application: Computing (FEM), Rendering



Mesh data structure

How to encode a tetrahedron

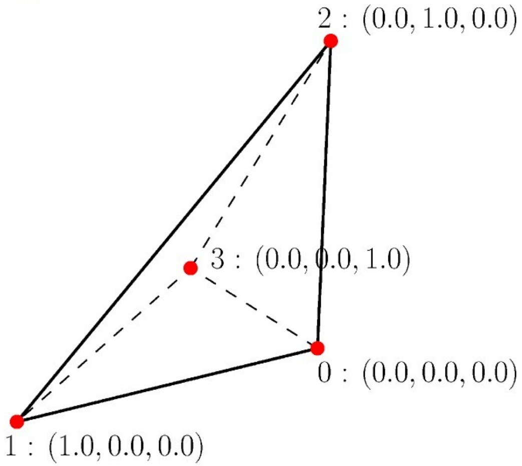


1st solution:

```
(0.0,0.0,0.0),(1.0,0.0,0.0),(0.0,0.0,1.0)
(0.0,0.0,0.0),(0.0,0.0,1.0),(0.0,1.0,0.0)
(0.0,0.0,0.0),(0.0,1.0,0.0),(1.0,0.0,0.0)
(0.0,1.0,0.0),(0.0,0.0,1.0),(1.0,0.0,0.0)
```

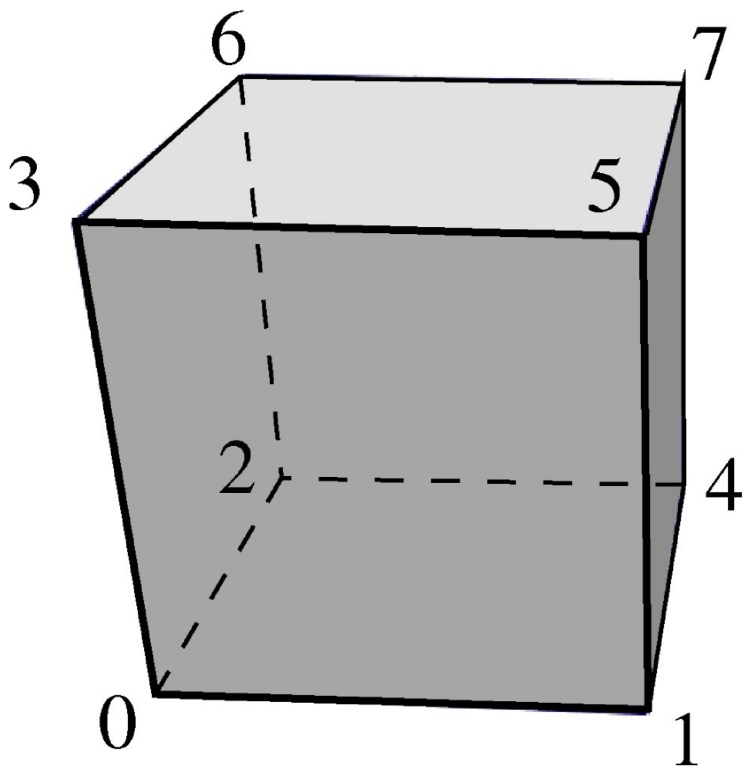
2nd solution:

```
Geometry:
(0,0,0),(1,0,0),(0,1,0),(0,0,1)
Connectivity
(0,1,3)
(0,3,2)
(0,2,1)
(1,2,3)
```



Example of file format: off

```
OFF
8 6 12
0 0 0
1 0 0
0 1 0
0 0 1
1 1 0
1 0 1
0 1 1
1 1 1
4 0 1 4 2
4 1 5 7 4
4 3 6 7 5
4 2 6 3 0
4 2 4 7 6
4 0 3 5 1
```



Data structure

Contiguous data in memory
=> Fast drawing using GPU

```
//(x0,y0,z0,x1,y1,z1,...)
std::vector <double> vertex

//(i00,i01,i02,i10,i11,i12,...)
std::vector <int> connectivity

std::vector <double> normal, color, texture ...
```

Access to the y coordinate of the vertex k

`vertex[3*k+1]`

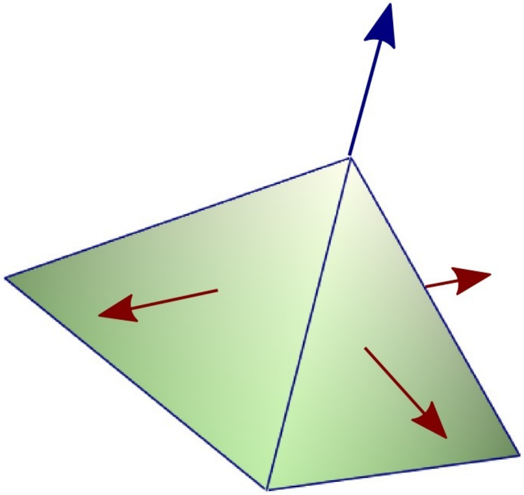
Access to the y coordinate of the vertex s (0,1,2) of triangle t

`vertex[3*connectivity[3*t+s]+1]`

Normals to a mesh

Smooth aspect

=> 1 normal per vertex

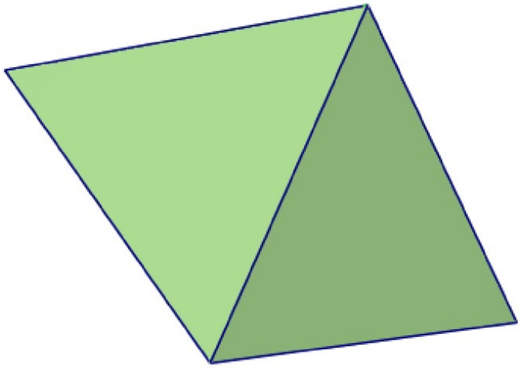


Average of normals

(wrong but commonly used)

$$n_k = \sum_{i \in \mathcal{V}(k)} n_i$$

$$n'_k = n_k / \|n_k\|$$



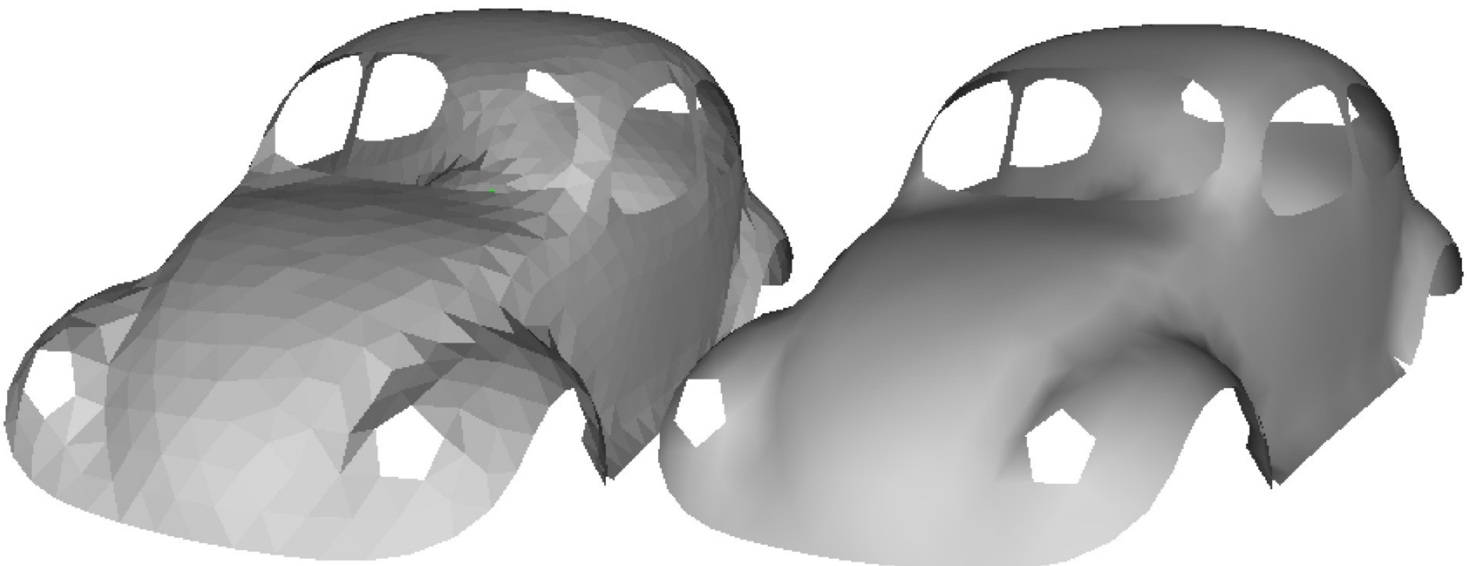
k: vertex index

i: face index

$\mathcal{V}(k)$: neighboring face of vertex k

Normals to a mesh

In OpenGL: One normal per vertex interpolated over the face.

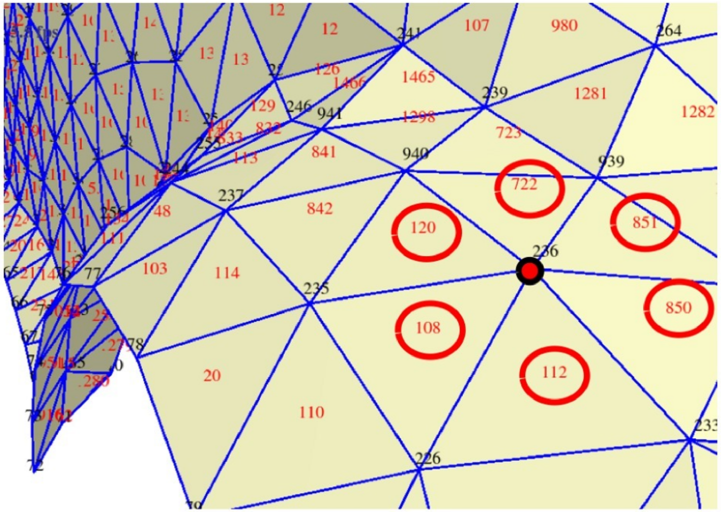


per face normal *per vertex normal*

Data structure: neighbors

1-ring = Vertices neighbors of a given vertex

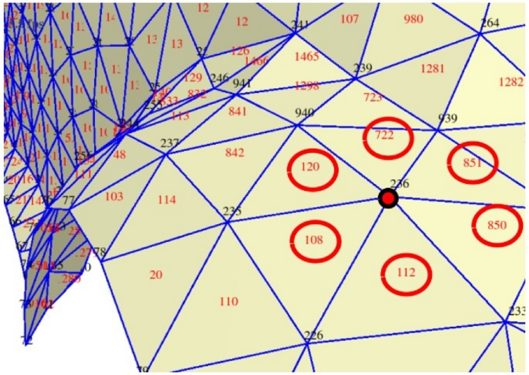
```
std::vector <std::vector <int> > one_ring
// example for the cube:
one_ring[0] = [1,2,3]
one_ring[1] = [5,4,0]
...
```



Data structure: neighbors

Neighboring triangles of another triangle

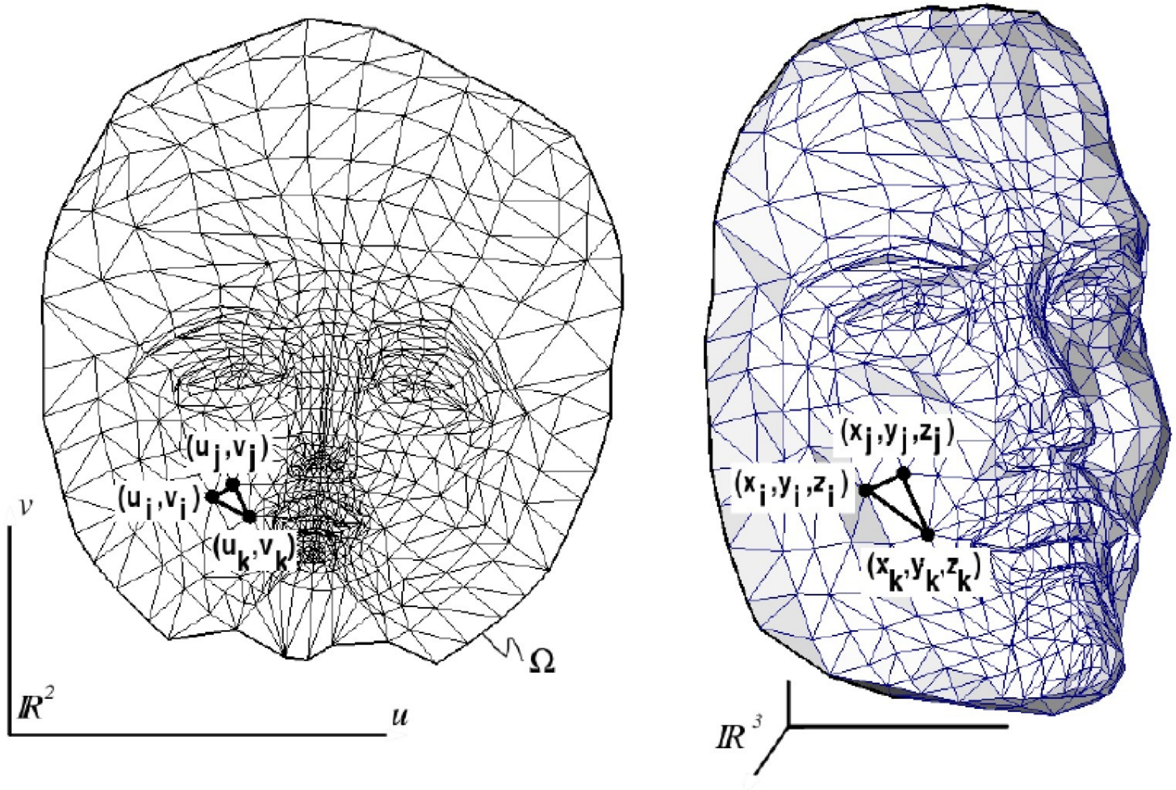
Neighboring triangles of a vertex (1-star)
=> Used to compute normals



Care is needed to the data structure:
Compromize between: access time, search time, memory consumption, easyness

Parameterization / Textures

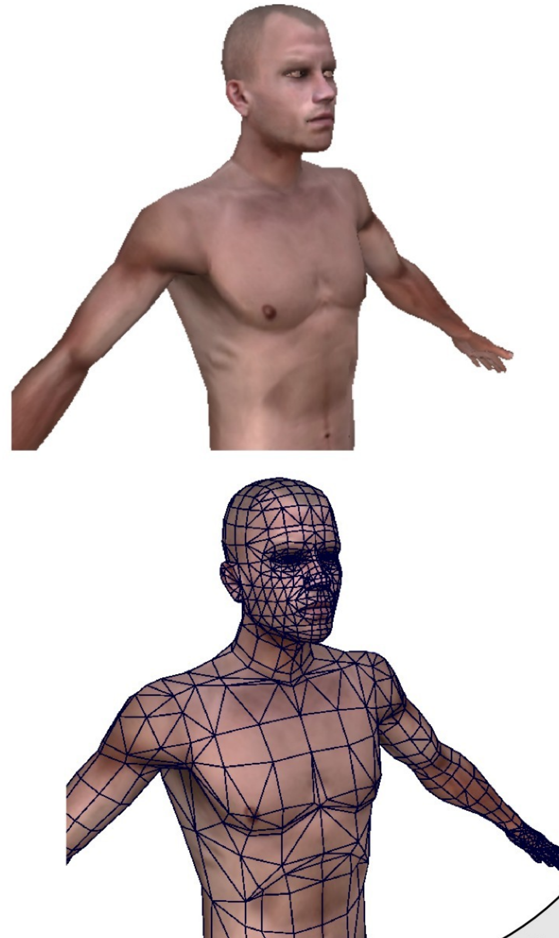
Parametrization of a mesh = construct S (piecewise) given Γ



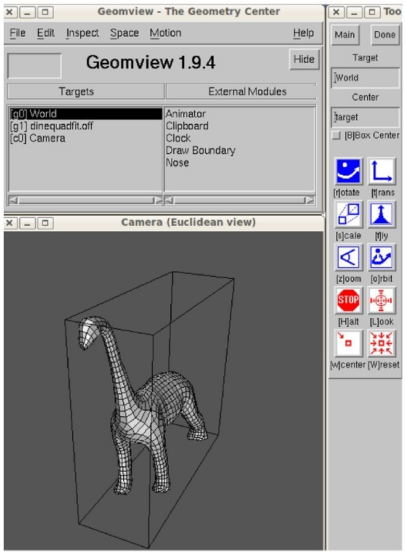
[Botsh, Pauly, Kobbelt, Alliez, SIGGRAPH Course Notes 2007]

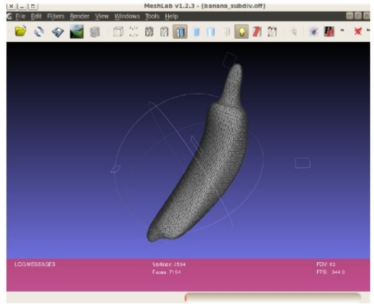
Textures

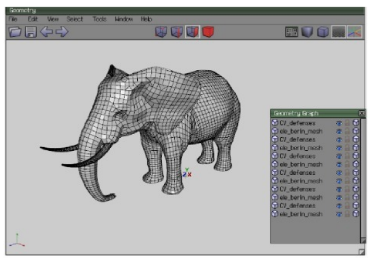
In practice: **Charts**(pieces with overlays)

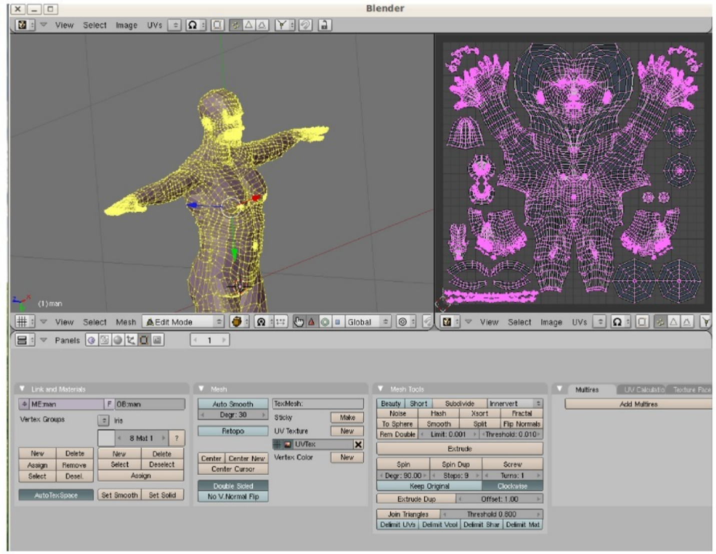


Softwares









Geomview (Viewer)
Meshlab (Mesh Processing)
Wings 3D (Subdivision)
Blender (Artists)

Meshes

Topological caracteristic
Data structure
Subdivision
Mesh Smoothing

Euler Poincaré caracteristic

Mesh with N_f faces, N_v vertices, N_e edges

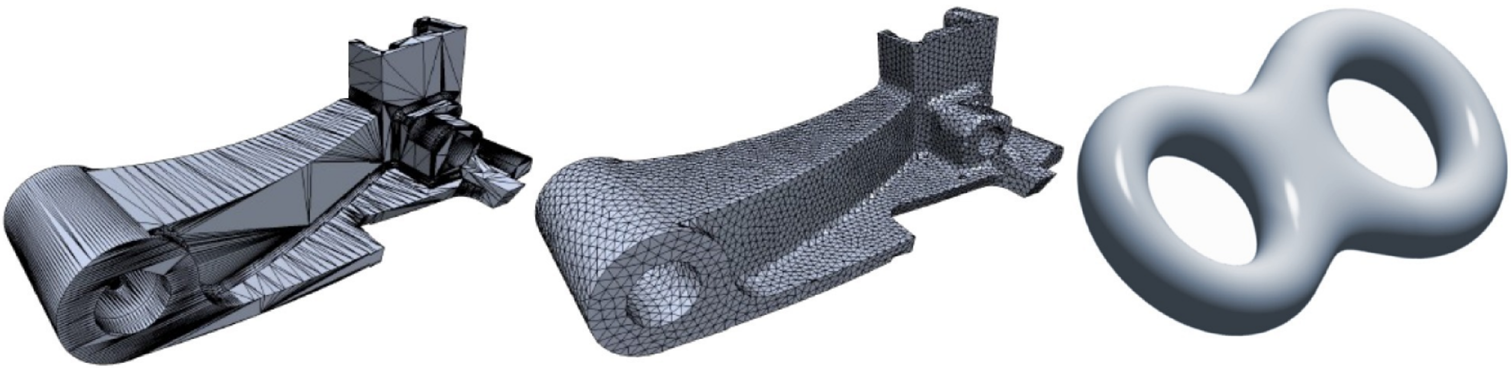
$$N_s - N_a + N_f = \chi = 2(c - g) - b$$

χ: Euler caracteristic (Gauss-Bonnet theorem)

c: Number of connex components

g: Number of holes (topological genus)

b: Number of boundaries



[Wikipedia]

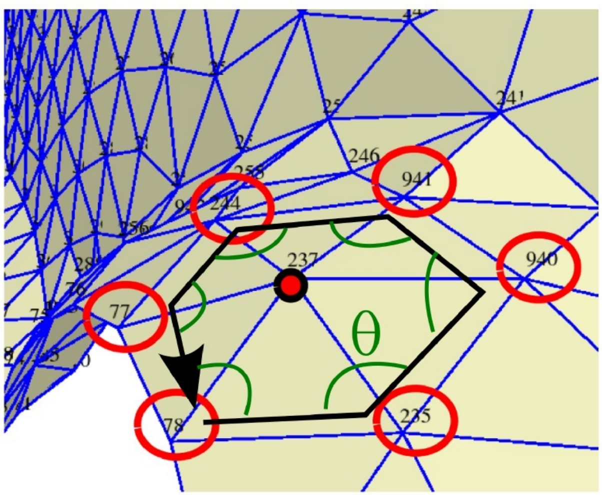
Data structure

Encoding geometry + connectivity

indices

+ Fast rendering
+ Generic
+ Simple

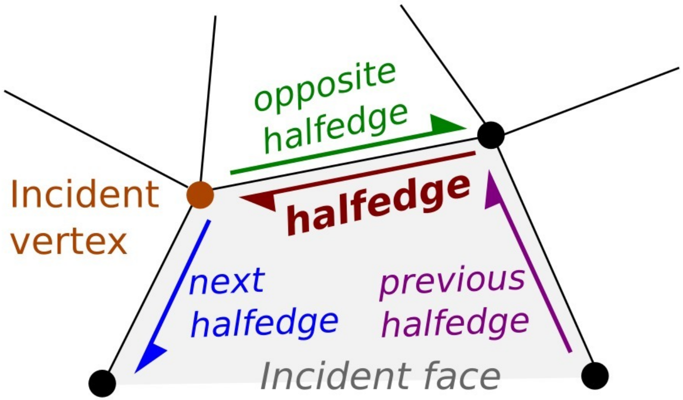
- No neighboring info
- Add/delete in O(N)




```
OFF
40 95 75
-0.175114 -0.047799 -0.046492
-0.199566 0.730914 -0.064795
-0.010689 0.674496 0.008900
-0.015538 0.153071 0.107408
-0.070148 0.767894 -0.116107
-0.053836 -0.702815 0.109714
-0.162416 -0.785481 0.088014
-0.112365 -0.782492 0.135482
-0.240928 0.031451 0.031966
-0.259289 0.200557 0.035420
0.296891 -0.707385 0.143375
-0.190129 -0.069002 0.109358
-0.010148 0.024179 -0.067283
-0.112968 -0.089127 0.092391
-0.185828 0.377372 -0.111155
3 20 4 1
3 34 11 13
3 12 30 0
3 30 13 17
3 23 22 21
3 29 38 17
3 32 0 13
3 14 0 37
3 24 4 21
3 14 32 1
3 24 2 22
3 3 12 25
3 4 24 15
3 21 15 26
3 35 34 13
3 19 32 13
3 19 13 27
```

Halfedge data structure

Store successive (half) edges
Encode the edges:
  Faces can be *reconstructed* along the path (only for 2-manifold)
Addition/suppression in O(1)

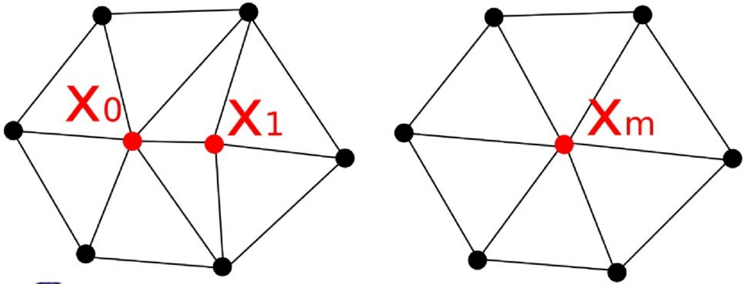


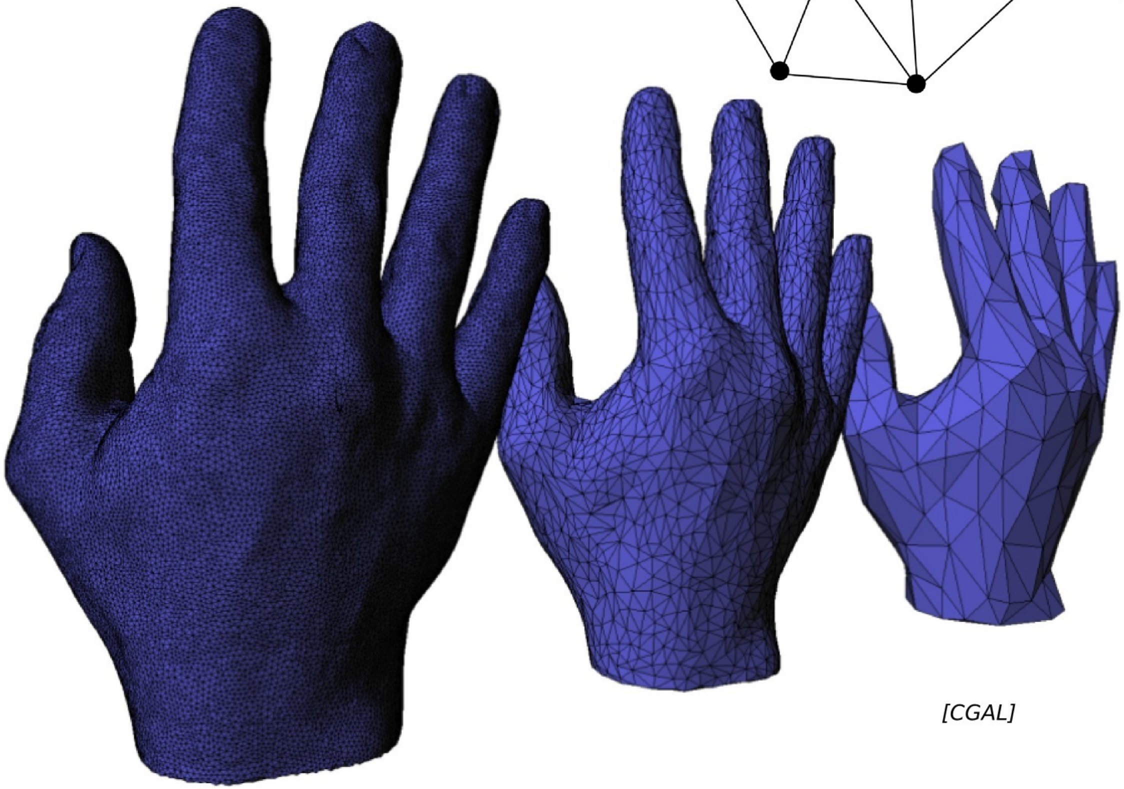
Halfedge

```
Halfedge opposite();
Halfedge next();
Halfedge prev();

Vertex vertex();
Face face();
```

Edge collapse

Delete an edge (base operation of mesh simplification)





[CGAL]

Best data structure

Find a suitable compromise

Contiguous indices in array

+Simple, general, adapted to GPU
+ Random access O(1)
- Neighborood access in O(N)
- Add/delete in O(N)

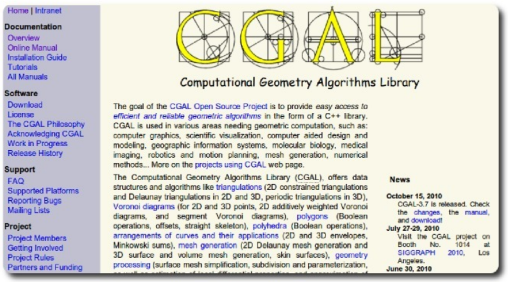
Halfedge in list

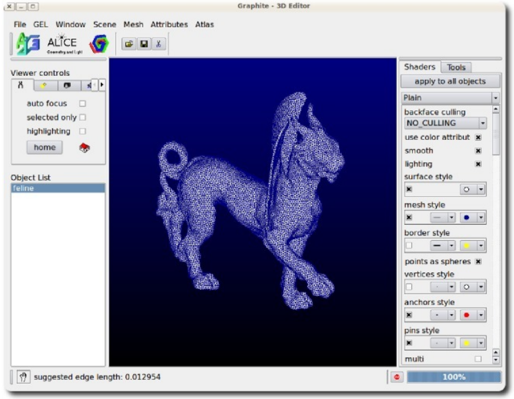
+ Neighbor in O(1)
+ Add/Delete in O(1)
- More complex structure
- Only for 2-manifold
- Non contiguous in memory
- Random access in O(N)

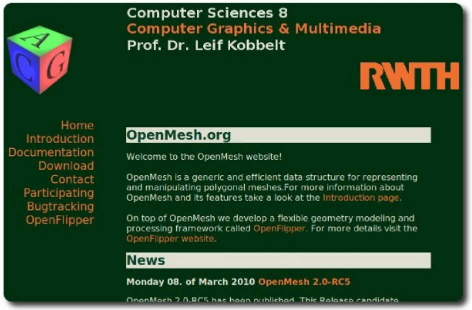
```
Halfedge_handle g = h->next()->opposite()->next();
P.split_edge( h->next());
P.split_edge( g->next());
P.split_edge( g);
h->next()->vertex()->point()     = Point( 1, 0, 1);
g->next()->vertex()->point()     = Point( 0, 1, 1);
g->opposite()->vertex()->point() = Point( 1, 1, 0);
Halfedge_handle f = P.split_facet( g->next(),
                                   g->next()->next()->next());
Halfedge_handle e = P.split_edge( f);
e->vertex()->point() = Point( 1, 1, 1);
P.split_facet( e, f->next()->next());
```

[CGAL]

Lib implementing Halfedge DS

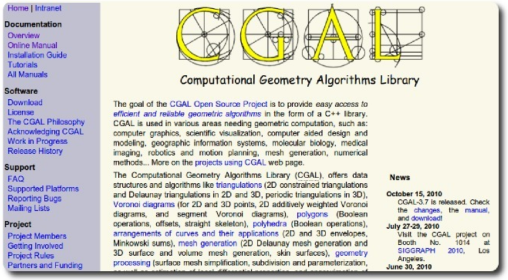


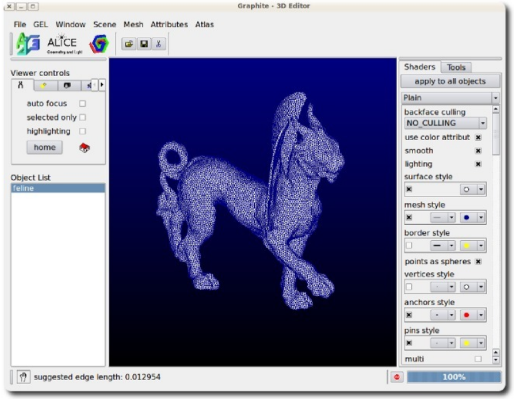


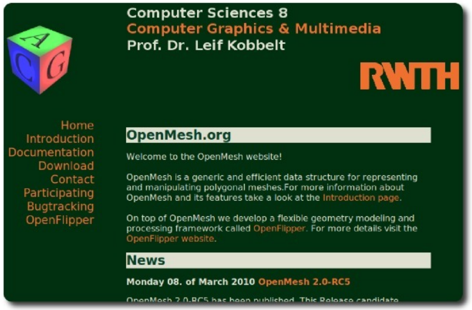


CGAL (C++ complex, exact computation, lot of algorithms)
Graphite (remeshing, parameterization, GUI)
OpenMesh (more simple than CGAL, less algorithms)

Lib implementing Halfedge DS







CGAL (C++ complex, exact computation, lot of algorithms)
Graphite (remeshing, parameterization, GUI)
OpenMesh (more simple than CGAL, less algorithms)

Short introduction to CGAL

Short introduction to CGAL

```
#include <CGAL/Cartesian.h>


// contains informations about types
//   in c++ : a traits class
typedef CGAL::Cartesian<float> Kernel;

int main()
{
    Kernel::Vector_3 x0(1.0f,2.0f,3.0f);
    Kernel::Vector_3 x1(2.0f,1.2f,5.2f);

    std::cout<<x0<<" , "<<x1<<" , "<<x0+x1<<std::endl;

    return 0;
}
```

Generate a mesh in CGAL

```
#include <CGAL/Cartesian.h>

//maillage 3D
#include <CGAL/Polyhedron_3.h>

typedef CGAL::Cartesian<double> Kernel;
typedef CGAL::Polyhedron_3<Kernel> Polyhedron;

int main()
{
    Kernel::Point_3 p0(0.0,0.0,0.0);
    Kernel::Point_3 p1(1.0,0.0,0.0);
    Kernel::Point_3 p2(0.0,1.0,0.0);
    Kernel::Point_3 p3(0.0,0.0,1.0);


    Polyhedron mesh;
    mesh.make_tetrahedron(p0,p1,p2,p3);


    Polyhedron::Vertex_iterator it=mesh.vertices_begin();
    Polyhedron::Vertex_iterator it_end=mesh.vertices_end();
    for(;it!=it_end;++it)
    {
        Polyhedron::Point_3 p=it->point();
        std::cout<<p<<std::endl;
    }

    return 0;
}
```

Generate a mesh in CGAL

```
#include <CGAL/Cartesian.h>

//maillage 3D
#include <CGAL/Polyhedron_3.h>

typedef CGAL::Cartesian<double> Kernel;
typedef CGAL::Polyhedron_3<Kernel> Polyhedron;

int main()
{
    Kernel::Point_3 p0(0.0,0.0,0.0);
    Kernel::Point_3 p1(1.0,0.0,0.0);
    Kernel::Point_3 p2(0.0,1.0,0.0);
    Kernel::Point_3 p3(0.0,0.0,1.0);


    Polyhedron mesh;
    mesh.make_tetrahedron(p0,p1,p2,p3);

    auto it=mesh.vertices_begin();
    auto it_end=mesh.vertices_end();
    for(;it!=it_end;++it)
    {
        const auto p=it->point();
        std::cout<<p<<std::endl;
    }

    return 0;
}
```

Load a off file in CGAL

```cpp
#include <iostream>
#include <fstream>
#include <CGAL/Cartesian.h>
#include <CGAL/Polyhedron_3.h>
#include <CGAL/IO/Polyhedron_iostream.h>

typedef CGAL::Cartesian<double> Kernel;
typedef CGAL::Polyhedron_3<Kernel> Polyhedron;

int main()
{
    Polyhedron mesh;
    std::ifstream stream("../cgal_4/cube.off");
    stream>>mesh;

    std::cout<<"N_vertices = "<<mesh.size_of_vertices()<<std::endl;
    std::cout<<"N_faces = "<<mesh.size_of_facets()<<std::endl;
    std::cout<<"N_halfedges = "<<mesh.size_of_halfedges()<<std::endl;

    auto it=mesh.vertices_begin();
    auto it_end=mesh.vertices_end();
    for(;it!=it_end;++it)
    {
        const auto p=it->point();
        std::cout<<p<<std::endl;
    }

    return 0;
}
```

Manipulate edges in CGAL

```
typedef CGAL::Cartesian<double> Kernel;
typedef CGAL::Polyhedron_3<Kernel> Polyhedron;

int main()
{
    Polyhedron mesh;
    std::ifstream stream("../cgal_5/cube.off");
    stream>>mesh;

    Polyhedron::Halfedge_handle halfedge=mesh.halfedges_begin();

    const auto p0=halfedge->vertex()->point();
    const auto p1=halfedge->opposite()->vertex()->point();

    std::cout<<"edge 1 : ["<<p0<<","<<p1<<"]"<<std::endl;

    halfedge=halfedge->next();

    const auto p2=halfedge->vertex()->point();
    const auto p3=halfedge->opposite()->vertex()->point();

    std::cout<<"edge 2 : ["<<p2<<","<<p3<<"]"<<std::endl;


    return 0;
}
```

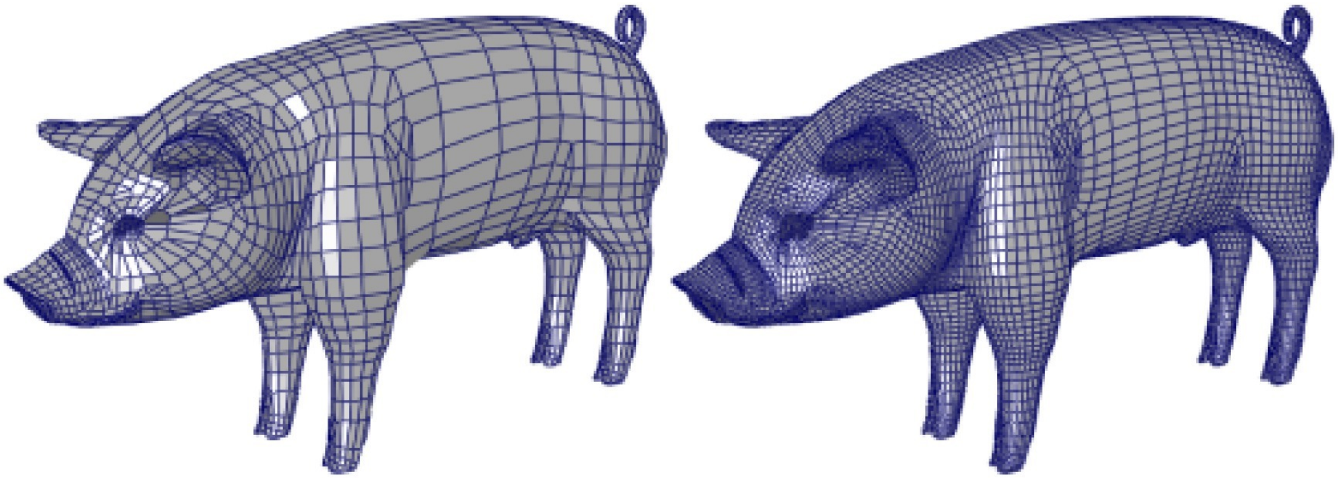
Travel through a mesh in CGAL

```
typedef CGAL::Cartesian<double> Kernel;
typedef CGAL::Polyhedron_3<Kernel> Polyhedron;

int main()
{
    Polyhedron mesh;
    std::ifstream stream("../cgal_5/cube.off");
    stream>>mesh;

    auto it_face=mesh.facets_begin();
    auto it_face_end=mesh.facets_end();

    int face_number=0;
    for(;it_face!=it_face_end;++it_face)
    {
        std::cout<<"FACE : "<<face_number<<std::endl;

        auto halfedge=it_face->halfedge();
        const auto halfedge_end=halfedge;
        do
        {
            const auto p=halfedge->vertex()->point();
            std::cout<<p<<std::endl;
            halfedge=halfedge->next();

        }while(halfedge!=halfedge_end);

        face_number++;
    }

    return 0;
}
```

Subdivision

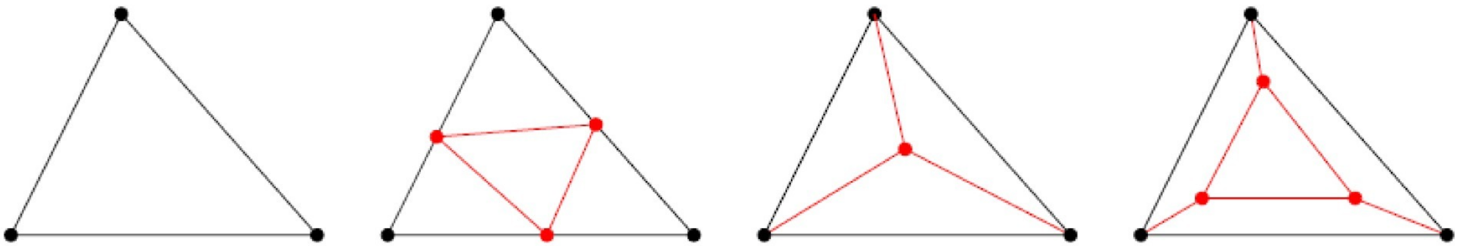
Mesh subdivision

Subdivide for rendering, computing, deforming, ...



Mesh subdivision

Subdivision of the connectivity
Several possibilities of subdivision

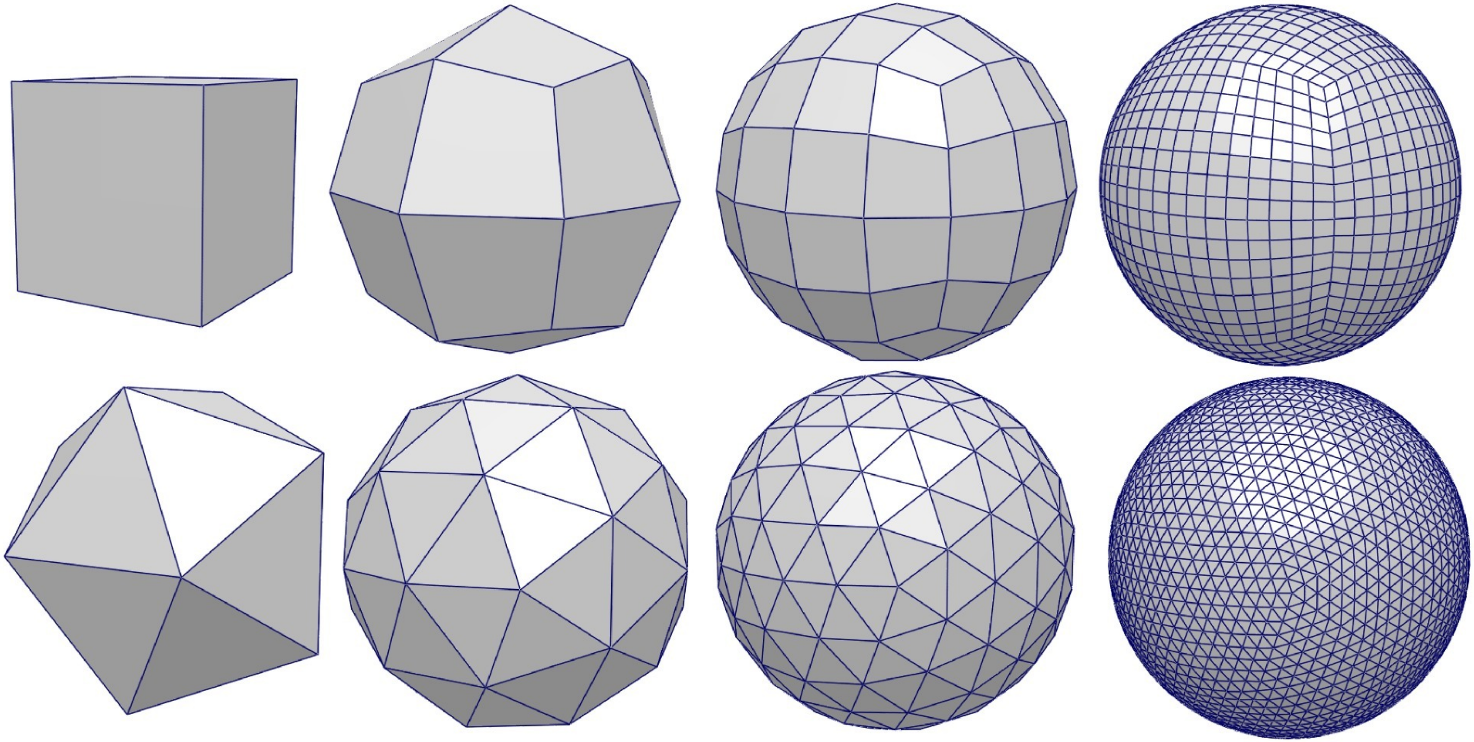


=> can generate arbitrary polygons

Two main approches:

- Interpolating schemes
- Approximating schemes

Application to sphere subdivision



high quality triangulation

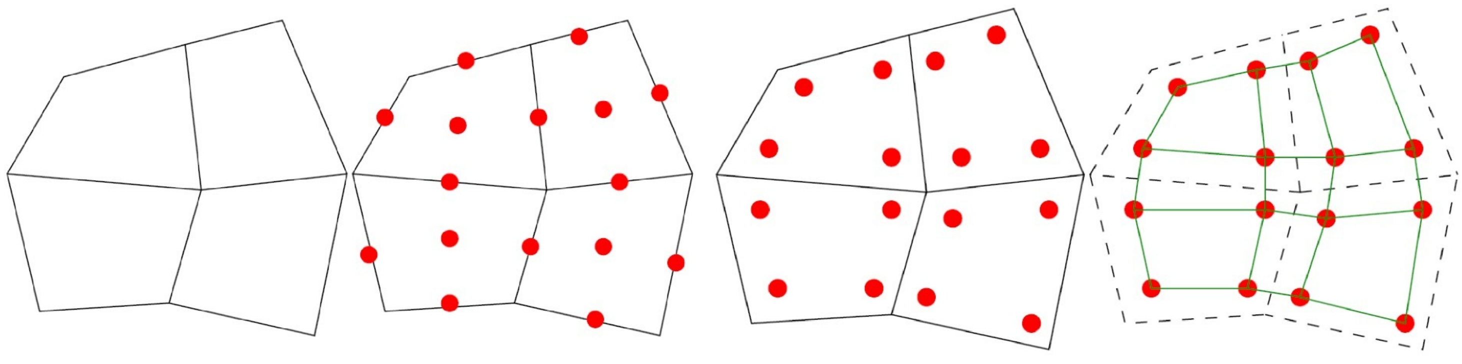
Doo-Sabin subdivision

Given a face $(\mathbf{p}_i)_{i=[\![0,N-1]\!]}$

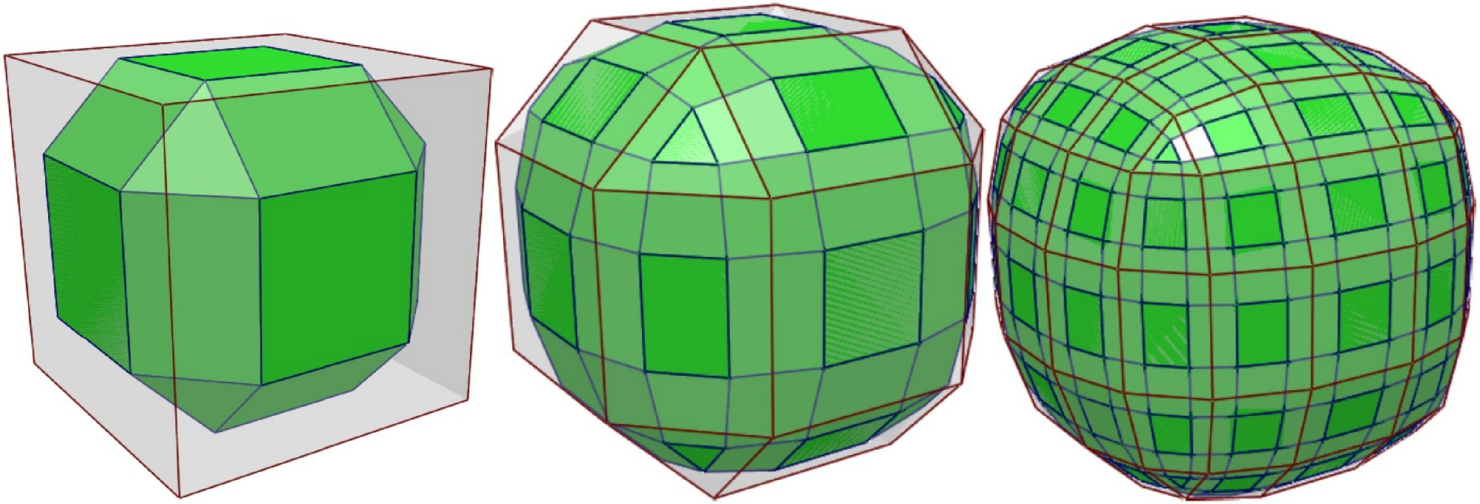
Compute middle vertex $m_i = (\mathbf{p}_i + \mathbf{p}_{i+1})/2$

Compute the barycenter of the face $\mathbf{b} = \frac{1}{N}\sum_{i=0}^{N} \mathbf{p}_i$

The new vertices are $\mathbf{n}_i = (\mathbf{p}_i + \mathbf{m}_i + \mathbf{m}_{i-1} + \mathbf{b})/4$



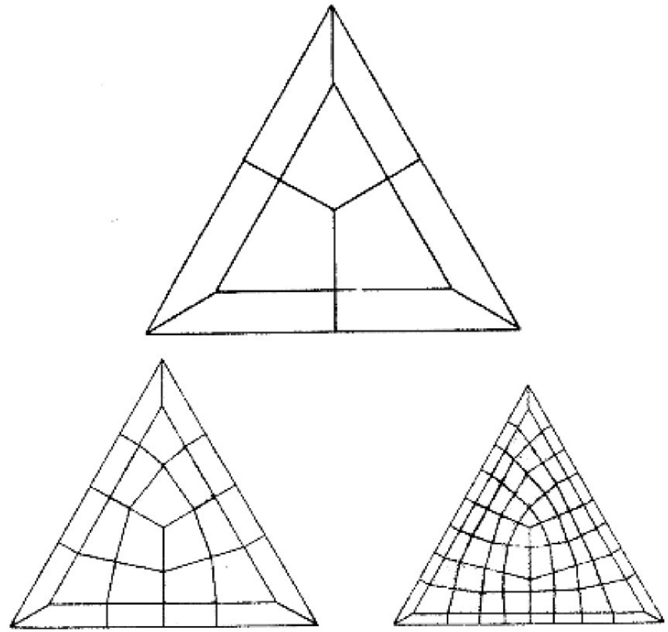
Doo-Sabin subdivision on a mesh



Catmull-Clark subdivision

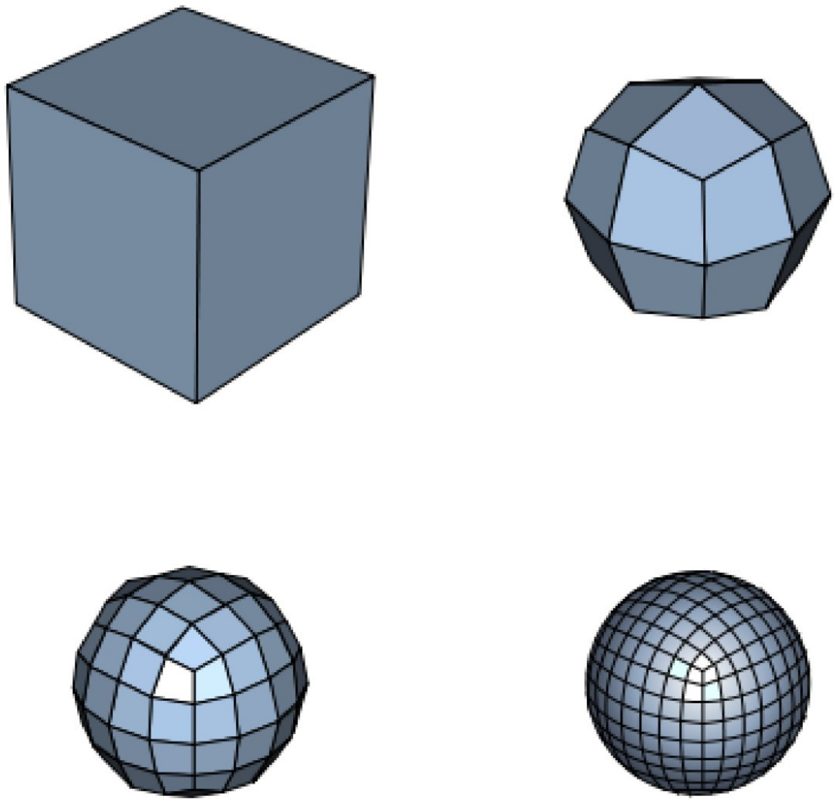
Barycenter of the face
   => new face vertex
Barycenter of the vertices + middle of face sharing the edge
   => new edge vertex
New vertex position : (Q+2R+S(n-3))/n

Q: Average of the face vertex
R: Average of the edge vertex
S: Old vertex
n: Valence



Catmull-Clark subdivision

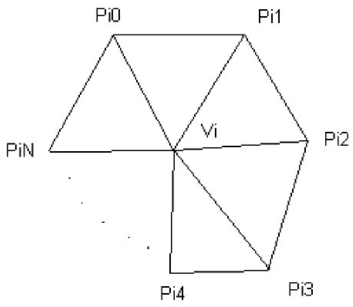
C^2 excepted at the extraordinary vertices
Approximation scheme
Face subdivision
Quad prefered



Loop subdivision

Triangular meshes

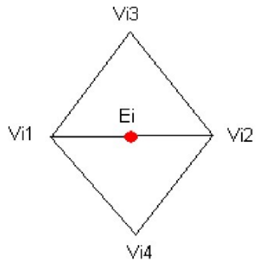
New vertex



$$V^{i+1} = (1 - n\alpha)V^i + \alpha \sum_{k=0}^{n} P_k$$

$$\alpha = \frac{1}{n}\left(\frac{5}{8} - \left(\frac{3}{8} - \frac{1}{4}\cos\left(\frac{2\pi}{n}\right)\right)^2\right)$$

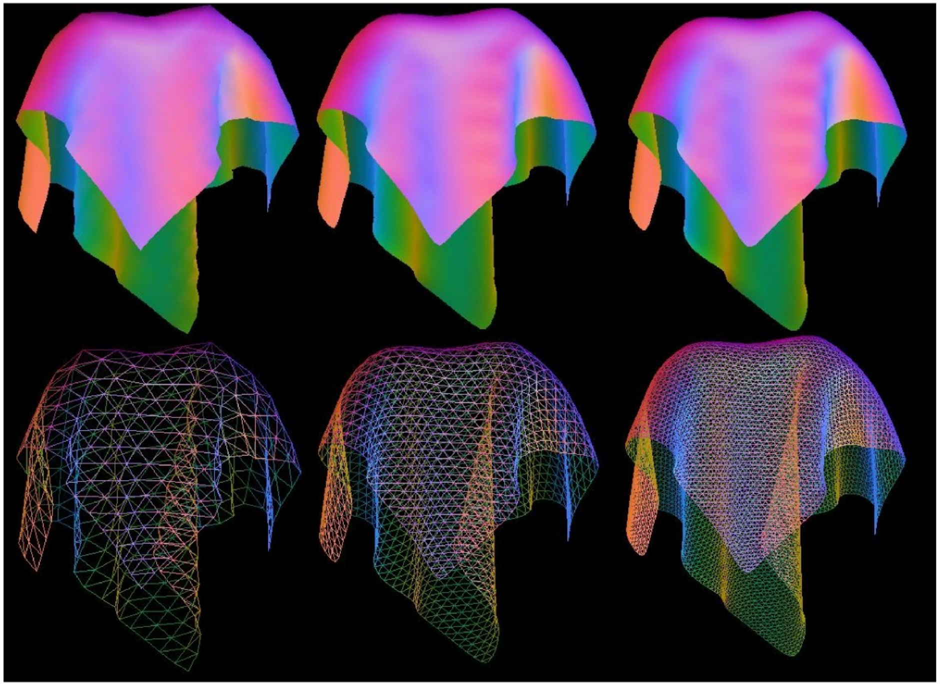
Edge vertex



$$E^{i+1} = \frac{3}{8}(V_1 + V_2) + \frac{1}{8}(V_3 + V_4)$$

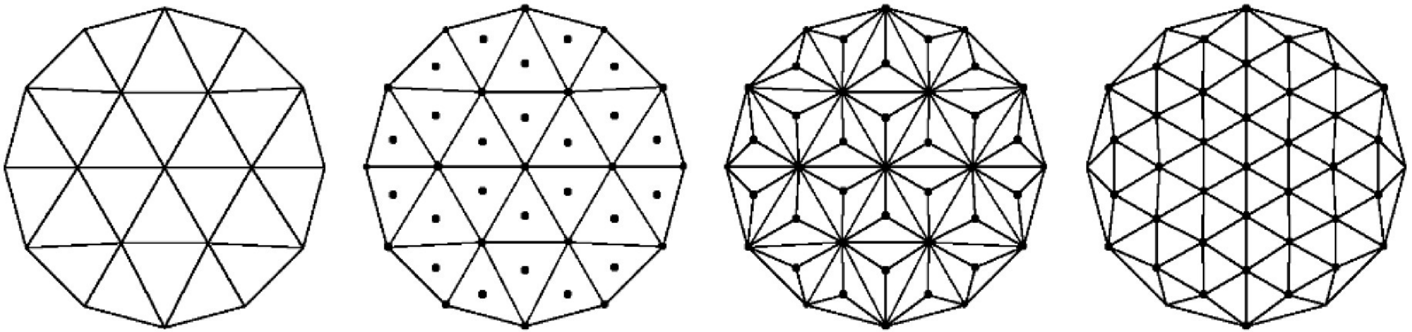
Loop subdivision

C^2 excepted for extraordinary vertices
Approximation scheme
Face subdivision
Triangle subdivision



$\sqrt{3}$ Kobbelt subdivision

Triangular mesh



New vertices : barycenter of the old face
Change position of previous vertex:

$$(1-\alpha_n)\mathbf{p}_i + \frac{\alpha_n}{n}\sum_{j\in\mathcal{V}_i}\mathbf{p}_j$$

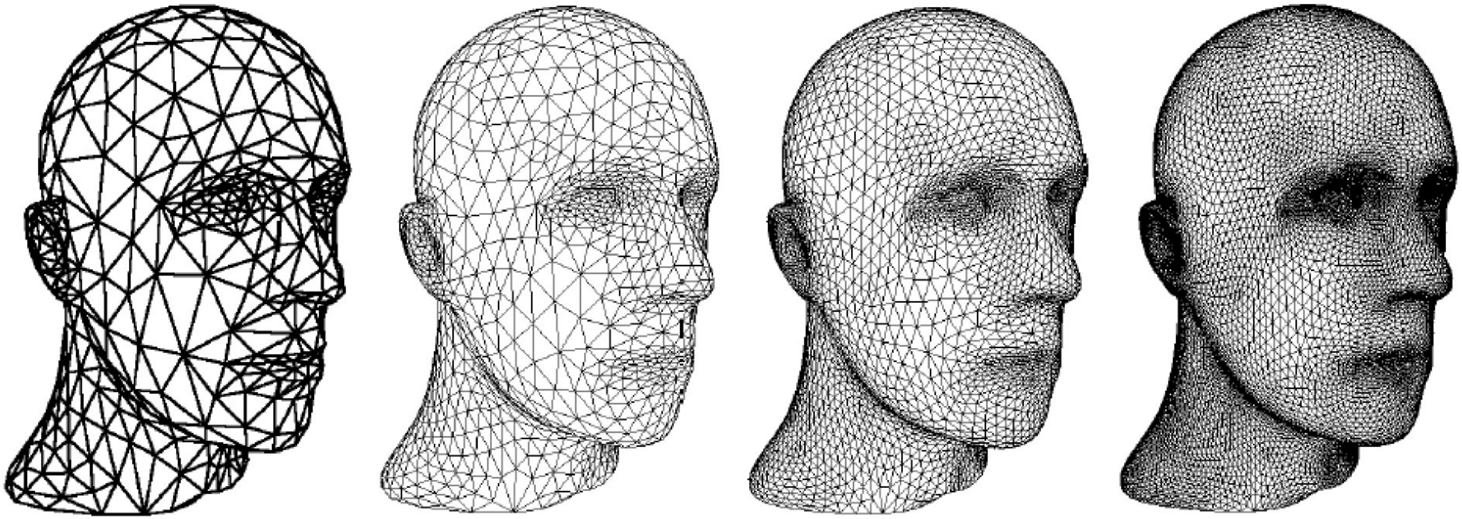
$$\alpha_n = \frac{1}{9}\left(4 - 2\cos\left(\frac{2\pi}{n}\right)\right)$$

n : valence

$\mathcal{V}$: neighbor

$\sqrt{3}$ Kobbelt subdivision

C^2 excepted at the extraordinary vertices
Approximation scheme
Face subdivision
Triangle subdivision

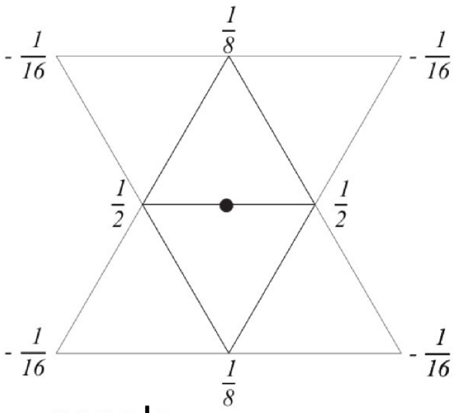


[Kobbelt, SIGGRAPH 00]

Butterfly

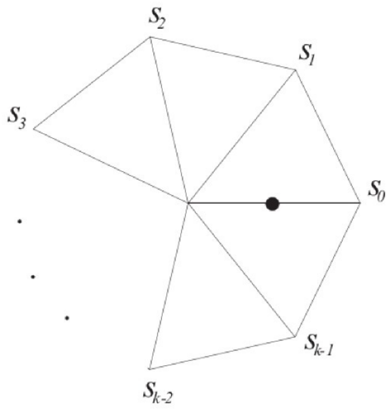
Triangular mesh

Regular vertex



mask

Extraordinary vertex



Add one point per edge

In the case of an irregular edge:

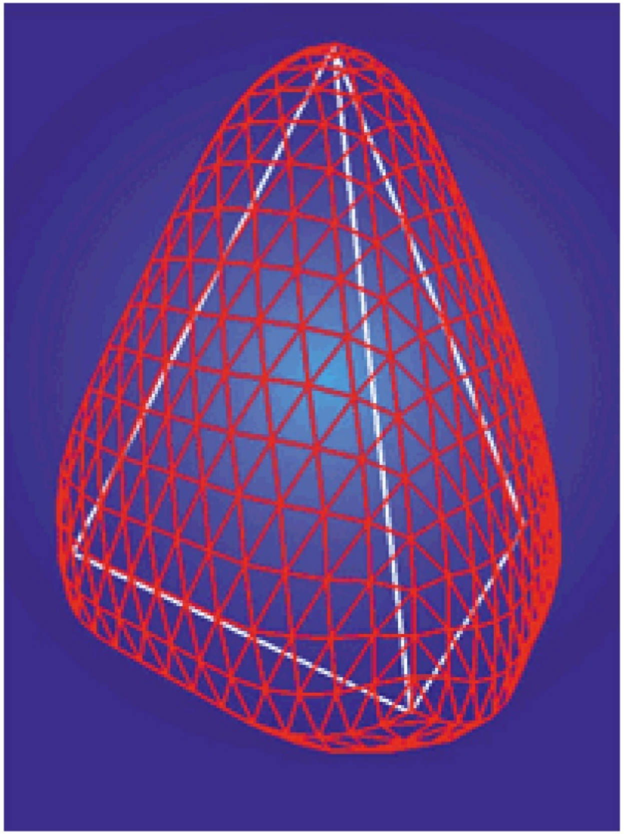
$$s_i = \frac{1}{k}\left(\frac{1}{4} + \cos\left(\frac{2\pi}{k}\right) + \frac{1}{2}\cos\left(\frac{4\pi}{k}\right)\right) \quad , k>5$$

$$s_0 = \frac{3}{8}, s_{1,3} = 0, s_2 = \frac{1}{8} \quad , k=4$$

$$s_0 = \frac{5}{12}, s_{1,2} = -\frac{1}{12} \quad , k=3$$

Butterfly

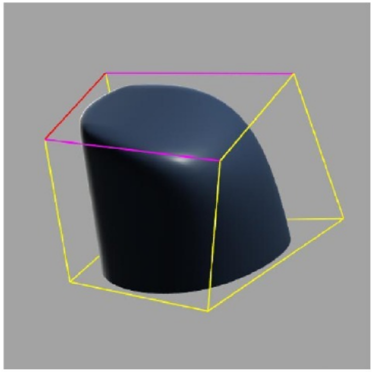
C^1 excepted at the extraordinary vertices
Interpolation scheme
Face subdivision
Triangle subdivision



Sharp edge in subdivision

Do not smooth every edges

[De Rose, Kass, Truong. Subdivision Surfaces in Character Animation. ACM SIGGRAPH 1998]



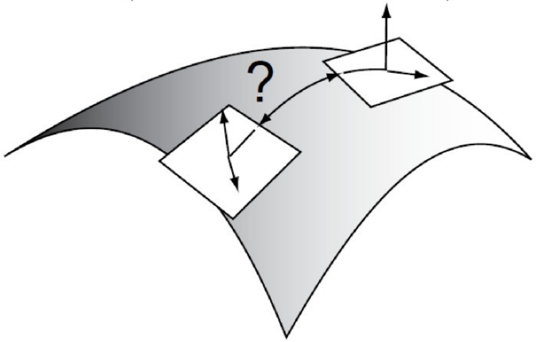
[Pixar, Gery's Game]

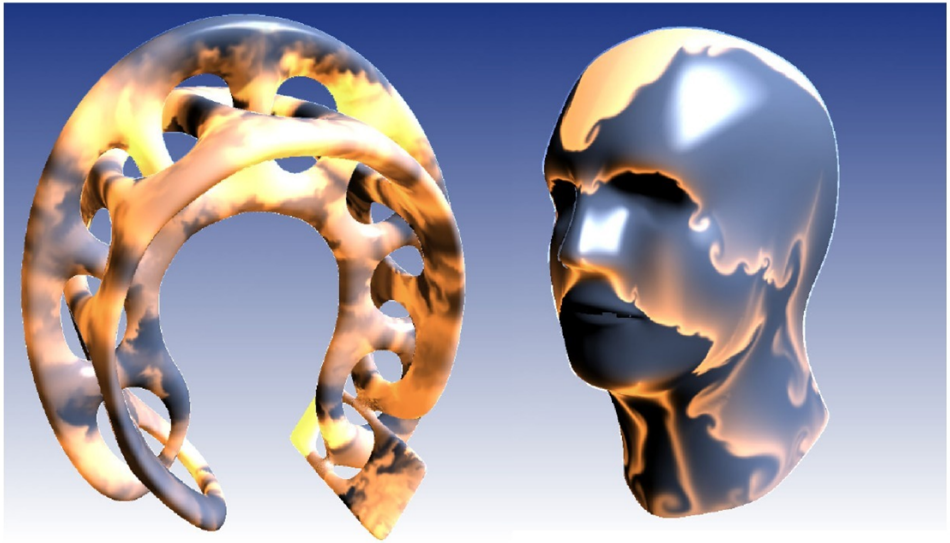
Laplacian smoothing

Laplacian smoothing

Let f be a function defined on a smooth manifold (smooth surface).

$$f : \begin{cases} \Gamma & \rightarrow & \mathbb{R}^N \\ (\xi_1, \xi_2) & \mapsto & f(\xi_1, \xi_2) \end{cases}$$





Differential geometry?
How to compare two vectors at different positions ?

Computation on manifold

Laplace Beltrami operator = Laplacian on manifold

$$\triangle = \frac{1}{\sqrt{\det(\mathrm{I}_\Gamma)}} \sum_i \frac{\partial}{\partial \xi_i} \left(\sqrt{\det(\mathrm{I}_\Gamma) \sum_j \mathrm{I}_\Gamma^{ij} \frac{\partial}{\partial \xi_j}} \right)$$

Special case: Laplace on the coordinates of the mesh

$$f : (\xi_1, \xi_2) \mapsto \mathbf{p}(\xi_1, \xi_2) = (x(\xi_1, \xi_2), y(\xi_1, \xi_2), z(\xi_1, \xi_2))$$

$\mathrm{Sp}\left(\triangle \mathbf{p}\right)$ = Eigenmode of vibrations = Fourier basis

Spectral theory of meshes

[Vallet, Levy, Eurographics 0

Laplacian smoothing

Low pass filter : Convolution of a Gaussian kernel in 2D

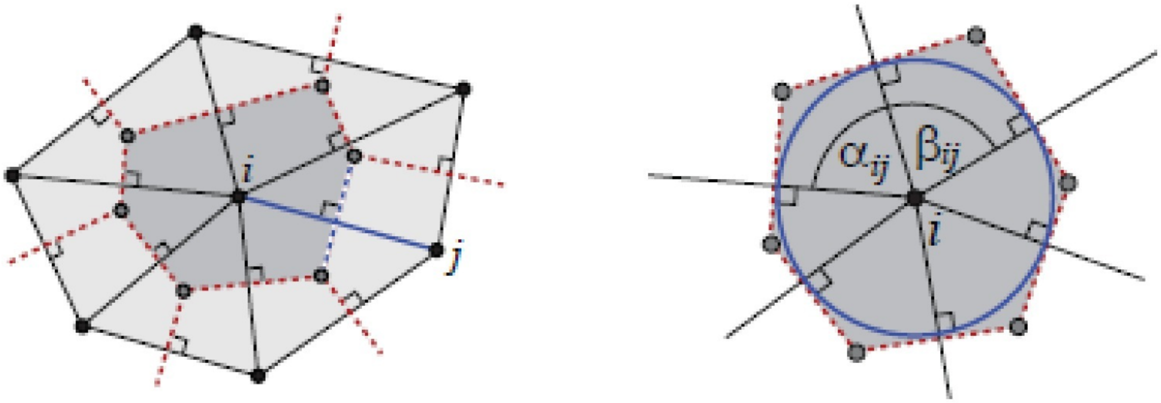
Solution of the heat equation (see Scale Space theory)

$$\frac{\partial \mathbf{p}}{\partial t} = \triangle \mathbf{p}$$

How do we approximate $\triangle$ on a mesh?

Several possibilities, none is perfect

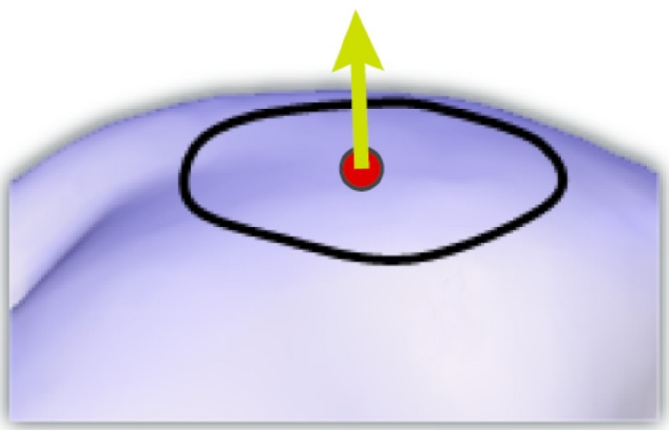
[Wardetzky, Mathur, Kalberer, Grinspun. **Discrete Laplace Operators: No Free Lunch**. SGP 07]

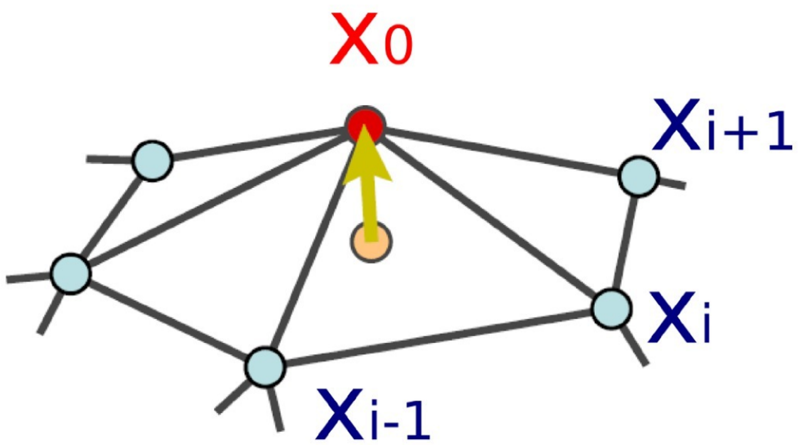


Laplacian smoothing

Simplest approximation

$$\triangle \mathbf{p}(\mathbf{p}_0) \simeq \frac{1}{N}\sum_i (\mathbf{p}_i - \mathbf{p}_0) = \mathrm{bar} - \mathbf{p}_0$$





[Sorkine, Eurographics 05]

Laplacian smoothing algorithm

$$\mathbf{p}^{k+1} = \mu\,\mathrm{bar} + (1-\mu)\,\mathbf{p}^k$$

$$\mu \in [0,1]$$