

Geometric Surface Deformation



000

Goal of surface deformation

We want to modify the coordinates

No modification of the connectivity

Topological changes = complex

```

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
    
```



001

Physically based approach ?

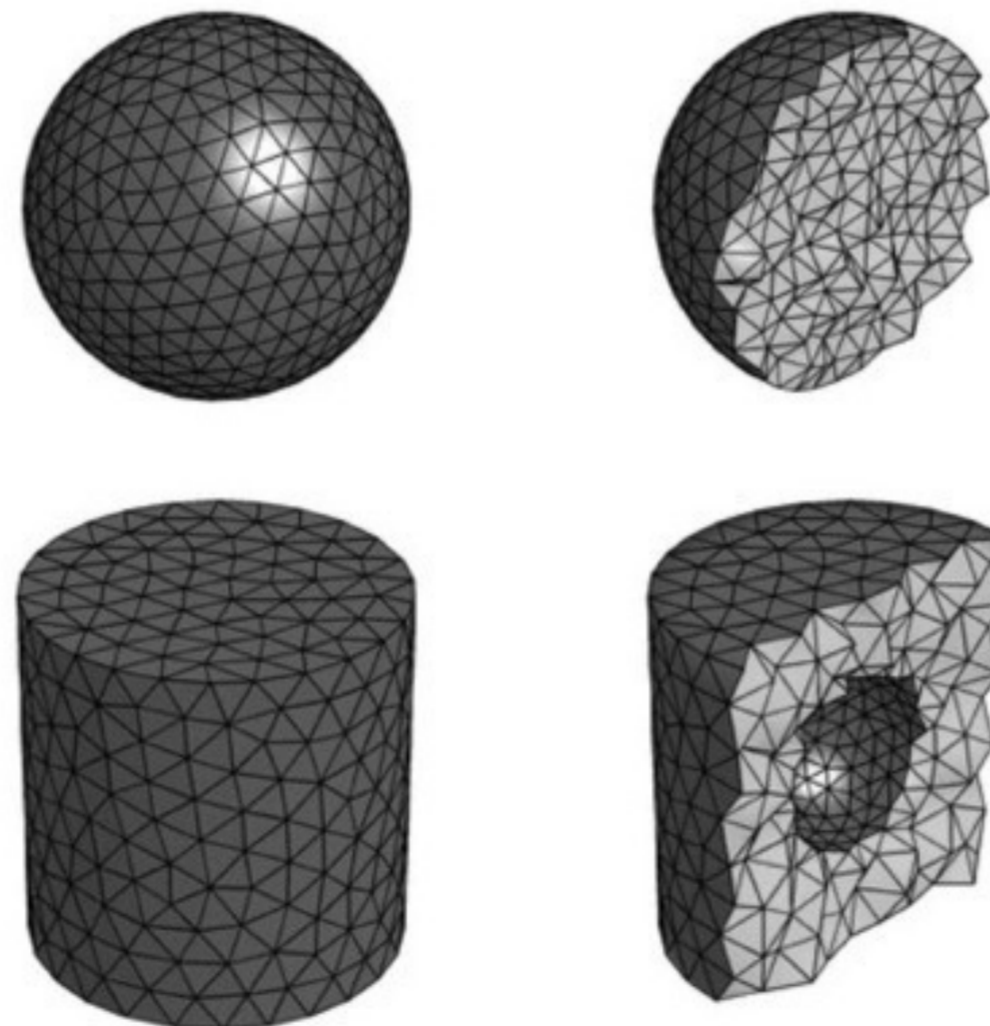
Express physical equations on volume elements
= PDE on tetrahedrons

Apply initial conditions
(forces, speed, parameters) on the volumetric mesh

Solve the equations using numerical approaches:
Inversion of large sparse matrices

We wait ...

Iterate with different parameters



002

Non physically based approaches

- + Deform as wanted the geometry
- + Goal oriented
- + Only deal with what is seen
- Potentially introduce non realistic deformations
=> geometrical constraints



003

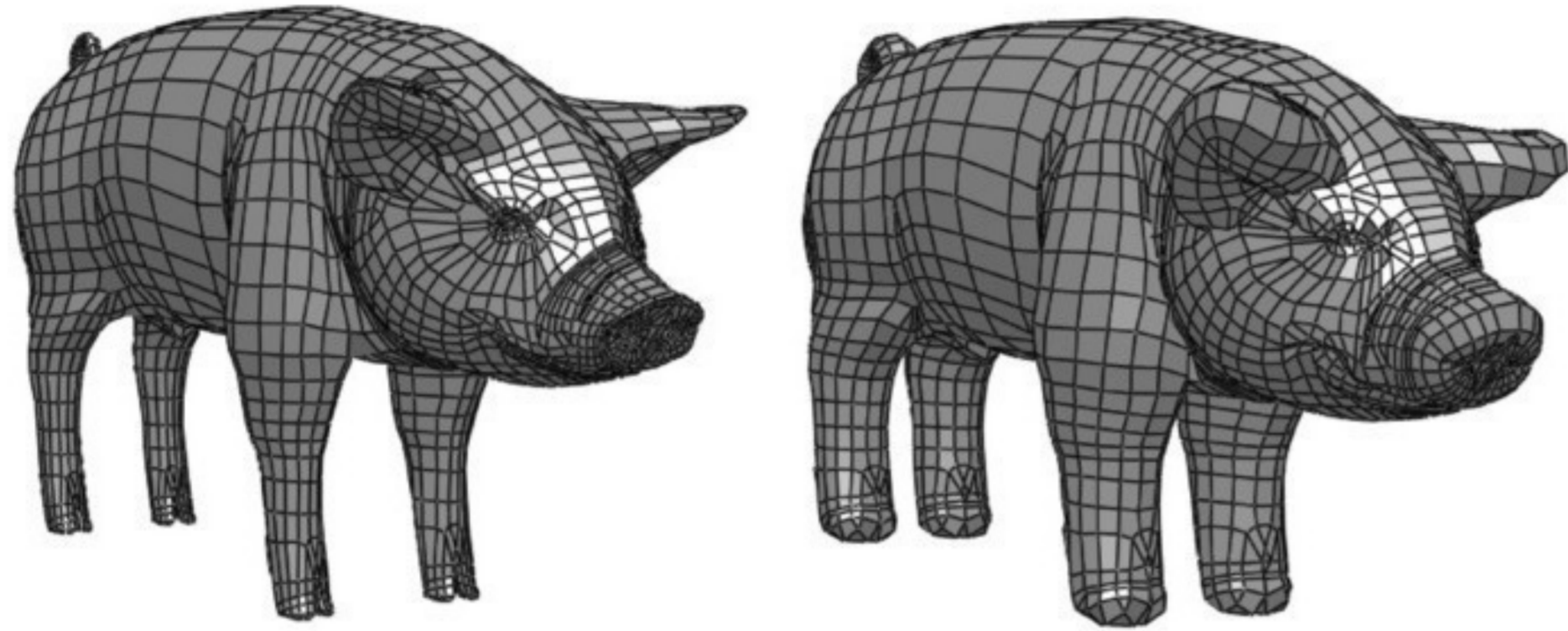
Mesh deformation

Surface animation / deformation

= find f such that

$$(\mathbf{y}_0, \mathbf{y}_1, \dots, \mathbf{y}_n) = f(\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_n)$$

We draw the mesh formed by $(\mathbf{y}_0, \mathbf{y}_1, \dots, \mathbf{y}_n)$ with the original connectivity



How to choose f ?

004

Deformation functions

Few functions are known

$$f : \mathbf{x} \in \mathbb{R}^3 \mapsto \mathbf{y} \in \mathbb{R}^3$$

We take f as a linear/affine map

$$f = M$$

Isometry

Translations

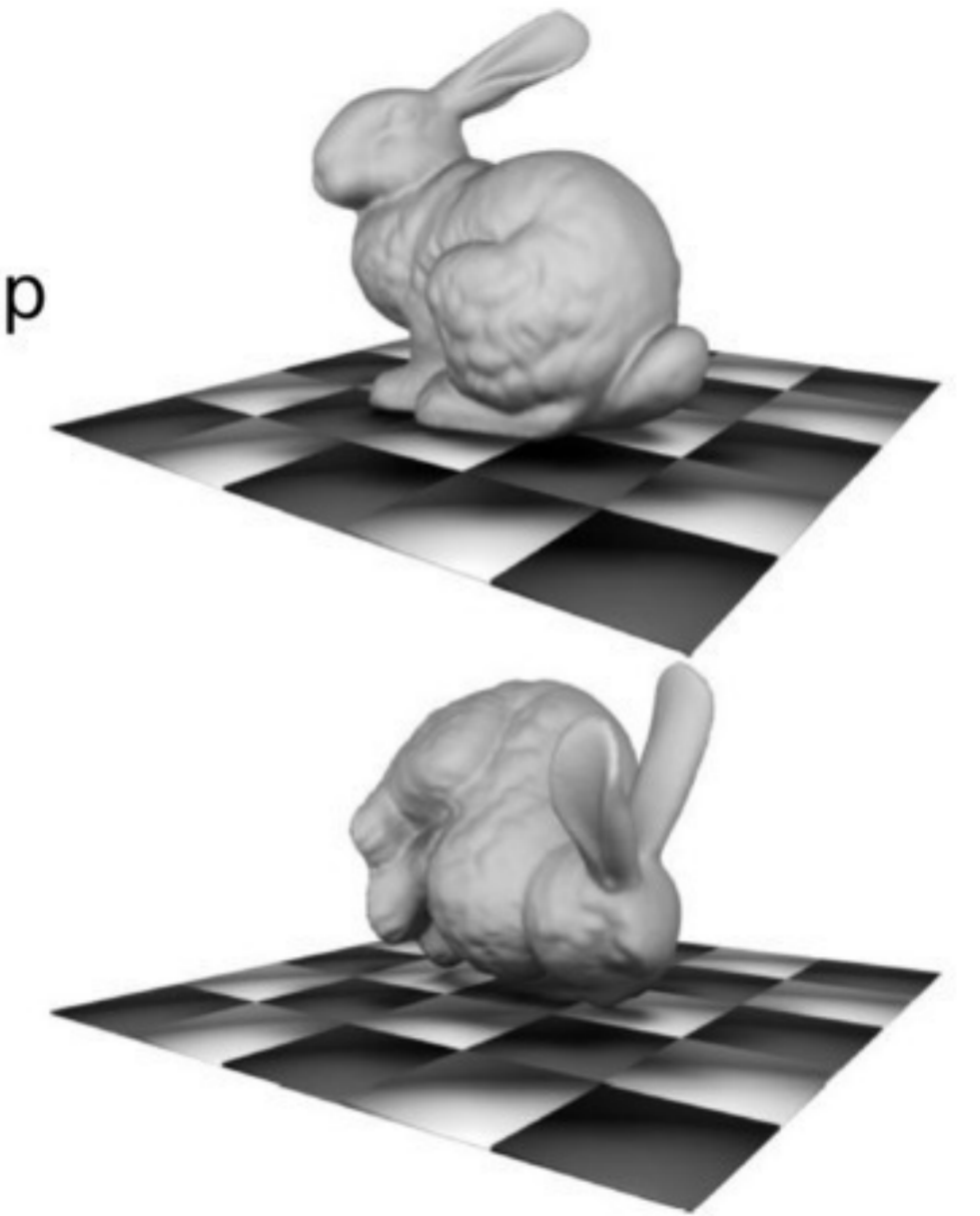
Rotations

Symetries

$$\mathbf{y} = M \mathbf{x} \quad \& \quad |\det(M)| = 1$$

Scaling

$$\mathbf{y} = \text{diag}(s_1, s_2, s_3) \mathbf{x}$$



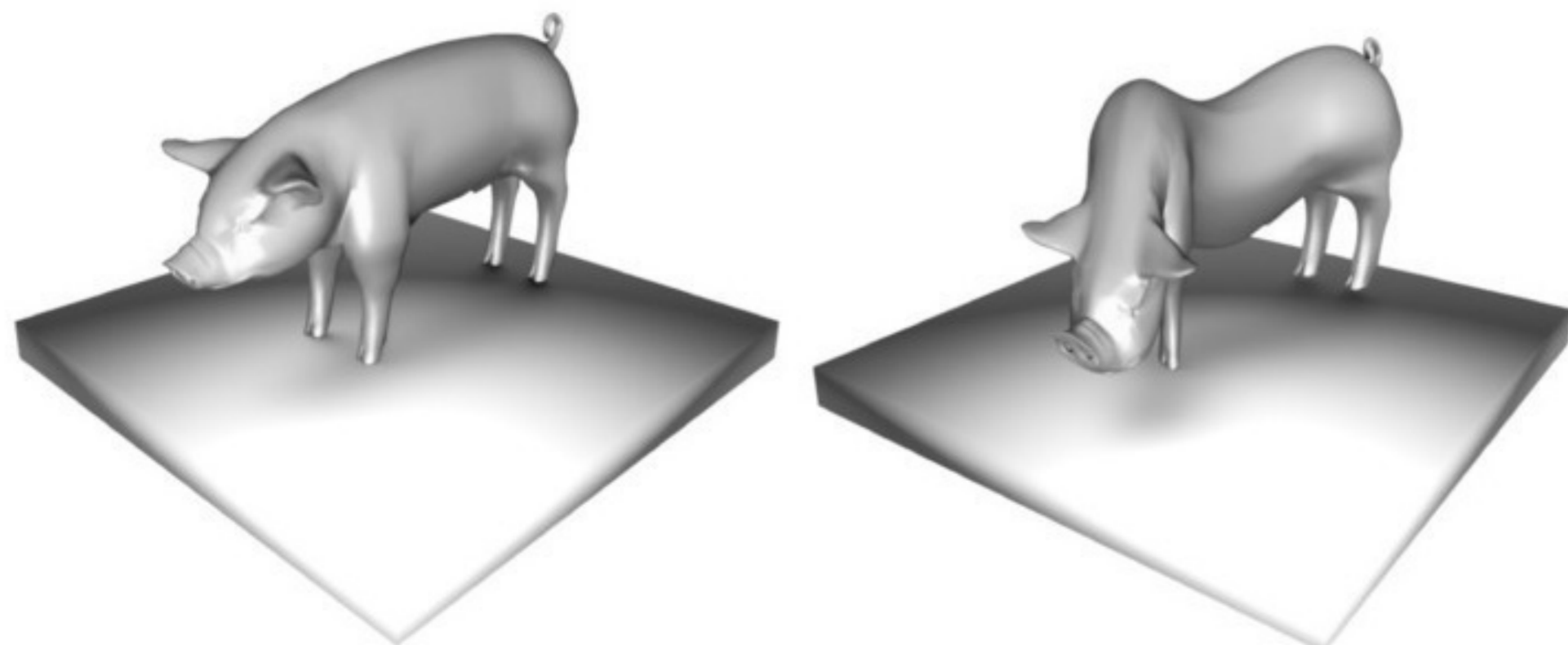
005

Rigid deformations

The parameter of a rigid transformation can vary in space

$$f : \mathbf{p} \in \mathbb{R}^3 \mapsto \mathbf{y} = M(\mathbf{p}) \mathbf{p}$$

ex. Translation depending of position



006

Rigid deformations

ex. Varying angle of rotation

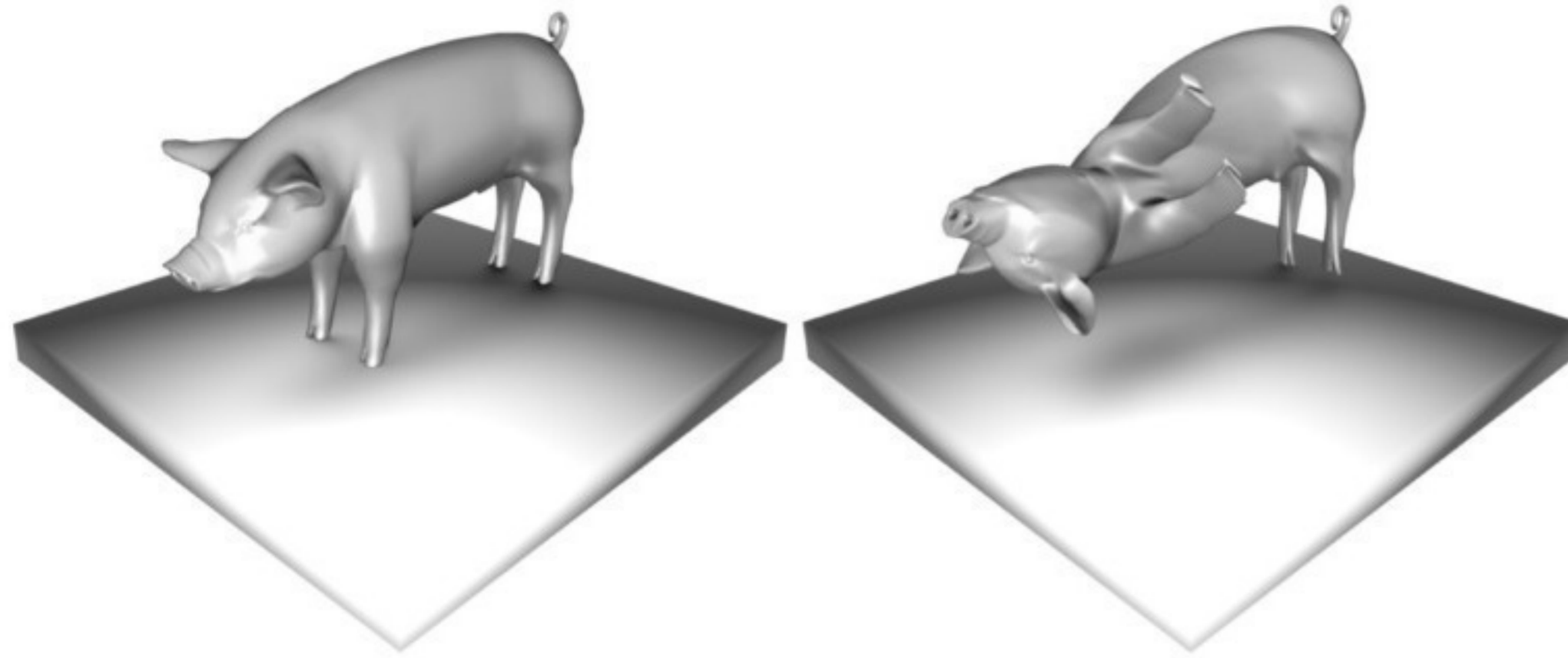
$$\forall i, \mathbf{y}_i = R \left(\mathbf{n}, \frac{z}{z_{\max}} \right) \mathbf{x}_i$$

```
for(k=0;k<N;k++)
{
  Vec x = mesh.get_vertex(k);
  double angle = x[0] * PI * time;
  Matrix R = Matrix::rotation(Vec(1,0,0),angle);
  mesh.set_vertex(k,R*x);
}
```

007

Rigid deformations

ex. Varying angle of rotation



008

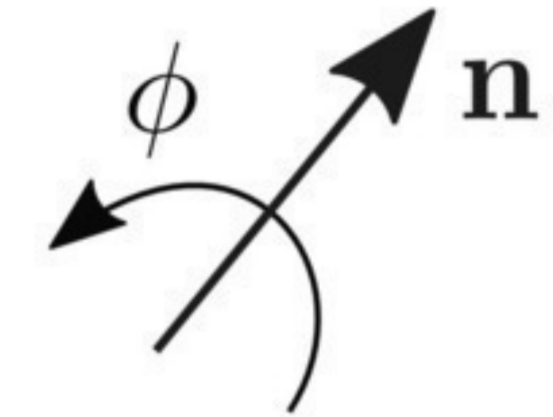
Rotation matrix

Every rotation can be parameterized by an **axe** and an **angle**

$$R(\mathbf{n}, \phi) = \begin{pmatrix} \cos(\phi) + n_x^2(1 - \cos(\phi)) & n_x n_y(1 - \cos(\phi)) - n_z \sin(\phi) & n_y \sin(\phi) + n_x n_z(1 - \cos(\phi)) \\ n_z \sin(\phi) + n_x n_y(1 - \cos(\phi)) & \cos(\phi) + n_y^2(1 - \cos(\phi)) & -n_x \sin(\phi) + n_y n_z(1 - \cos(\phi)) \\ -n_y \sin(\phi) + n_x n_z(1 - \cos(\phi)) & n_x \sin(\phi) + n_y n_z(1 - \cos(\phi)) & \cos(\phi) + n_z^2(1 - \cos(\phi)) \end{pmatrix}$$

To apply a rotation:

$$\forall i, \mathbf{y}_i = R \mathbf{x}_i$$



009

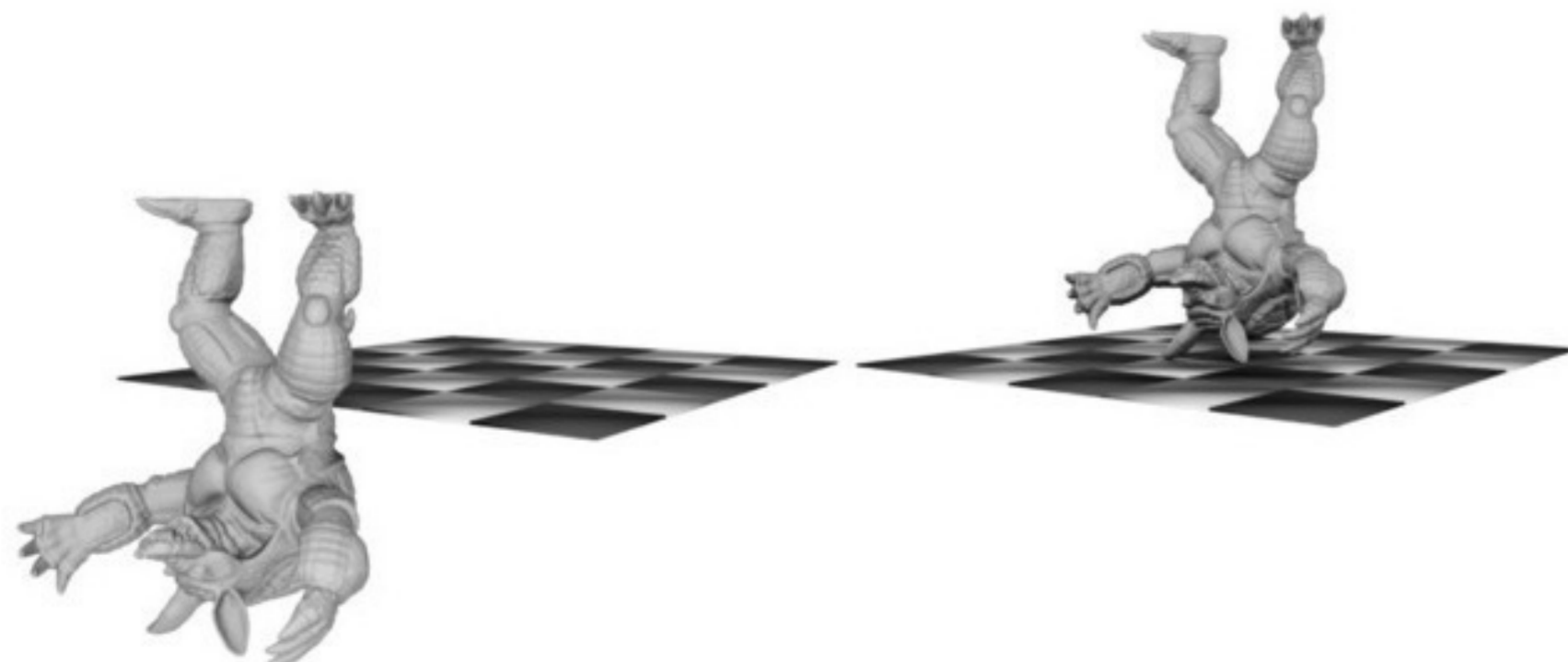
Rotation matrix

Should not forget the rotation center

$$\mathbf{q} = R(\mathbf{p} - \mathbf{p}_0) + \mathbf{p}_0$$

Or in its matrix form

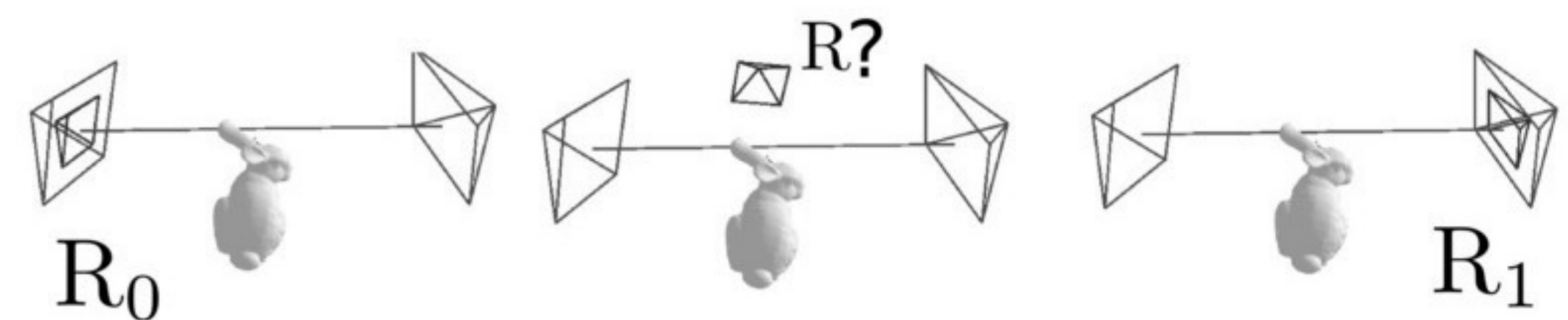
$$\mathbf{q} = T^{-1} R T \mathbf{p}$$



010

Rotation interpolation

How to interpolate rotation ?



Linear interpolation does not work

$$M = \alpha R_1 + (1 - \alpha) R_2 \notin SO(3)$$

Problem: Rotation space is not a vectorial space

Rotation space is a manifold called Lie Group
(unit sphere in 4D)

011

Rotation interpolation

Generalization of linear interpolation in Lie Group

$$R = (R_2 R_1^{-1})^\alpha R_1$$

But: requires matrix exponential

- Computational cost
- Numerical instabilities

012

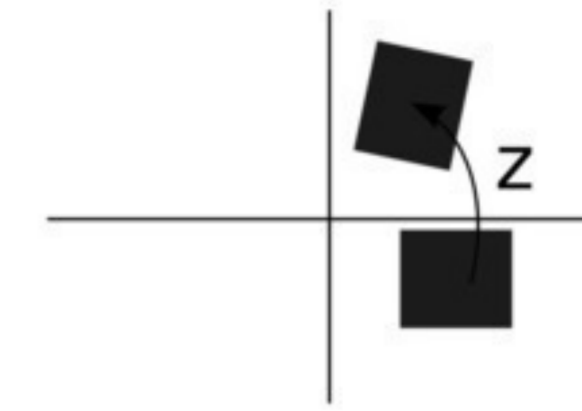
Quaternion

Remember that complex numbers can model rotation in 2D

$$z = e^{i\theta}$$

$$z = x + y \mathbf{i}$$

$$|z| = 1$$



Quaternions are generalization of complex numbers in 3D

$$q = x + y \mathbf{i} + z \mathbf{j} + w \mathbf{k} \in \mathbb{R}^4$$

$$|q| = 1$$

013

Quaternion

Unit quaternion are rotations in 3D

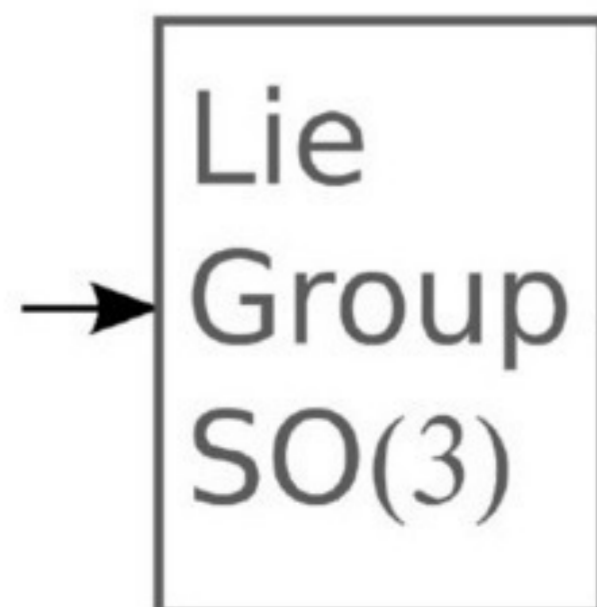
$$q = x + y \mathbf{i} + z \mathbf{j} + w \mathbf{k}$$

$$|q| = 1$$

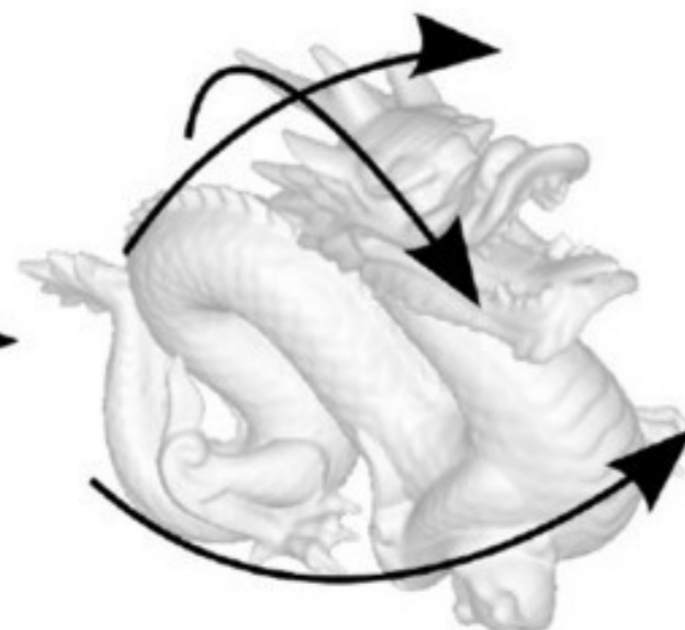
+ Short representation of rotation $q_1 q_2 \Leftrightarrow R_1 R_2$
(non comutative product)

Algebre

$$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$



rotations



014

Equivalence rotation/quaternion

Quaternion $q = (q_0, q_1, q_2, q_3) \Rightarrow$ Matrix R

$$R = \begin{pmatrix} 1 - 2(q_1^2 + q_2^2) & 2(q_0 q_1 - q_3 q_2) & 2(q_0 q_2 + q_3 q_1) \\ 2(q_0 q_1 + q_3 q_2) & 1 - 2(q_0^2 + q_2^2) & 2(q_1 q_2 - q_3 q_0) \\ 2(q_0 q_2 - q_3 q_1) & 2(q_1 q_2 + q_3 q_0) & 1 - 2(q_0^2 + q_1^2) \end{pmatrix}$$

Efficient representation of multiplication: $q = (\mathbf{v}, w)$

$$q_1 q_2 = (w_0 \mathbf{v}_1 + w_1 \mathbf{v}_0 - \mathbf{v}_0 \times \mathbf{v}_1, w_0 w_1 - \mathbf{v}_0 \cdot \mathbf{v}_1)$$

015

SLERP

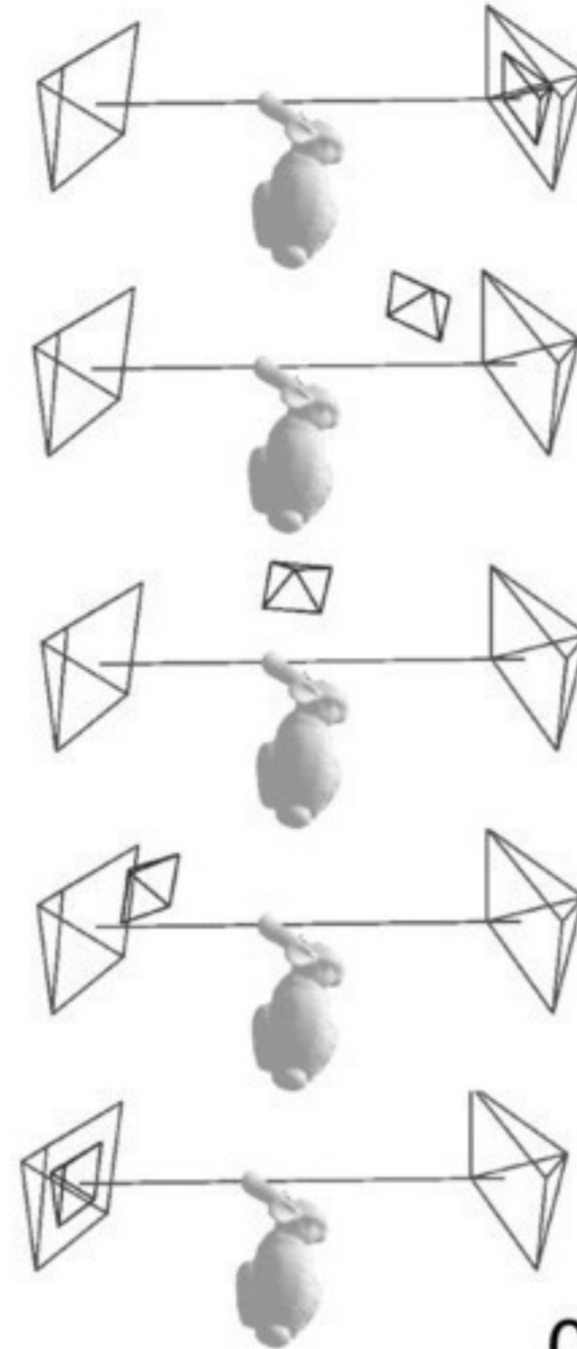
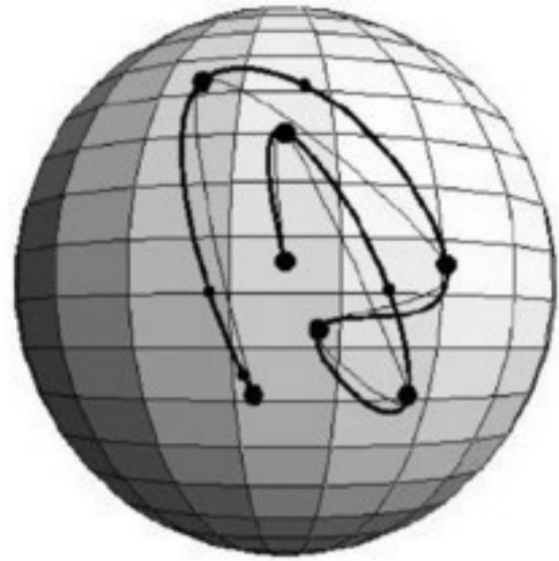
Quaternion interpolation => Rotation interpolation

SLERP: Spherical Linear Interpolation

$$\text{SLERP}(q_0, q_1, \alpha) = \frac{\sin(\alpha(1 - \theta))}{\sin(\theta)} q_0 + \frac{\sin(\alpha\theta)}{\sin(\theta)} q_1$$

$$\cos(\theta) = q_0 \cdot q_1$$

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



016

Rigid transformations

- Non local transformation
- Complex to manipulate if axe/angle vary in space

=> We use mainly constant rigid transformation

=> Make them artificially piecewise local

$$\forall i, \mathbf{y}_i = \mathbf{M}(\varepsilon(i)) \mathbf{x}$$

$\mathbf{M}(\varepsilon(i)) = \mathbf{I}$: for all undeformed set

$\varepsilon(i)$: mesh segmentation

017

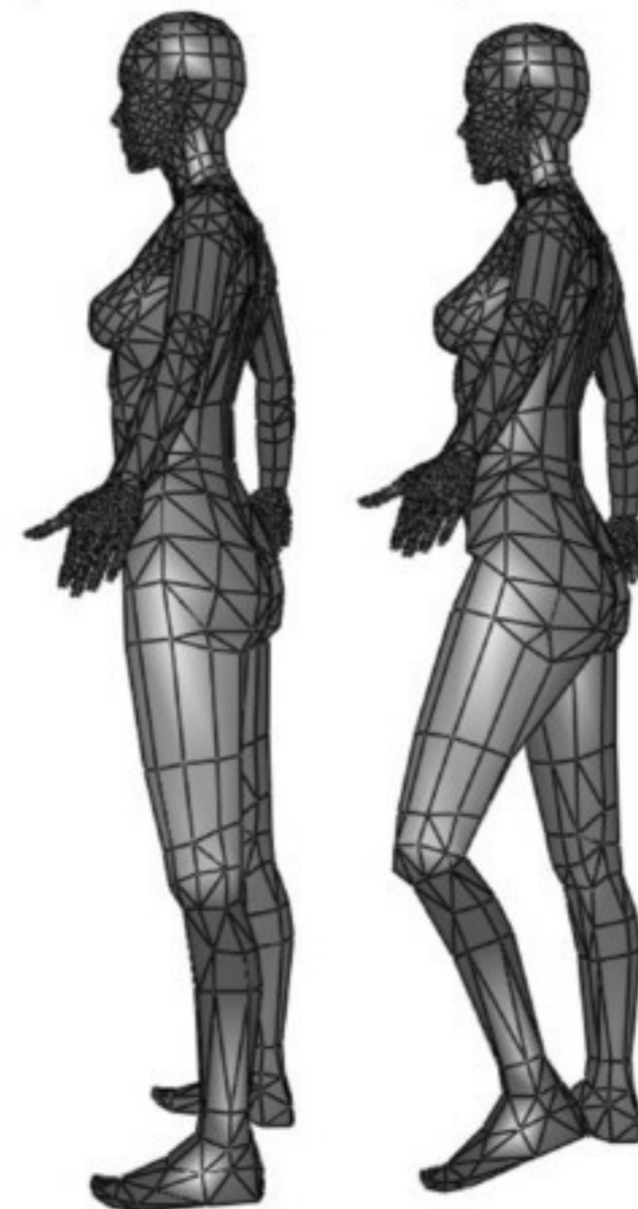
Local transformation

Define the regions ε

Apply a different transformation $\mathbf{M}(\varepsilon)$ on each region

Mainly use rotation \mathbf{R}_ε

How to define the parameters ?

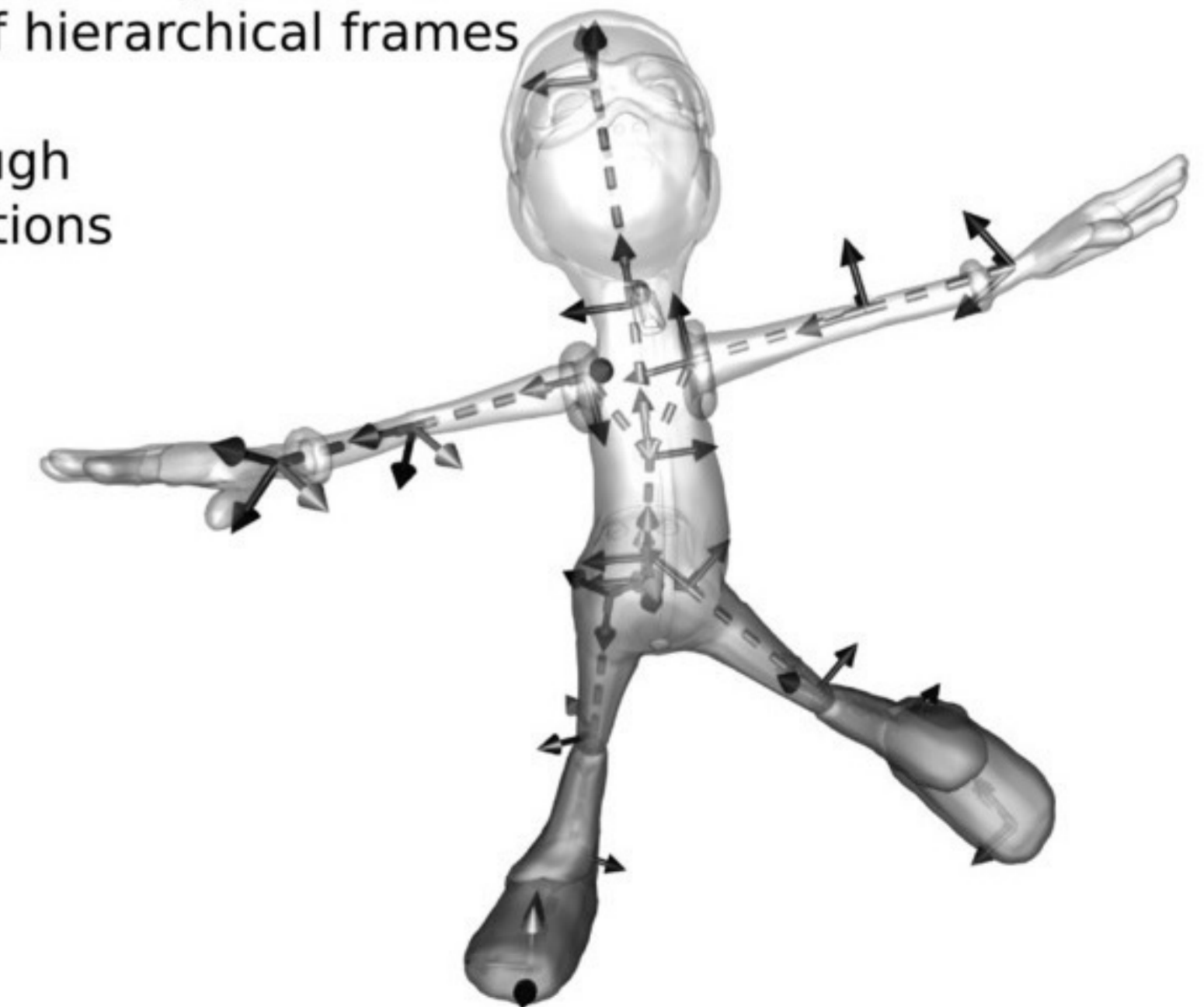


018

Animation skeleton

Object articulated using a skeleton
Skeleton=Set of hierarchical frames

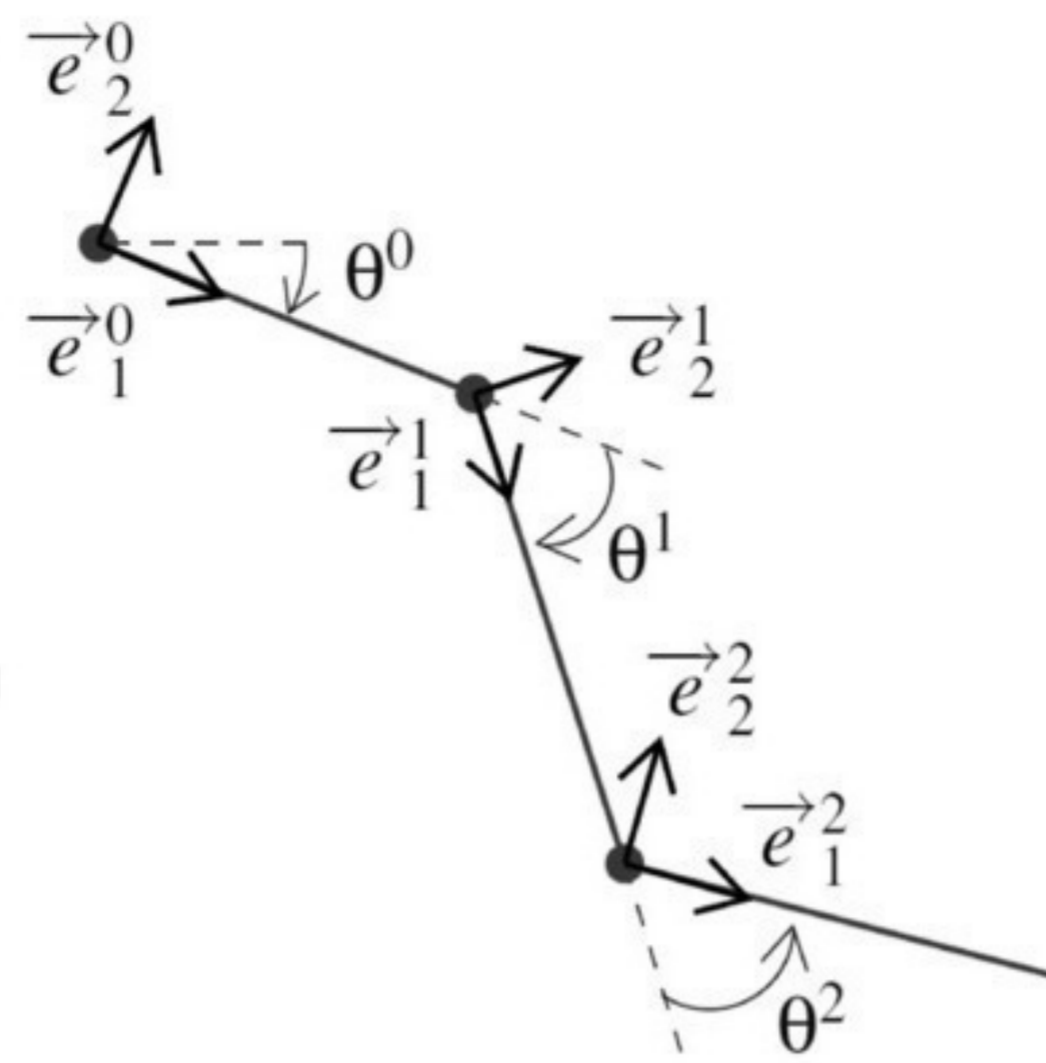
Animation through
successive rotations



019

Animation skeleton

Hierarchical transformations



$$T_0 = R_0$$

$$T_1 = T_0 M_1 R_1 M_1^{-1} = R_0 M_1 R_1 M_1^{-1}$$

$$T_2 = T_1 M_2 R_2 T_1^{-1} = \dots$$

$$\vdots$$

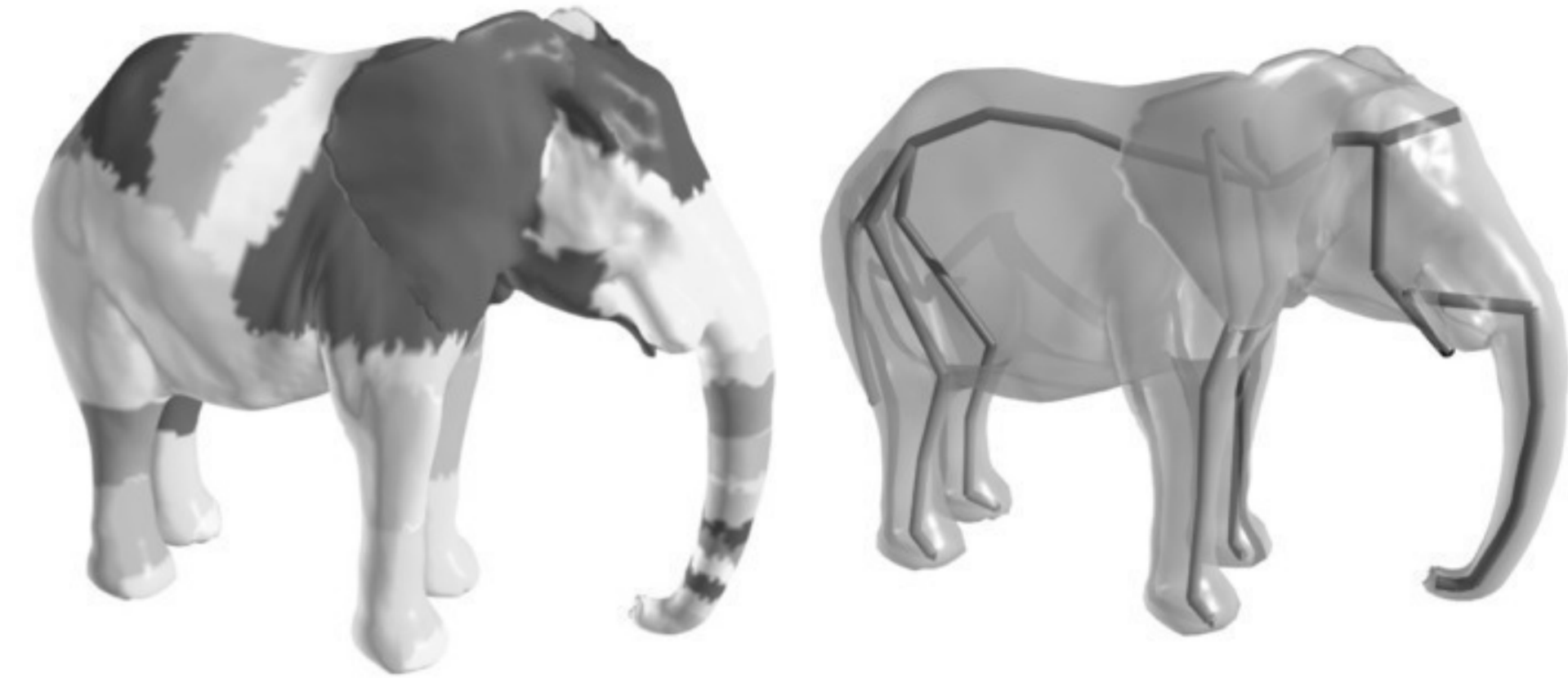
$$T_i = \prod_{k=0}^i M_k R_k M_k^{-1}$$

020

Rigid skinning

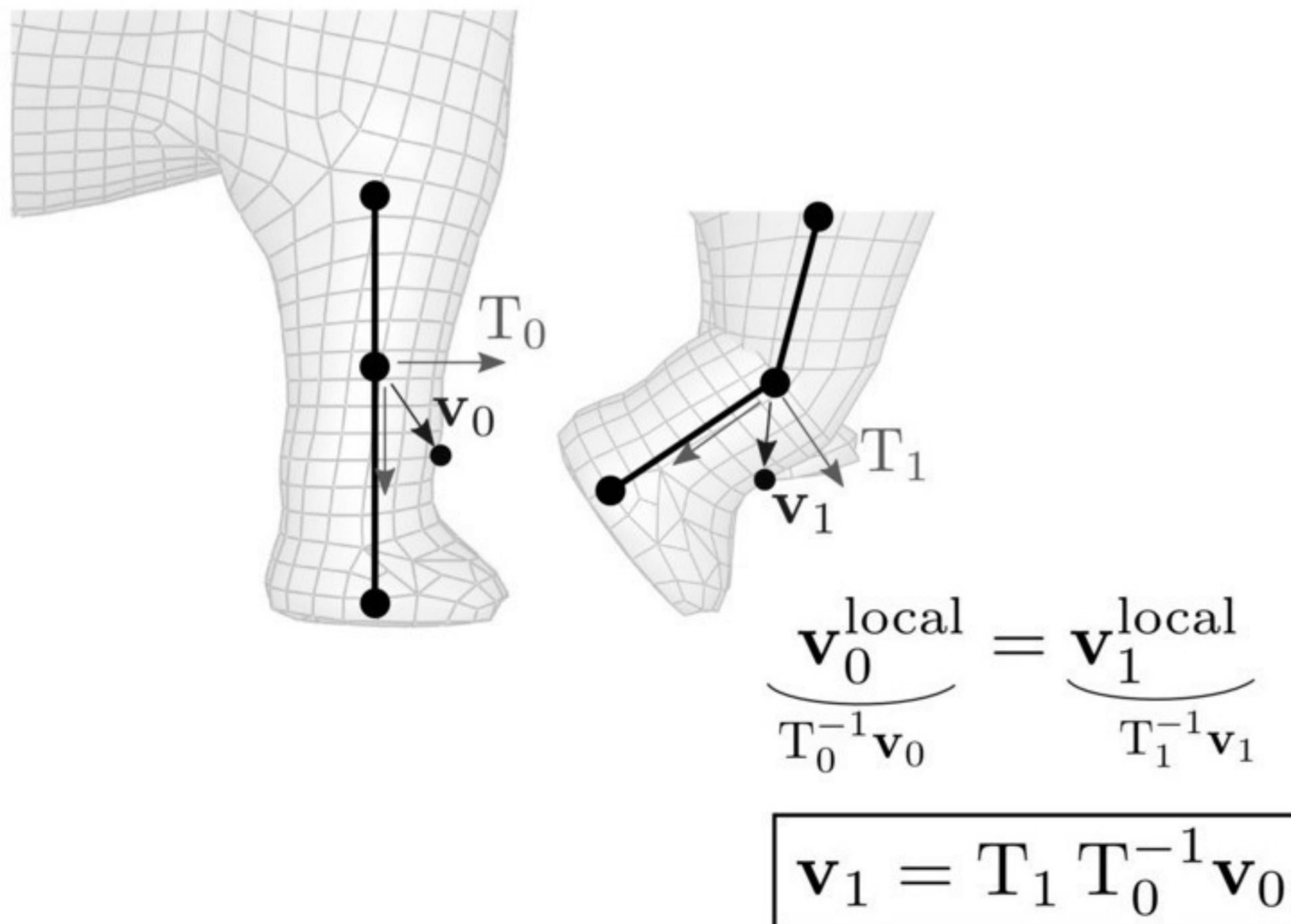
Link a set of vertices to the frame of a skeleton

Apply the frame deformation to all the vertices of the set



021

Rigid skinning



022

Skeleton

```
class Frame
{
    Matrix R;
    Matrix Bind;
    Joint *father;
    std::list <Joint*> son;
}
```

```
for(k_vertex=0;k_vertex<N_vertex;k_vertex++)
{
    int bone_dependency =
    skinning.bone_dependency(k_vertex);
    Matrix T = skeleton.get_matrix(bone_dependency);
    Matrix B =
    (skeleton.get_bind_pose(bone_dependency)).invert();
    deformed_mesh(k_vertex) = T*B * mesh(k_vertex);
}
```

023

Good point of Skinning

The skeleton is

- + Easy to construct
- + Easy to animate
- + Intuitif to manipulate an articulated character



024

Disadvantage of skinning

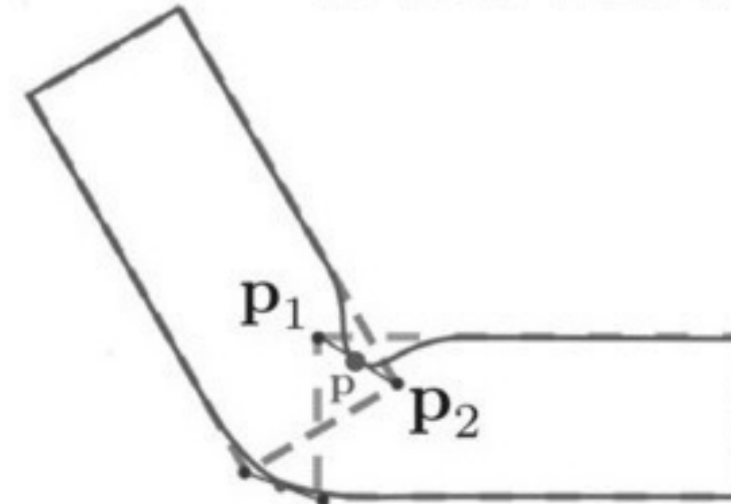
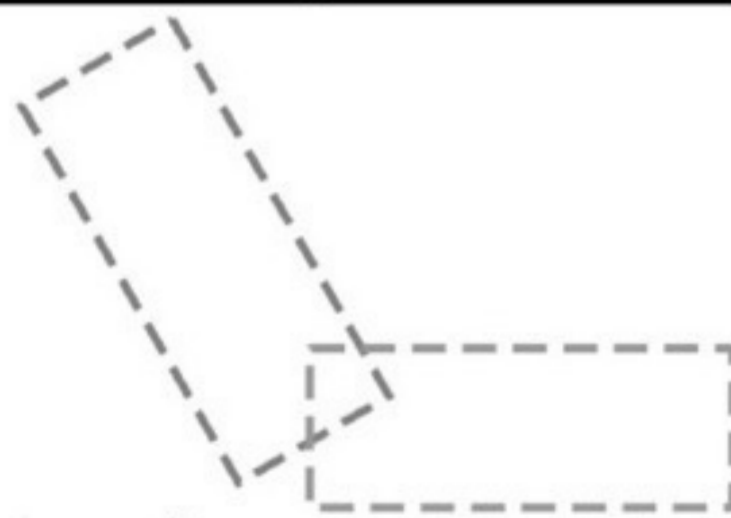
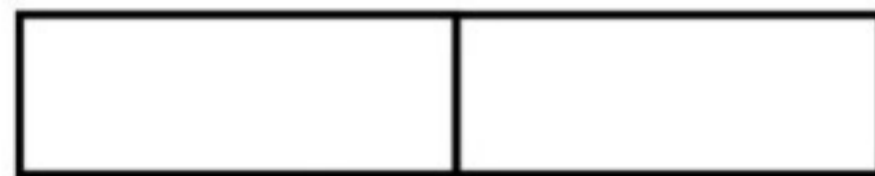
- Discontinuities



025

Interpolation of transformation

Idea: *Should interpolate deformation between frames*



$$\mathbf{p} = 0.5 \mathbf{p}_1 + 0.5 \mathbf{p}_2$$

$$\mathbf{p} = 0.5 T_1 \mathbf{p}_0 + 0.5 T_2 \mathbf{p}_0$$

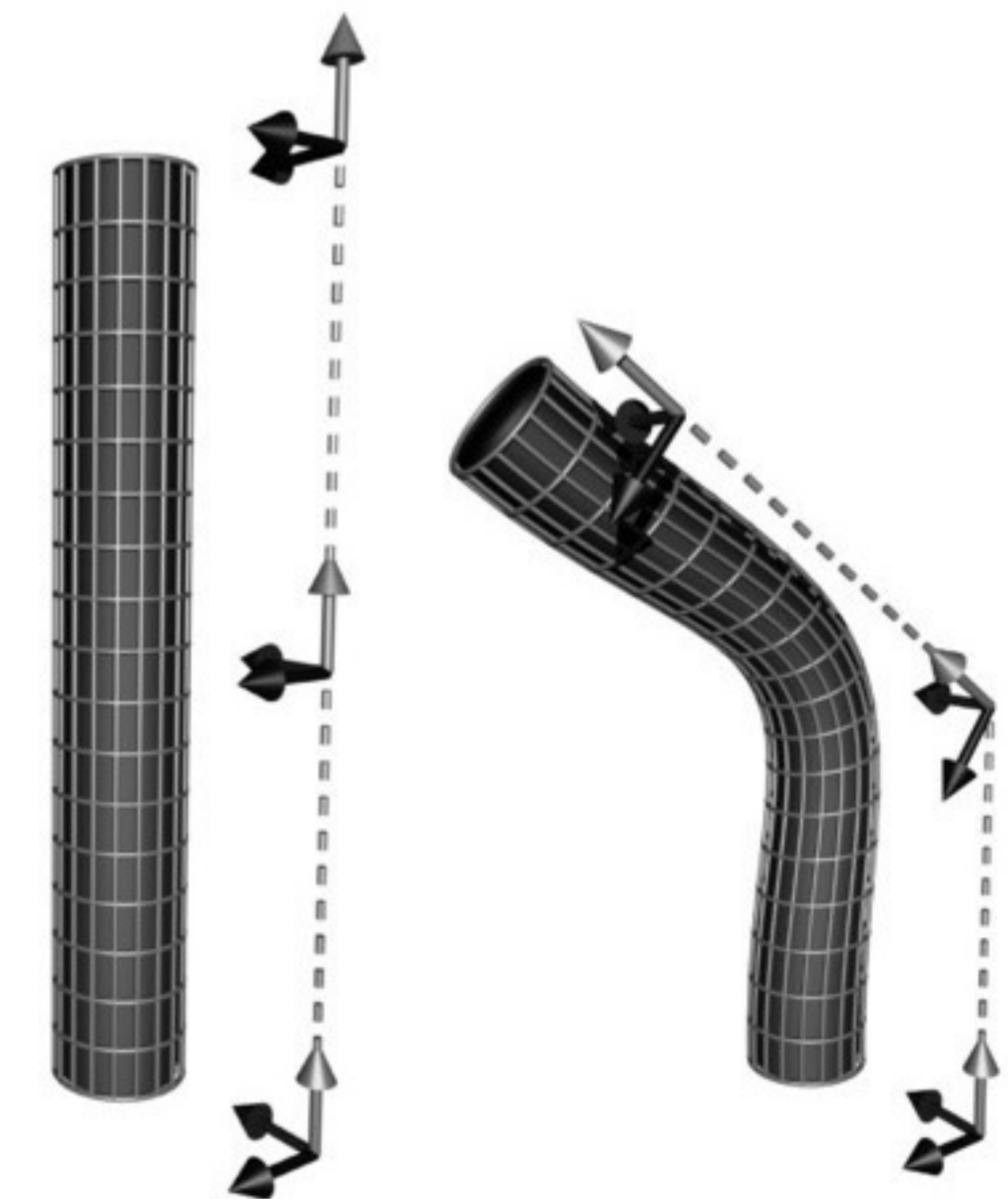
$$\mathbf{p} = \underbrace{(0.5 T_1 + 0.5 T_2)}_{\alpha T_1 + (1 - \alpha) T_2} \mathbf{p}_0$$

026

Interpolation of transformation

$$T = \alpha T_1 + (1 - \alpha) T_2$$

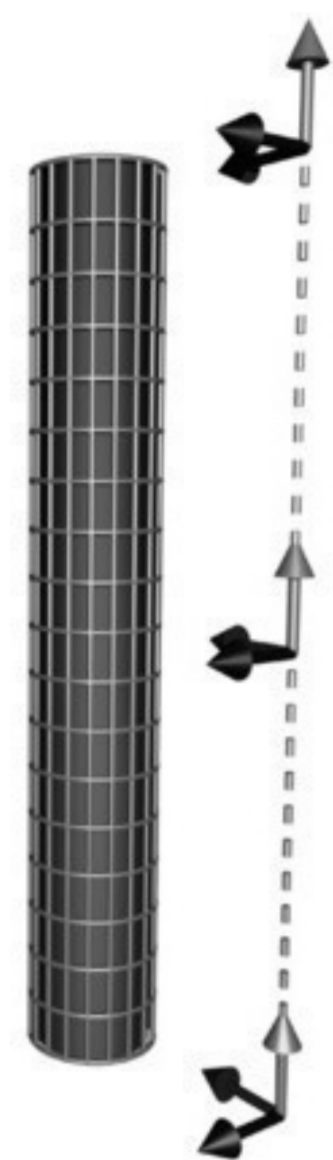
How to compute the α



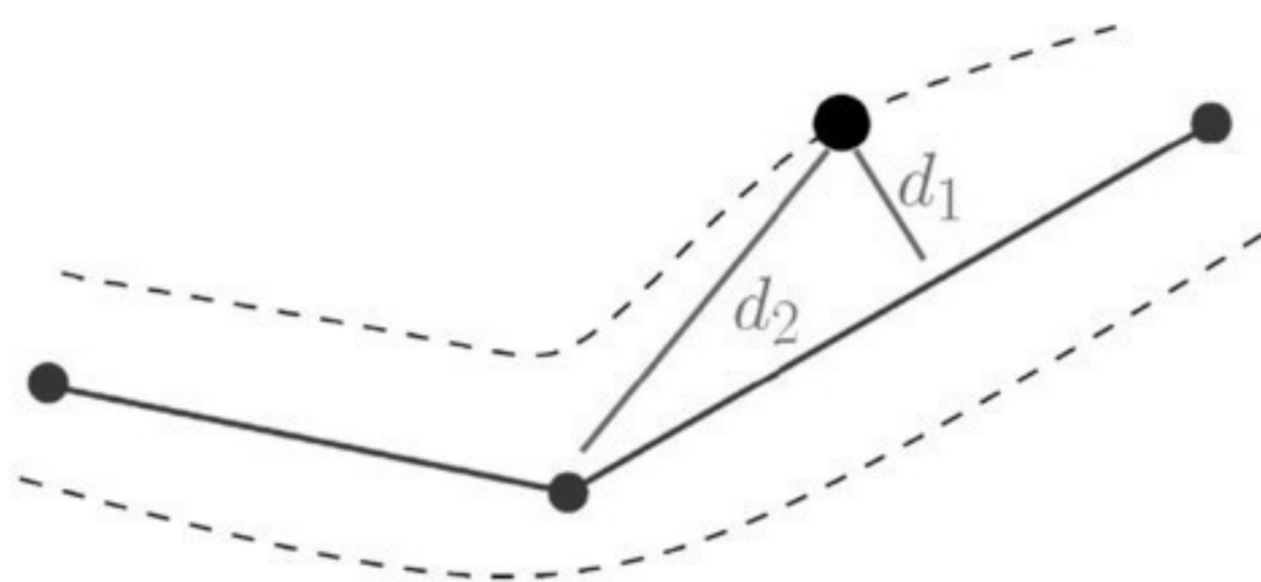
027

Skinning weights

Use of a distance function



$$\alpha = \frac{d_1^{-1}}{d_1^{-1} + d_2^{-1}}$$



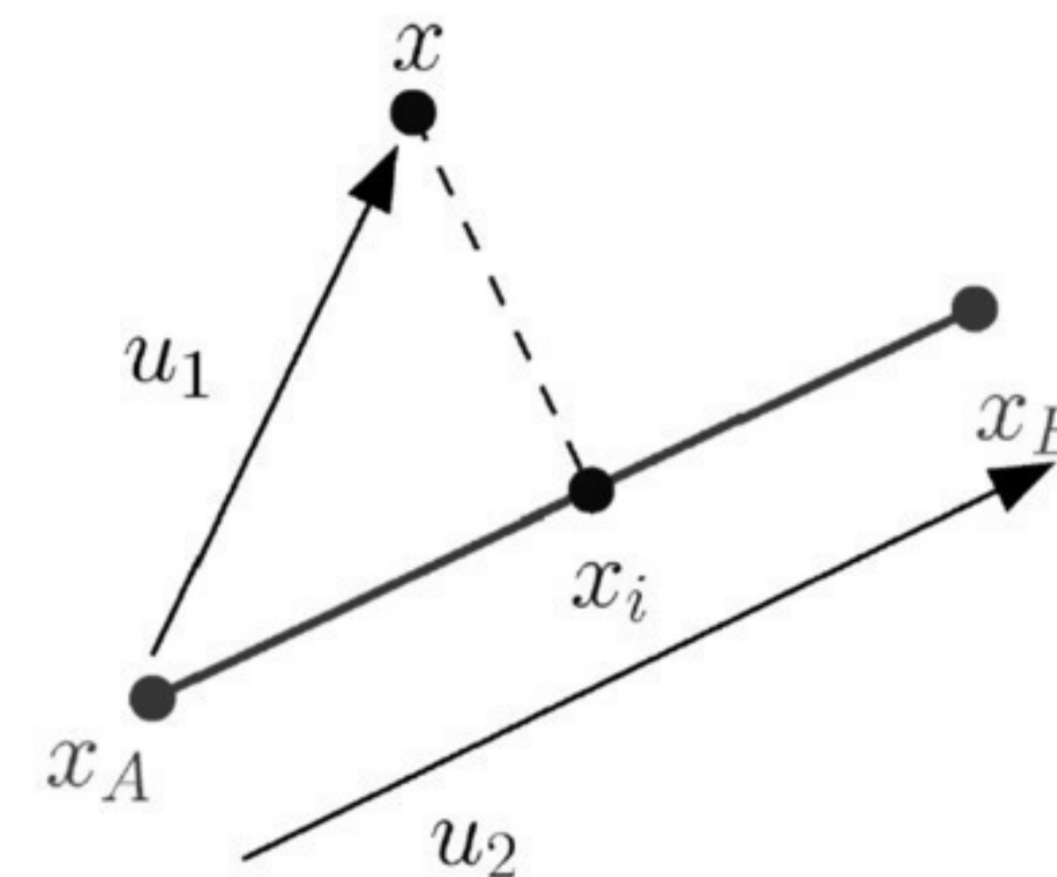
028

Cylindrical distance

Shortest distance of a position \mathbf{x} to a segment [AB]

$$\mathbf{u}_1 = \mathbf{x} - \mathbf{x}_A \quad \rho = \frac{\mathbf{u}_1 \cdot \mathbf{u}_2}{\|\mathbf{u}_2\|^2}$$
$$\mathbf{u}_2 = \mathbf{x}_B - \mathbf{x}_A$$

If $\rho < 0$, $\mathbf{x}_i = \mathbf{x}_A$
If $\rho > 1$, $\mathbf{x}_i = \mathbf{x}_B$
Else $\mathbf{x}_i = \mathbf{x}_A + \rho \mathbf{u}_2$



$$d = \|\mathbf{x} - \mathbf{x}_i\|$$

029

Data structure: skinning weights

For every vertex i , we associate the pair (bone, weight)

```
std::vector <std::vector <int>> bone_dependencies;  
std::vector <std::vector <double>> skinning_weights;  
for(k_vertex=0;k_vertex<N_vertex;k_vertex++)  
{  
    N_dep = bone_dependencies[k_vertex].size();  
    for(k_dep=0;k_dep<N_dep;k_dep++)  
    {  
        bone = bone_dependencies[k_vertex][k_dep];  
        weight = skinning_weight[k_vertex][k_dep];  
    }  
}
```

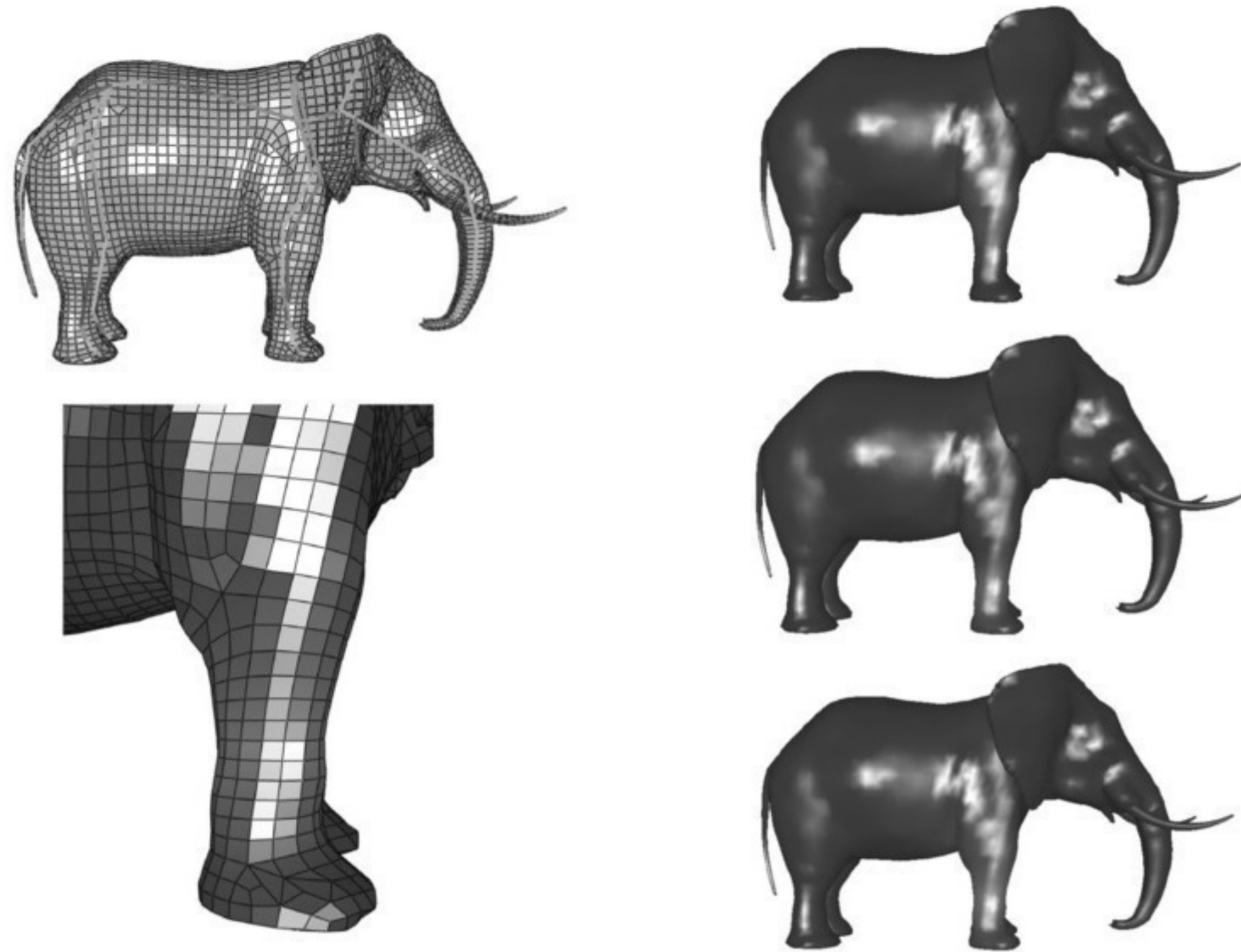
030

Skinning weights computation

```
//skinning weights  
for(k_vertex=0;k_vertex<N_vertex;k_vertex++)  
{  
    for(k_bone=0;k_bone<N_bone;k_bone++)  
    {  
        alpha = push_back(1/distance(k_vertex,k_bone));  
        if(alpha>epsilon)  
        {  
            skinning_weights[k_vertex].push_back(alpha);  
            bone_dependencies[k_vertex].push_back(k_bone);  
        }  
    }  
    norm_skinning_weights();  
}
```

031

Skinning weights example



032

Smooth skinning

$$\mathbf{p} = \left(\sum_i \omega_i \mathbf{M}_i \right) \mathbf{p}_0$$

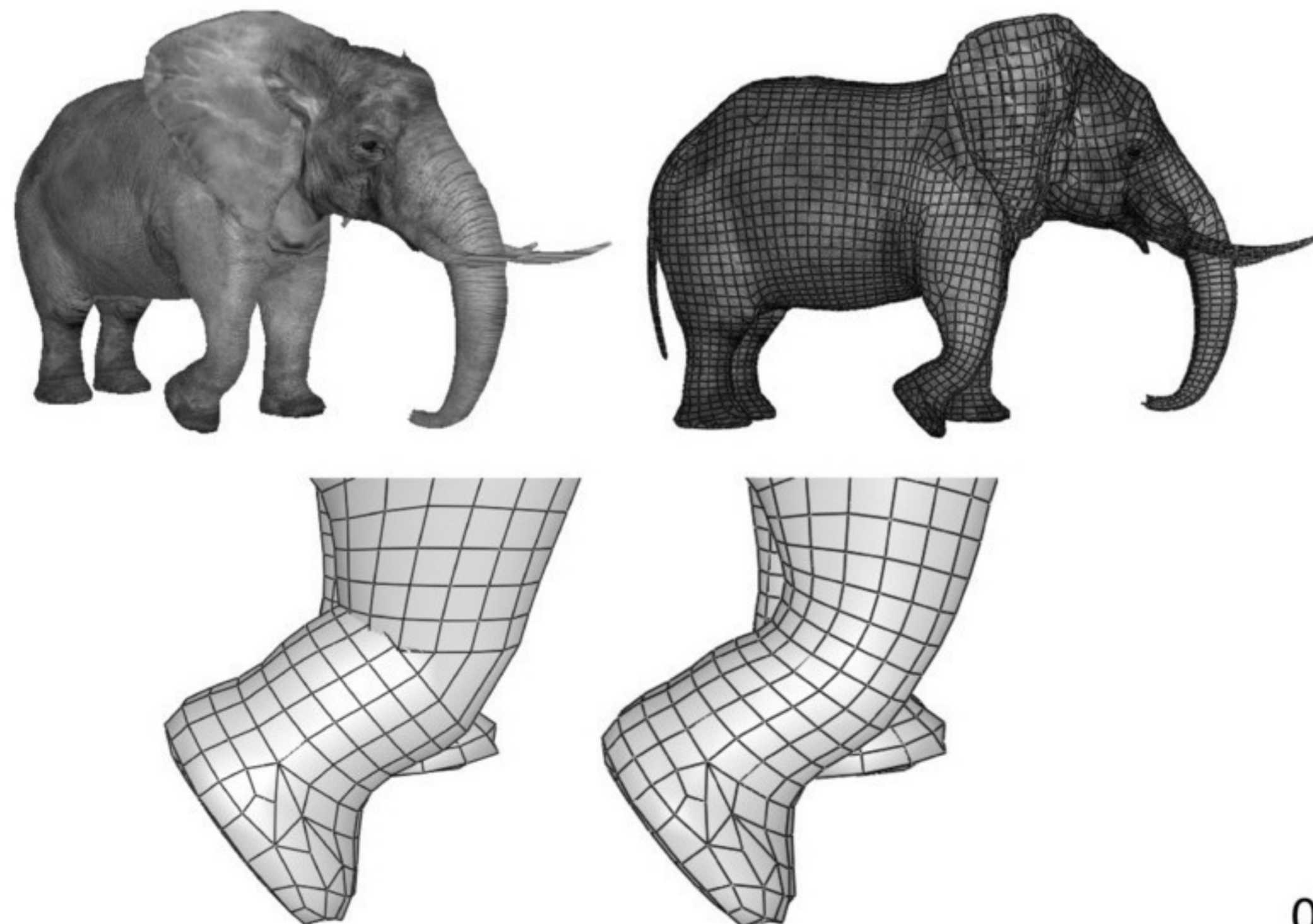
$$\sum_i \omega_i = 1$$

```
for(k_vertex=0;k_vertex<N_vertex;k_vertex++)
{ int N_dep = bone_dependencies[k_vertex].size();
  Matrix D;
  for(k_dep=0;k_dep<N_dep;k_dep++)
  { int k_bone = bone_dependency[k_vertex][k_dep];
    double weight = skinning_weights[k_vertex][k_dep];
    D += weight*skeleton.matrix(k_bone);
    deformed_mesh(k_vertex) = D * mesh(k_vertex);
  }
}
```



033

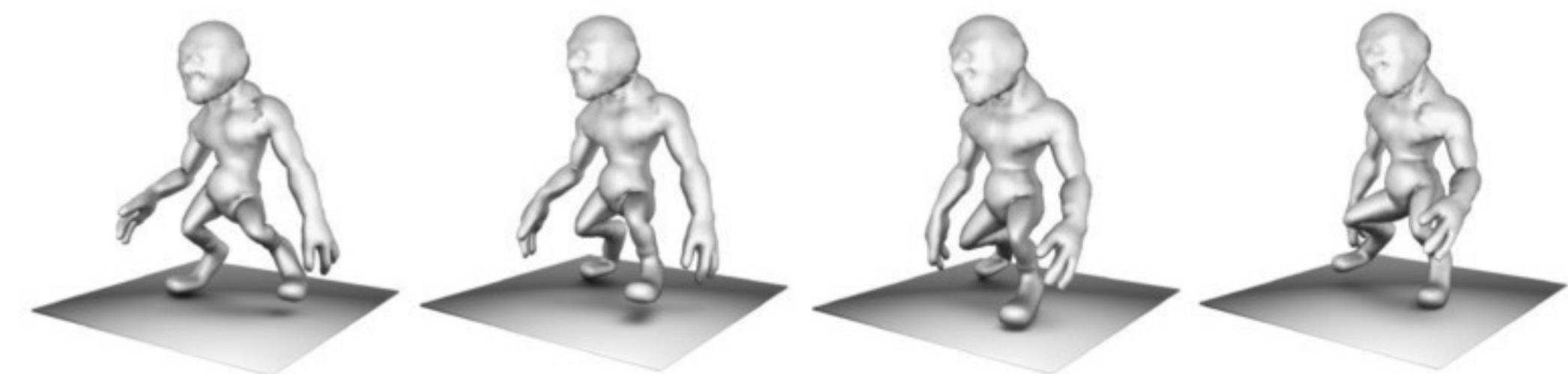
Smooth skinning examples



034

Animation

Angles varies through time

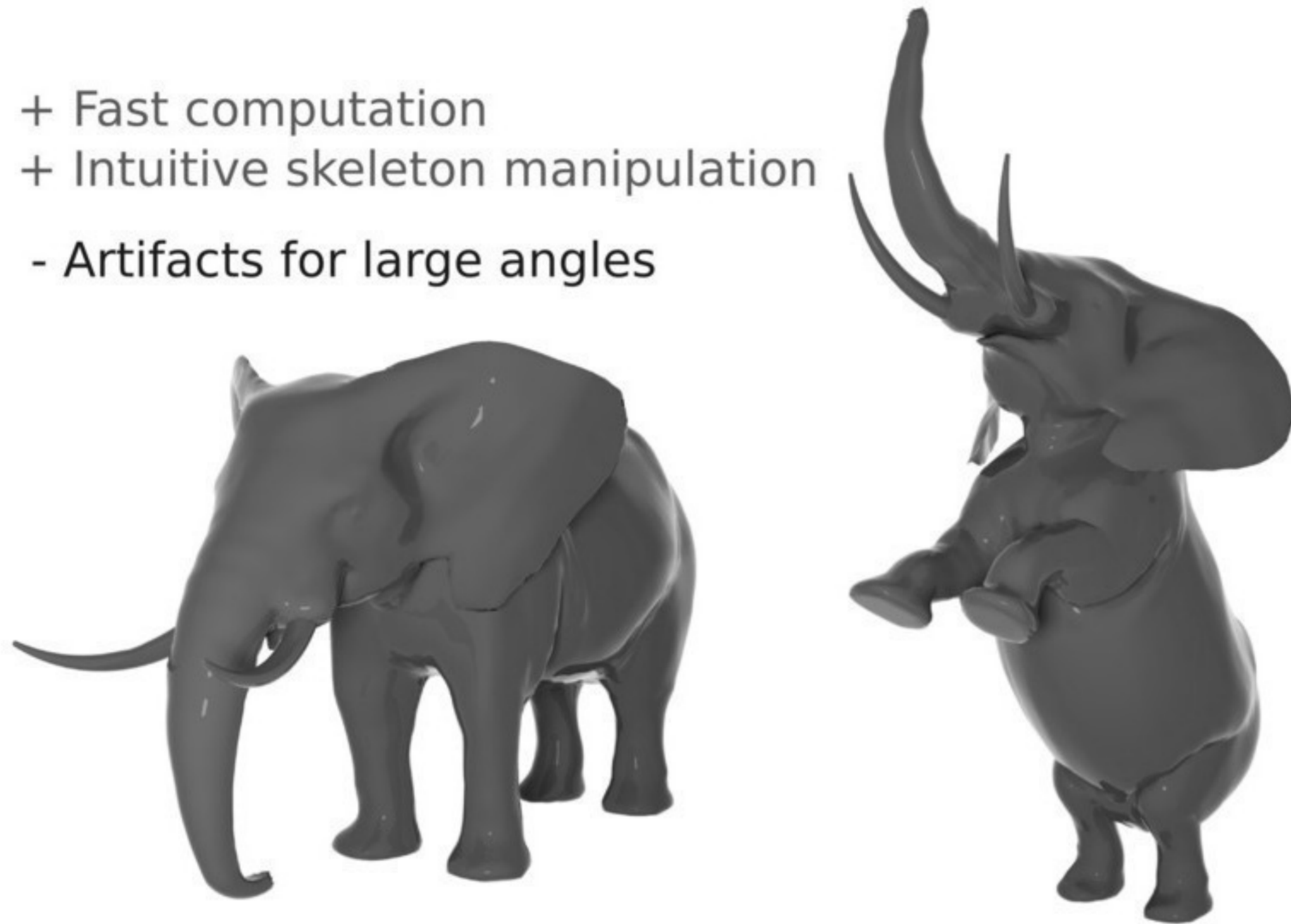


```
- <animation id="cle_R_elbow_rotateY">
- <source id="cle_R_elbow_rotateY_cle_R_elbow_rotateY_ANGLE-input">
- <float_array id="cle_R_elbow_rotateY_cle_R_elbow_rotateY_ANGLE-input-array" count="134">
0.040000 0.080000 0.120000 0.160000 0.200000 0.240000 0.280000 0.320000 0.360000 0.400000 0.440000 0.480000 0.520000
0.560000 0.600000 0.640000 0.680000 0.720000 0.760000 0.800000 0.840000 0.880000 0.920000 0.960000 1.000000 1.040000
1.080000 1.120000
</float_array>
</source>
- <source id="cle_R_elbow_rotateY_cle_R_elbow_rotateY_ANGLE-output">
- <float_array id="cle_R_elbow_rotateY_cle_R_elbow_rotateY_ANGLE-output-array" count="134">
0 -0.072670 -0.223444 -0.389830 -0.549435 -0.692583 -0.812962 -0.905247 -0.964565 -0.985765 -0.985721 -0.985668 -0.985607
-0.985540 -0.961561 -0.887361 -0.755854 -0.560878 -0.285142 -0.201935 -0.173549 -0.175789 -0.182313 -0.192826 -0.207032
-0.224636
</float_array>
```

035

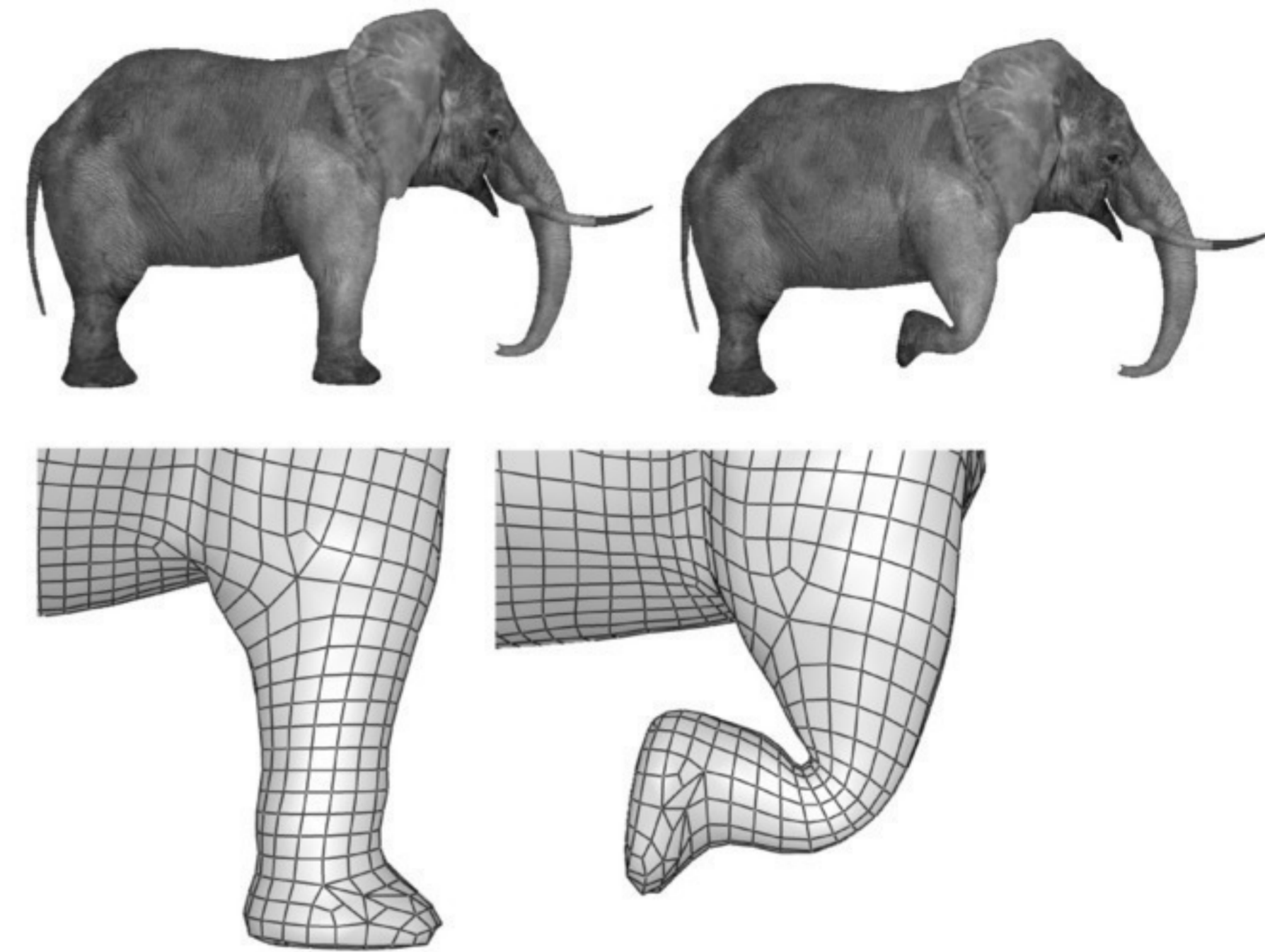
Smooth skinning, conclusion

- + Fast computation
- + Intuitive skeleton manipulation
- Artifacts for large angles



036

Skinning artifact: Collapsing Elbow

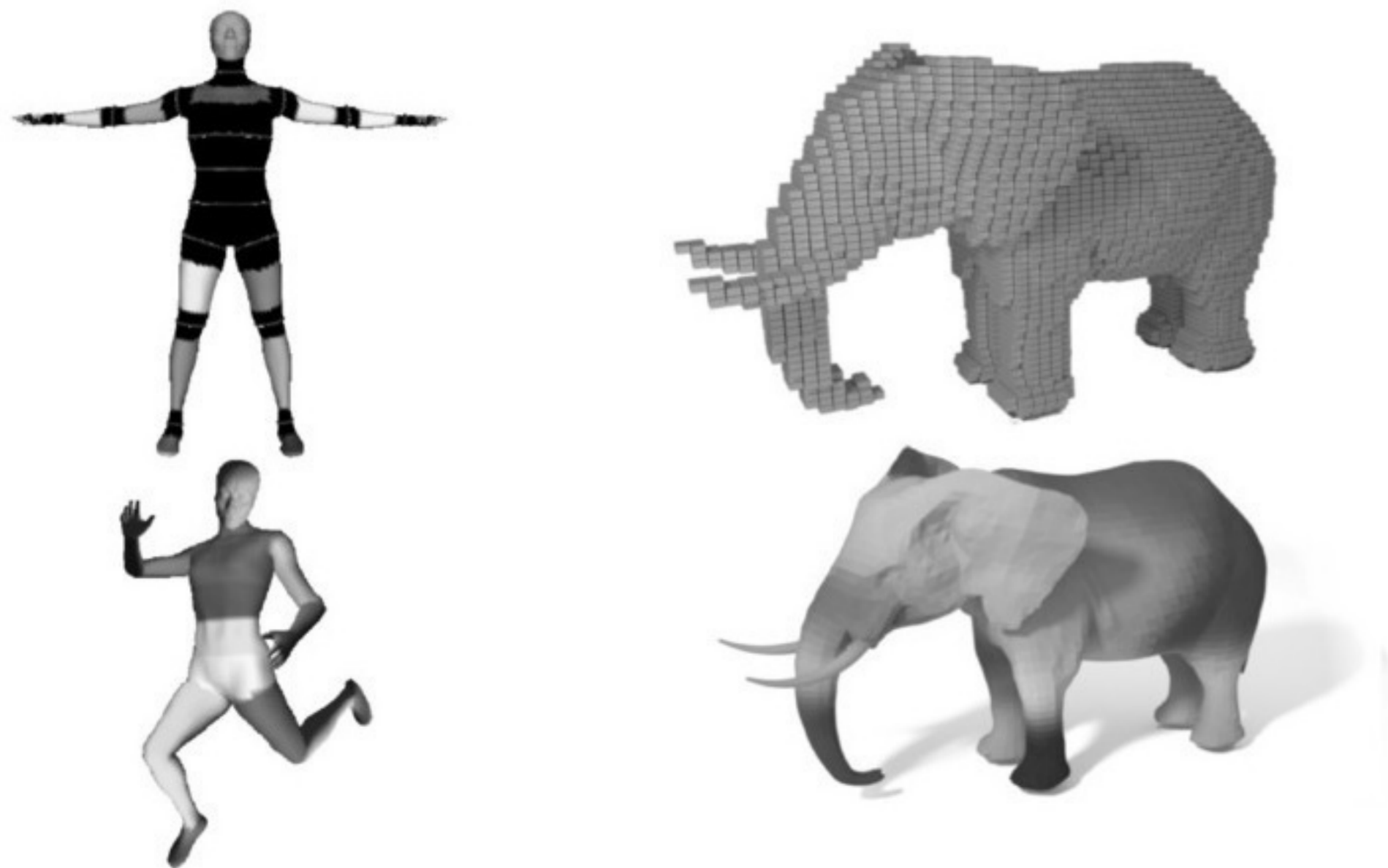


037

Skinning weight computation

Problem with cartesian distance

- Artistic tools to paint weights
- Use of geodesic distance

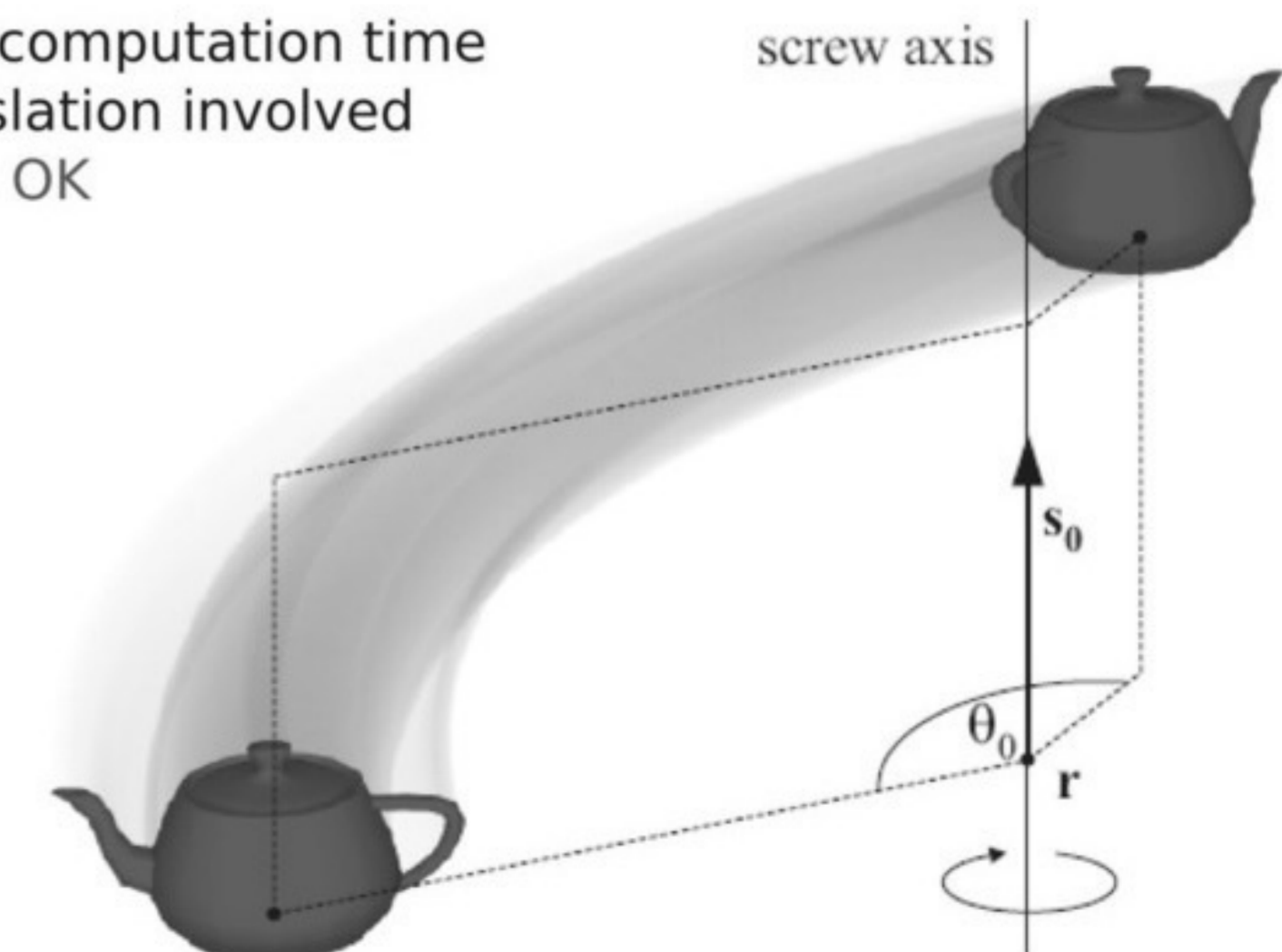


038

Skinning improvement

Improving interpolation

- Problem of linear interpolation of deformation
- Exponential map: computation time
- Quaternions: translation involved
- Dual Quaternions: OK



039

Other approaches

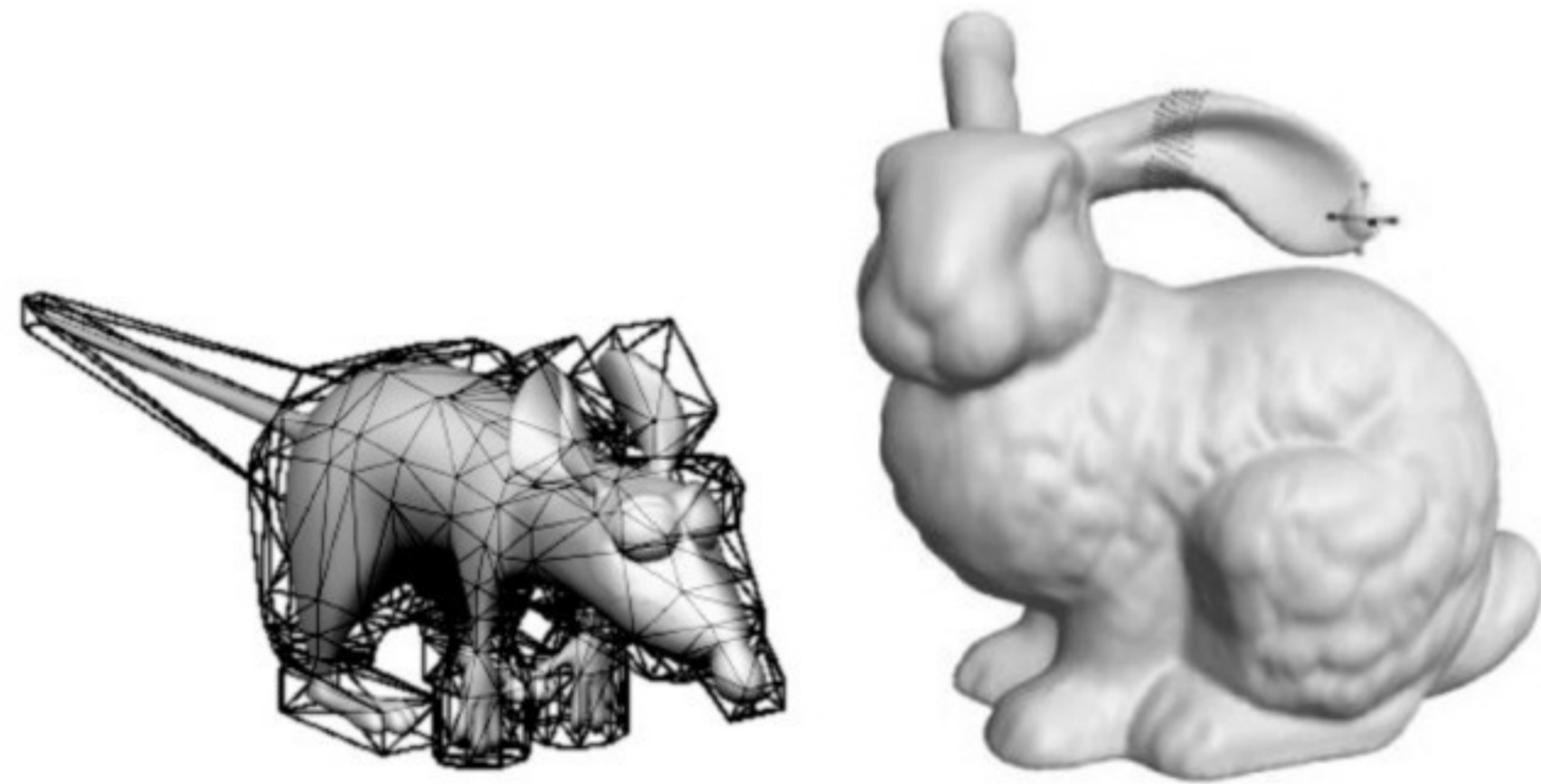
Skinning = **S**keleton **S**ubspace **D**eformation

Face animation: Shape interpolation

Smooth deformation: FFD

Hierarchical deformation: Multires

Free deformation: Matrix inversion+constraints



040