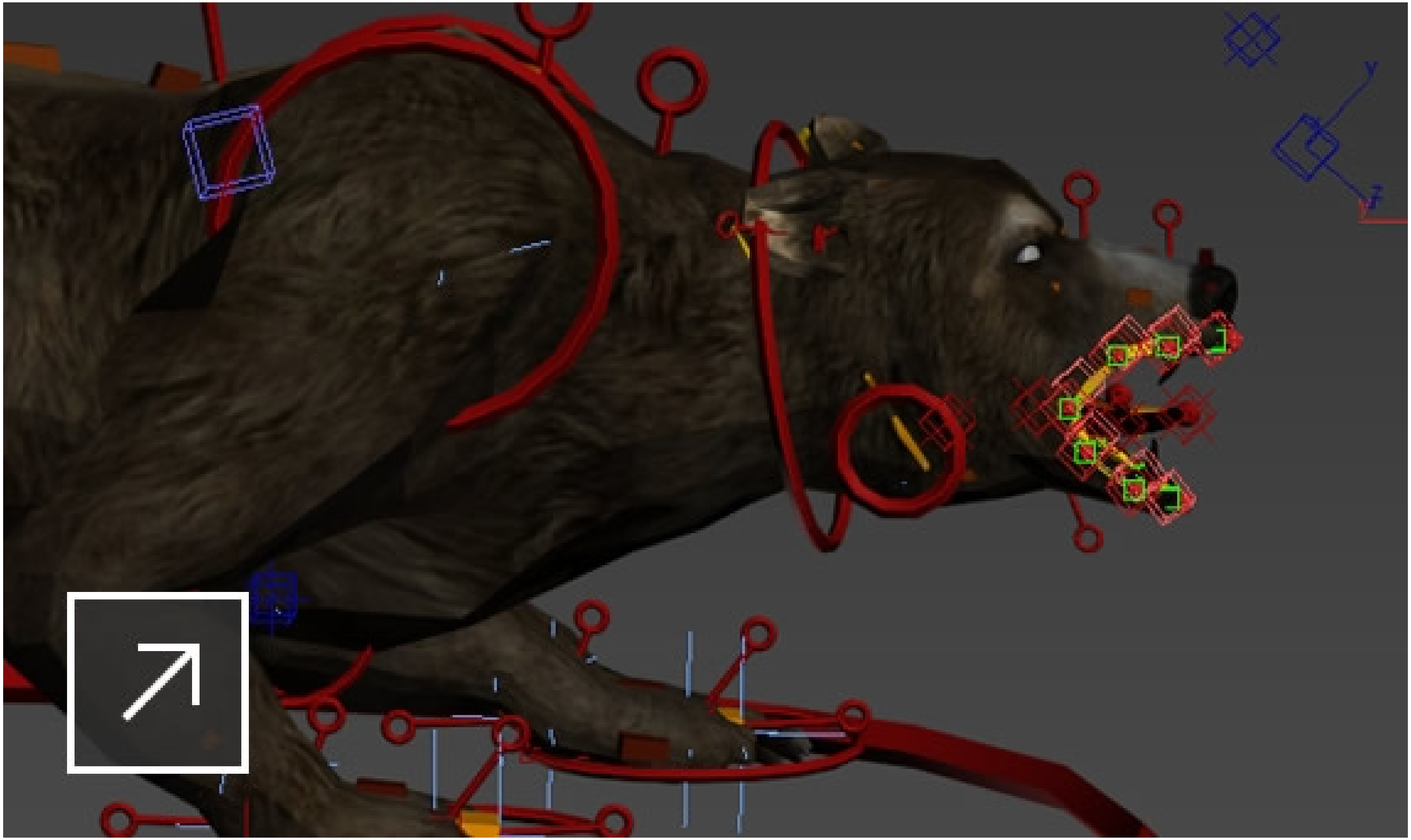
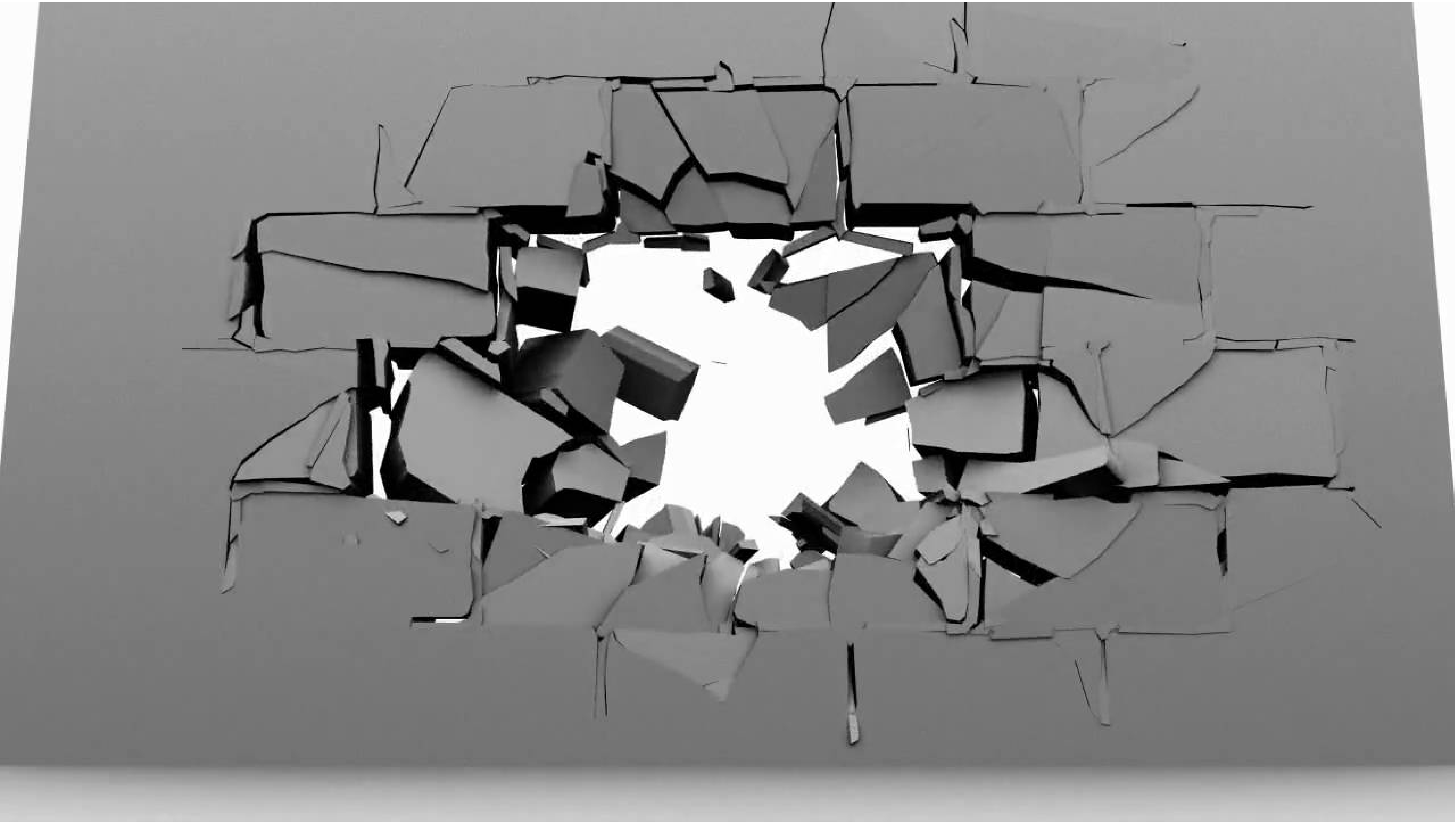
Shape deformation

What is deformation ?

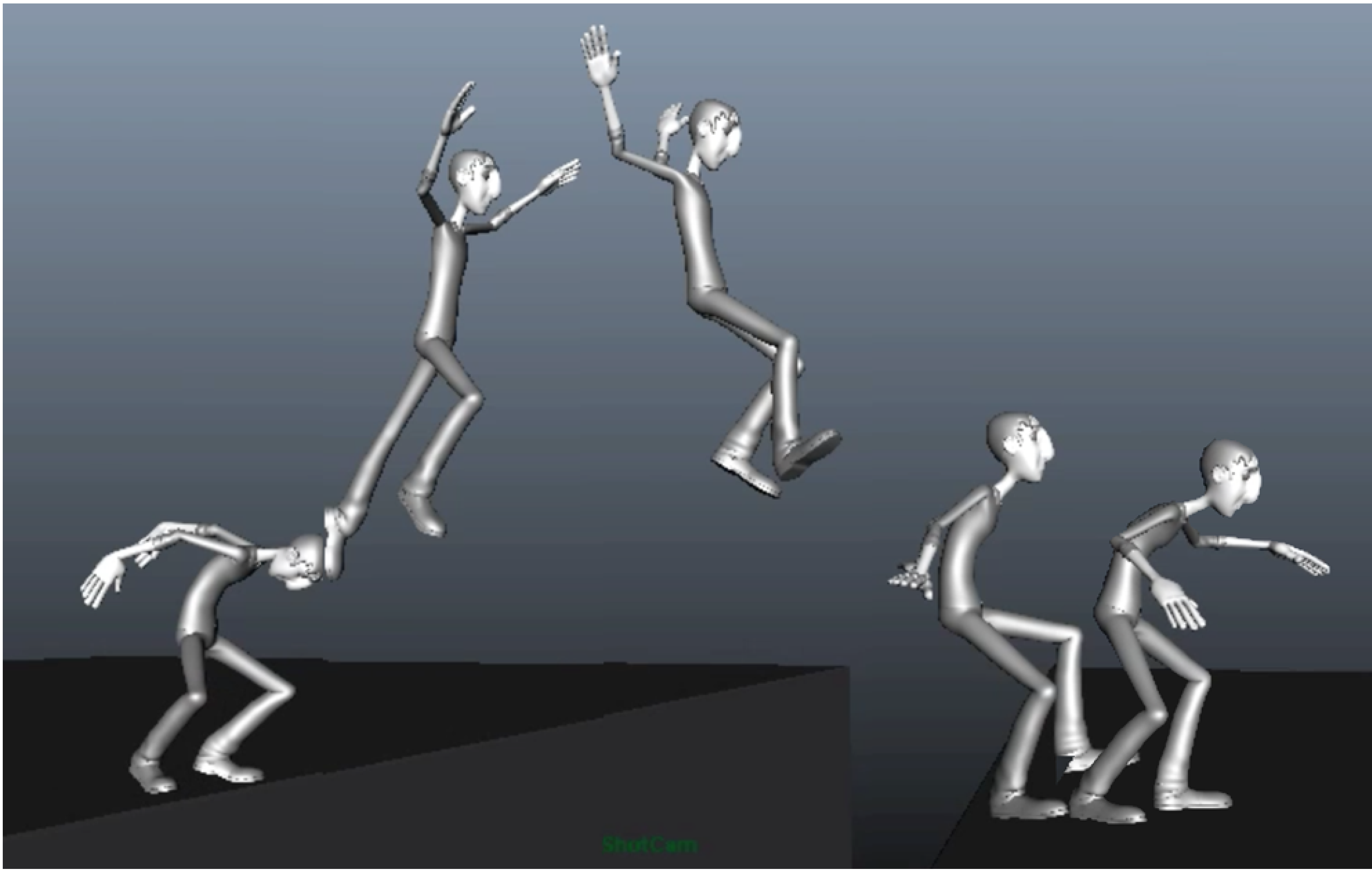


What is deformation ?

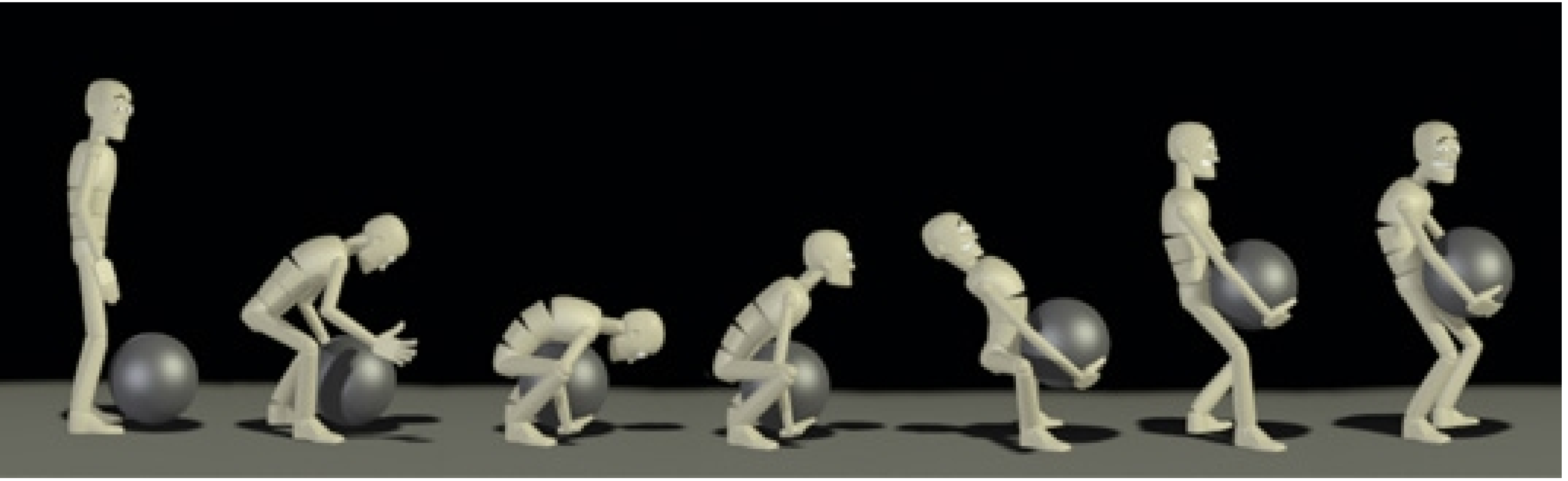


What is deformation ?

What is deformation ?



What is deformation ?

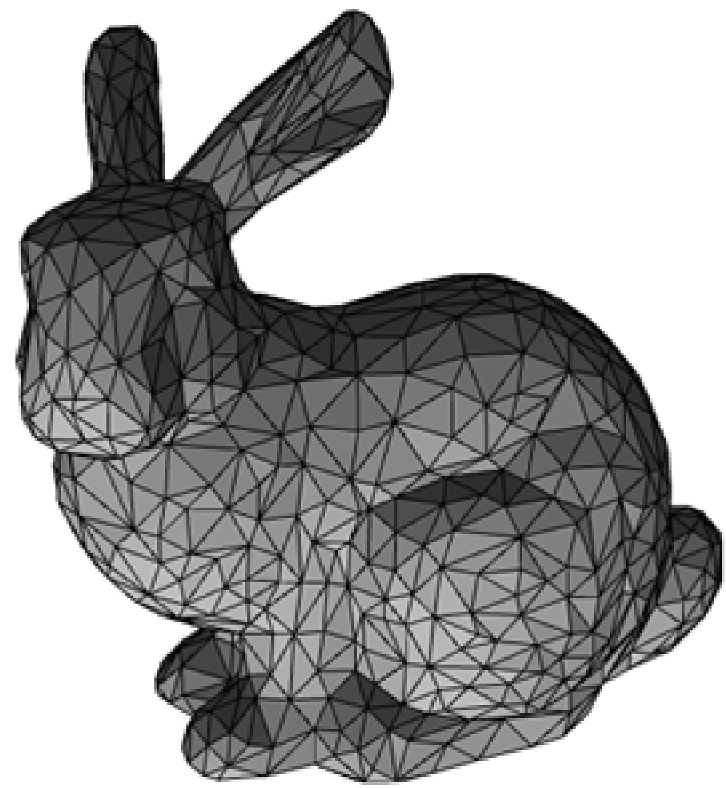


What is deformation ?

For today

- Direct manipulation of a surface (a shape is deformed when a user grabs vertices and move them)
- Deformation transfer (a shape is deformed similarly to another shape)
- Shape interpolation (a shape is the result of a blending of various poses)
- Mathematical properties of a deformation function, powerful algorithms to solve difficult and ill-defined problems

Manipulation of a MESH !!!



As-rigid-as-possible modelling

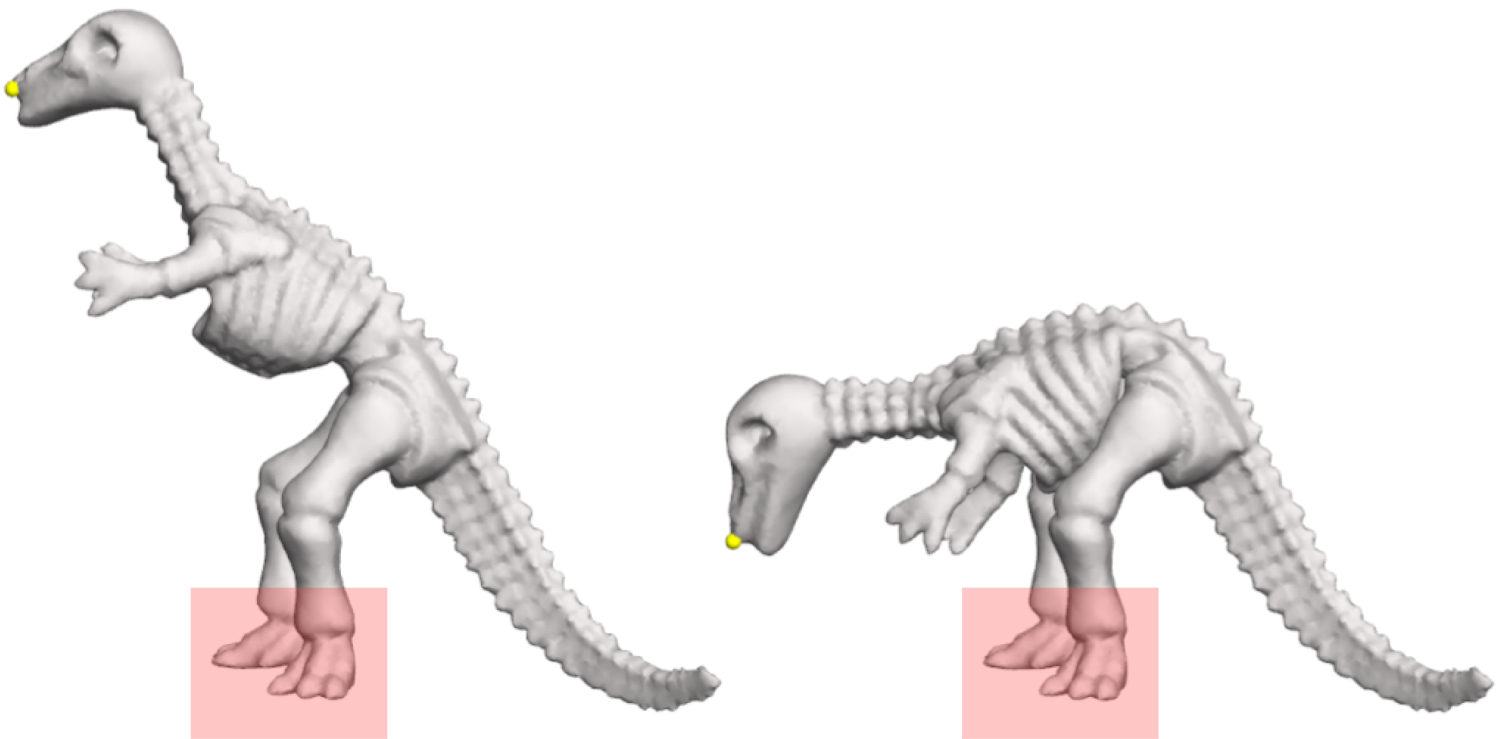
Why ARAP for today's lesson ?

- State of the art for shape direct manipulation
- Will allow me to show you how to setup a linear system (basics of numerical optimization)
- Will present the Procustes problem (basic problem in geometry)

[Sorkine & Alexa] : As-Rigid-As-Possible Surface Modeling

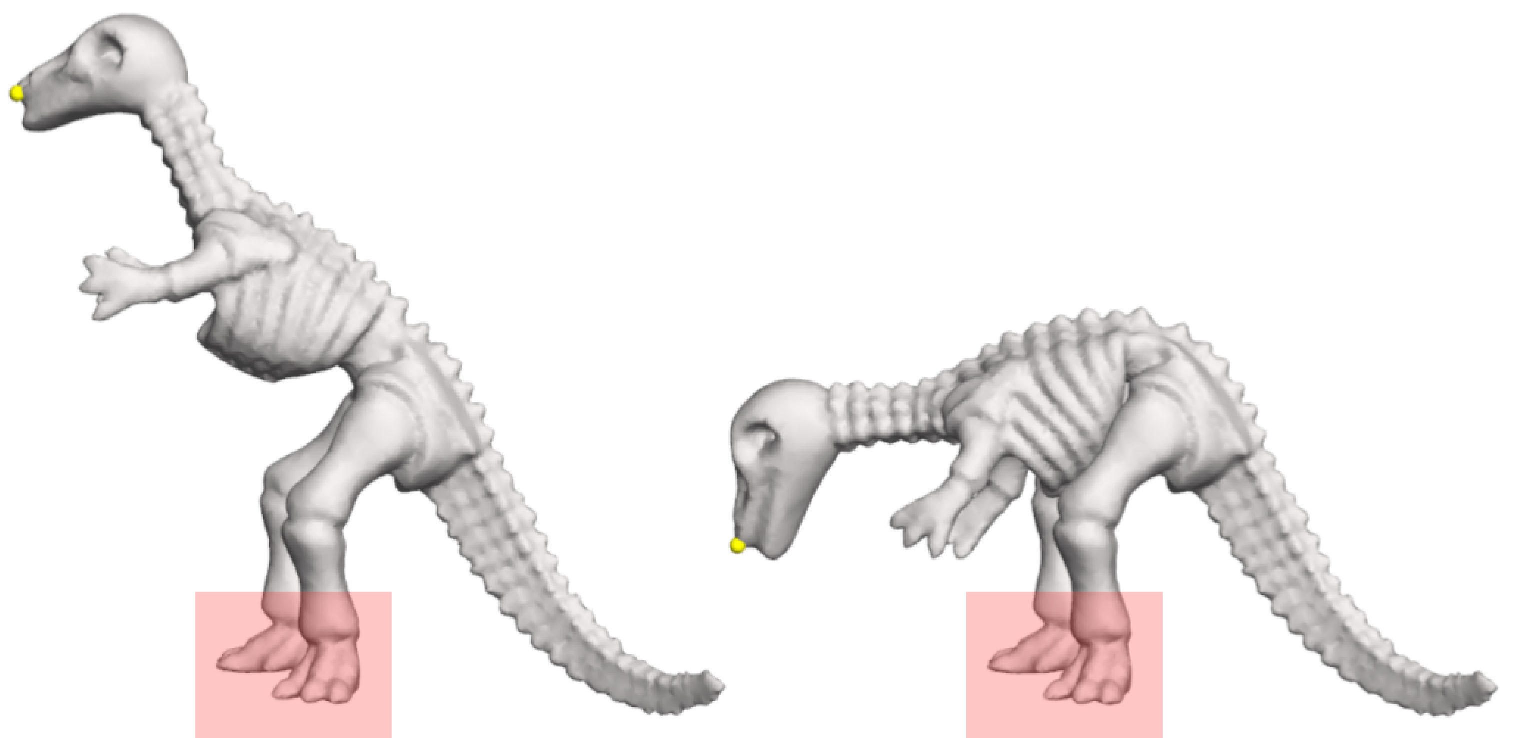
Principle

- A few vertex positions are specified by the user
- The other vertices should be placed in « a natural fashion »



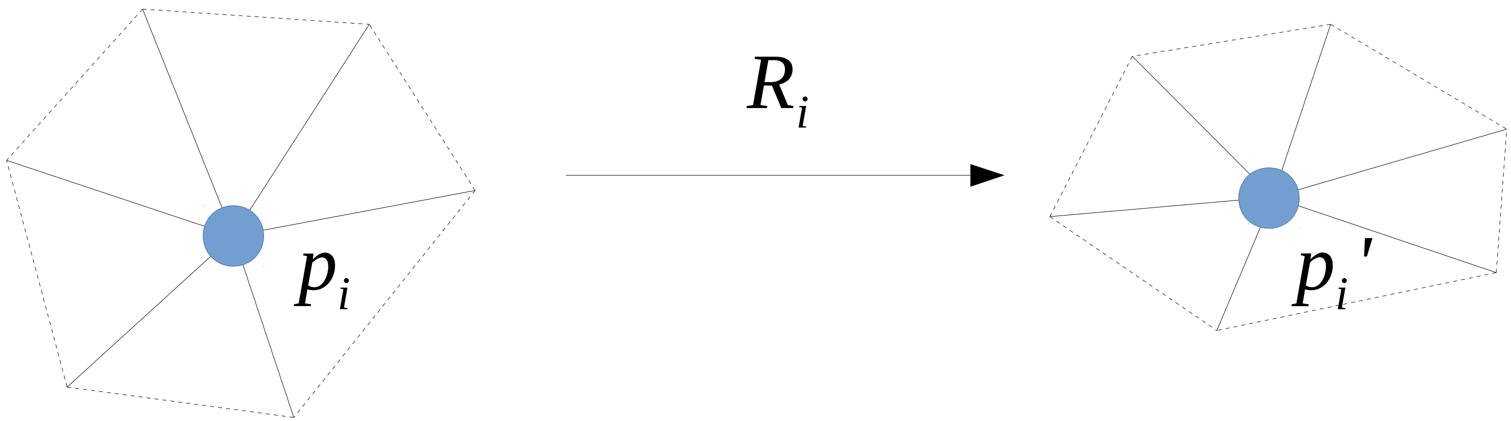
Minimizing stretch (enforcing rigidity)

- Physically « plausible »
- Impact on textures, stretch, volume



What is rigidity ?

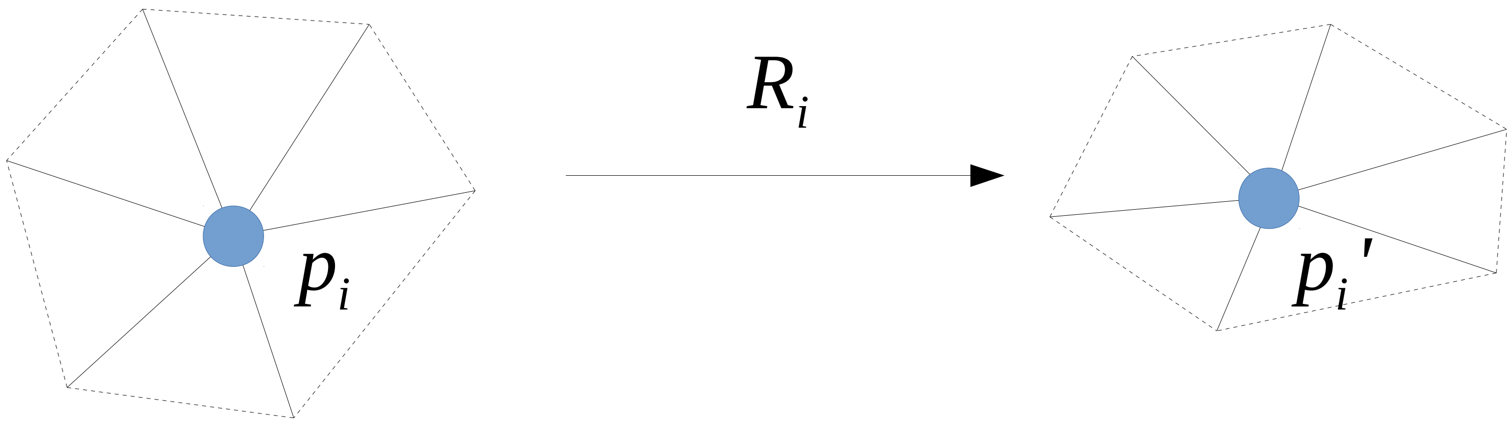
- A small piece of the shape should be globally rotated :



$$\mathbf{p}'_i - \mathbf{p}'_j = \mathbf{R}_i \left(\mathbf{p}_i - \mathbf{p}_j \right), \;\; \forall \; j \in \mathcal{N}(i)$$

Rigidity energy

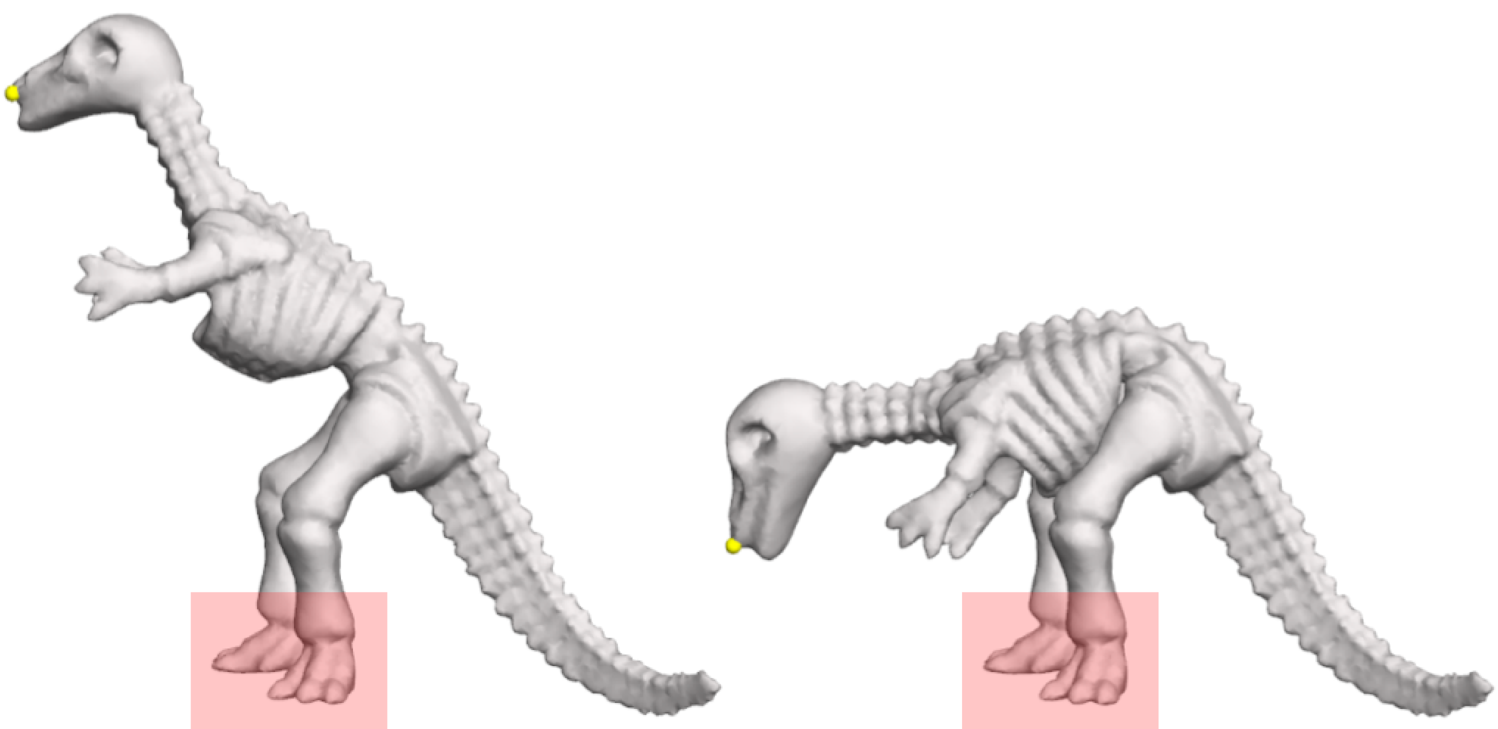
- A small piece of the shape should be globally rotated :



$$E(\mathcal{C}_i, \mathcal{C}_i') = \sum_{j \in \mathcal{N}(i)} w_{ij} \left\| (\mathbf{p}_i' - \mathbf{p}_j') - \mathbf{R}_i (\mathbf{p}_i - \mathbf{p}_j) \right\|^2$$

ARAP framework

- Unknowns : New positions p' AND rotations R
- Constraints : A few specified positions



Problem

- Unknowns : New positions p’ AND rotations R
- Highly NON-LINEAR and NON-CONVEX
- Minimizing R and p’ at the same time is not feasible
- This is the rigidity energy alone, don’t forget to add the handle energies !

$$E\left(\mathcal{S}'\right) = \sum_{i=1}^{n} w_i \sum_{j \in \mathcal{N}(i)} w_{ij} \left\| \left(\mathbf{p}'_i - \mathbf{p}'_j\right) - \mathbf{R}_i(\mathbf{p}_i - \mathbf{p}_j) \right\|^2$$

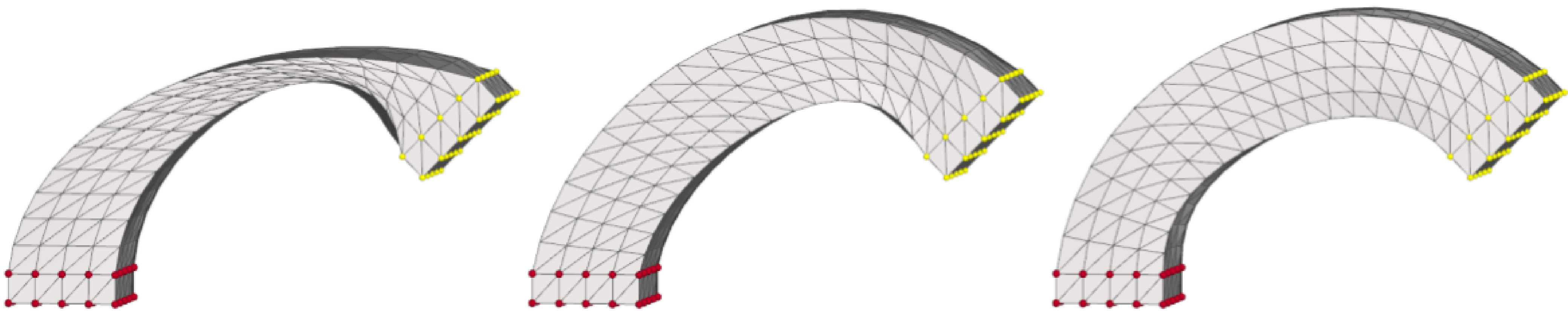
Ad hoc solution

- Fix R, optimize p’
- Fix p’, optimize R
- Fix R, optimize p’
- ...

$$E\left(\mathcal{S}'\right) = \sum_{i=1}^{n} w_i \sum_{j \in \mathcal{N}(i)} w_{ij} \left\|\left(\mathbf{p}'_i - \mathbf{p}'_j\right) - \mathbf{R}_i(\mathbf{p}_i - \mathbf{p}_j)\right\|^2$$

Ad hoc solution

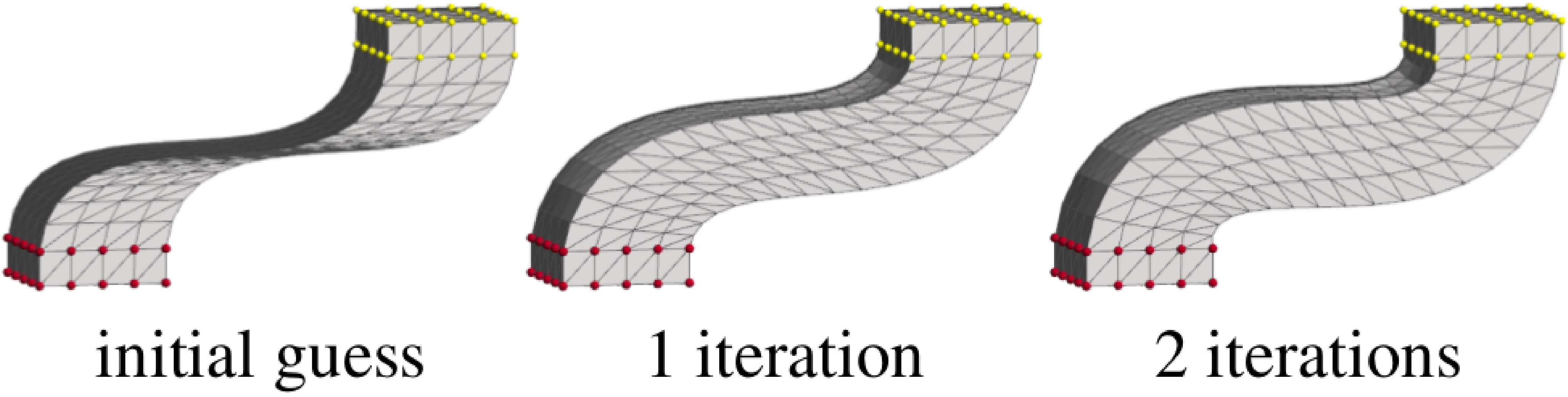
- Fix R, optimize p’
- Fix p’, optimize R
- Fix R, optimize p’
- ...



initial guess 1 iteration 4 iterations

Ad hoc solution

- Fix R, optimize p’
- Fix p’, optimize R
- Fix R, optimize p’
- ...



1) Fix R, optimize p’

- Linear system with p’ as unknowns
- Example on black board
- C++/Matlab : Cholmod, Eigen, ...
- Recall that ! Solving a linear system is the ABC of numerical optimization !

$$E\left(\mathcal{S}'\right) = \sum_{i=1}^{n} w_i \sum_{j \in \mathcal{N}(i)} w_{ij} \left\| \left(\mathbf{p}'_i - \mathbf{p}'_j\right) - \mathbf{R}_i(\mathbf{p}_i - \mathbf{p}_j) \right\|^2$$

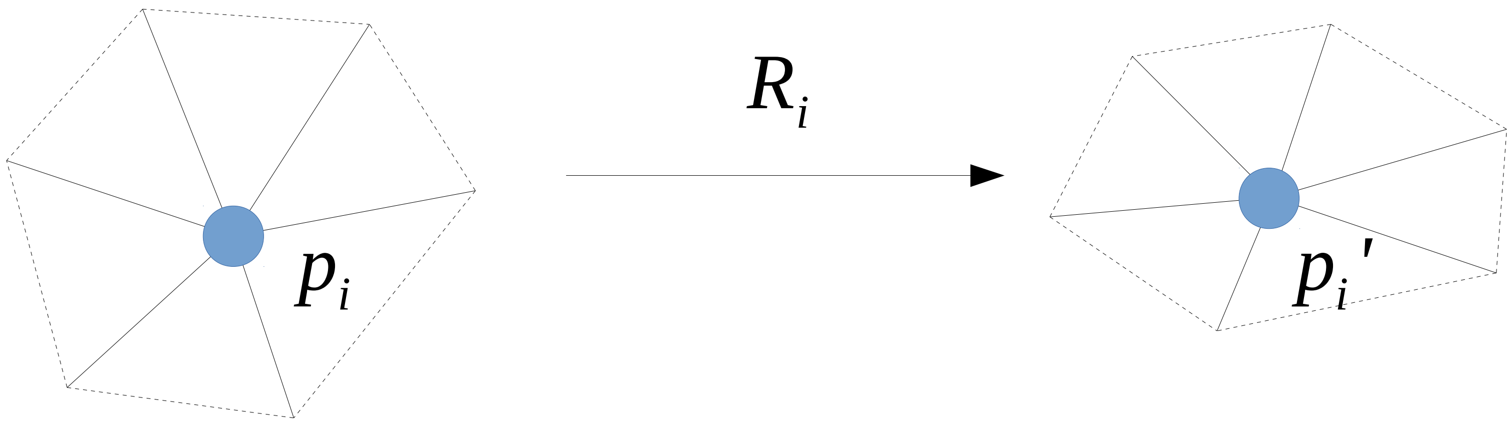
2) Fix p', optimize R

- Can be done per vertex i

$$E\left(\mathcal{S}'\right) = \sum_{i=1}^{n} w_i \sum_{j \in \mathcal{N}(i)} w_{ij} \left\|\left(\mathbf{p}'_i - \mathbf{p}'_j\right) - \mathbf{R}_i(\mathbf{p}_i - \mathbf{p}_j)\right\|^2$$

2) Fix p', optimize R

- Can be done per vertex i
- Minimize E(Ci, Ci')



$$E(\mathcal{C}_i, \mathcal{C}_i') = \sum_{j \in \mathcal{N}(i)} w_{ij} \left\| (\mathbf{p}_i' - \mathbf{p}_j') - \mathbf{R}_i (\mathbf{p}_i - \mathbf{p}_j) \right\|^2$$

2) Fix p', optimize R

$$\sum_j w_j \|e_j' - R.e_j\|^2 \longrightarrow$$ To minimize

$$\sum_j w_j \|e_j' - R.e_j\|^2 = \sum_j w_j \|e_j'\|^2 + \sum_j w_j \|R.e_j\|^2 - 2\sum_j w_j (R.e_j)^T.e_j'$$

$$\sum_j w_j \|e_j' - R.e_j\|^2 = const - 2\sum_j w_j Trace(R.e_j.e_j'^T)$$

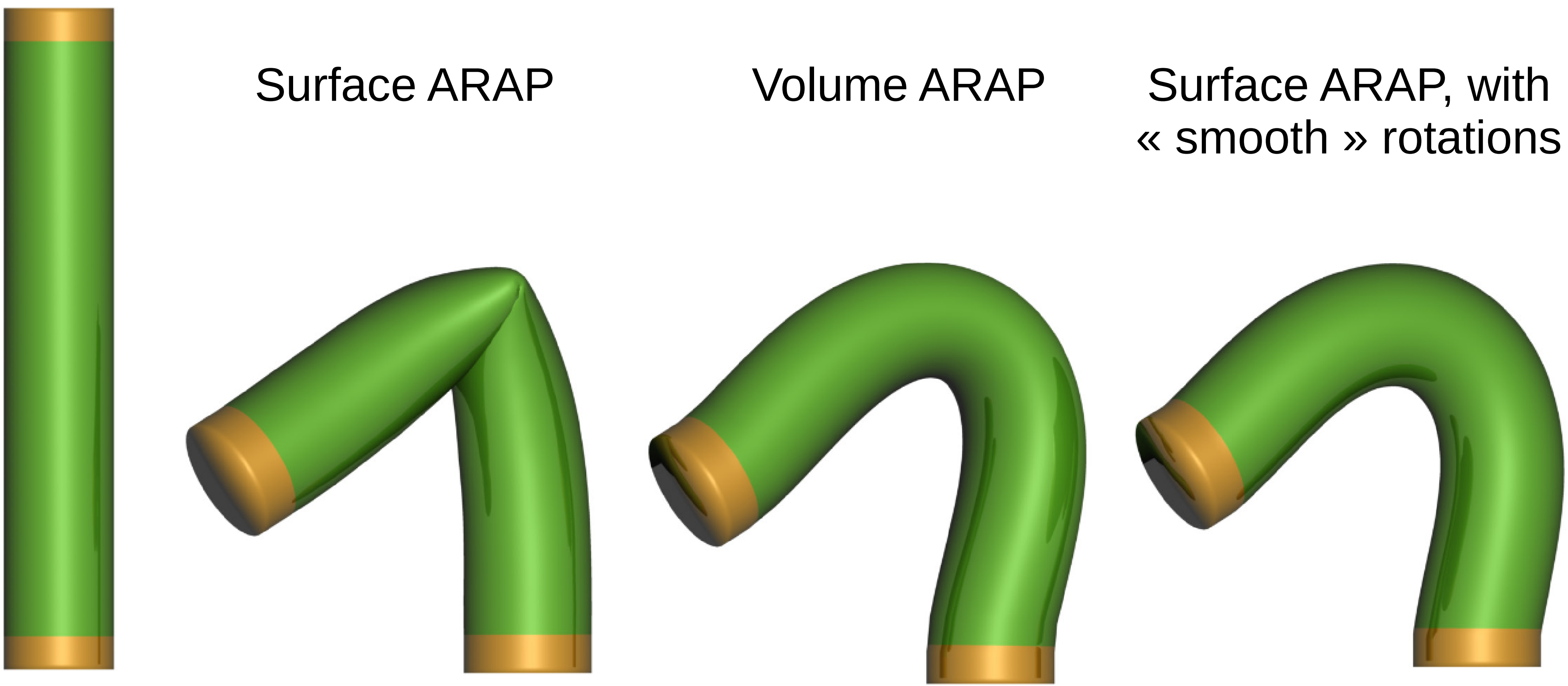
$$Trace(R.\sum_j w_j e_j.e_j'^T) \longrightarrow$$ To maximize

1) Build	$S := \sum_j w_j e_j'.e_j^T$
2) Compute SVD :	$S := U.\Sigma.V^T$
3) Solution :	$R := U.diag(1,1,det(U.V^T)).V^T$

At this point

- You know how to setup a linear system
- You know how to solve the Procustes problem
- You can implement ARAP in c++ with a few lines of code (something like 20 in Matlab)

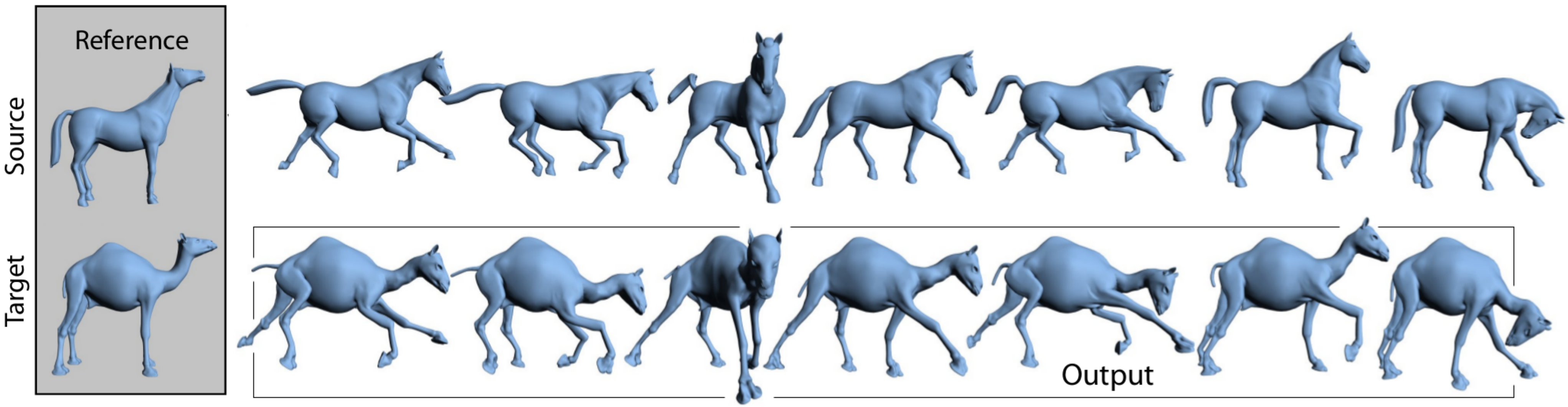
Different flavours of ARAP



[Levi & Gotsman] : Smooth Rotation Enhanced As-Rigid-As-Possible Mesh Animation

Deformation transfer

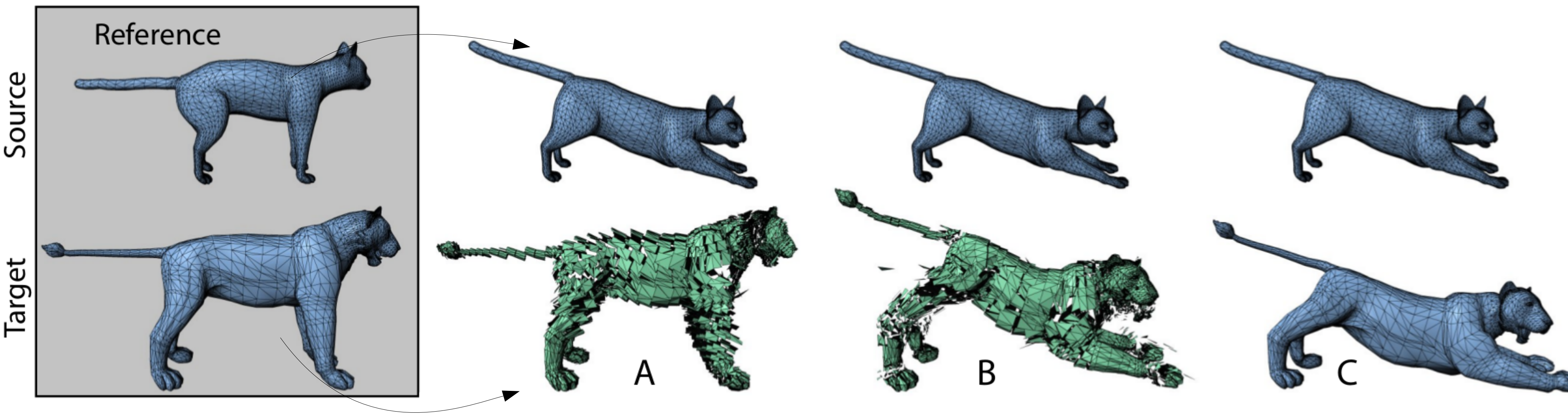
« Animation by example »



- Also called motion retargetting
- If skeletons are available, map the motion of one skeleton to the other : easy
- What can you « transfer » if you only have surfaces ?

[Sumner & Popović] : Deformation Transfer for Triangle Meshes

Transfer « local gradients », or local transformations

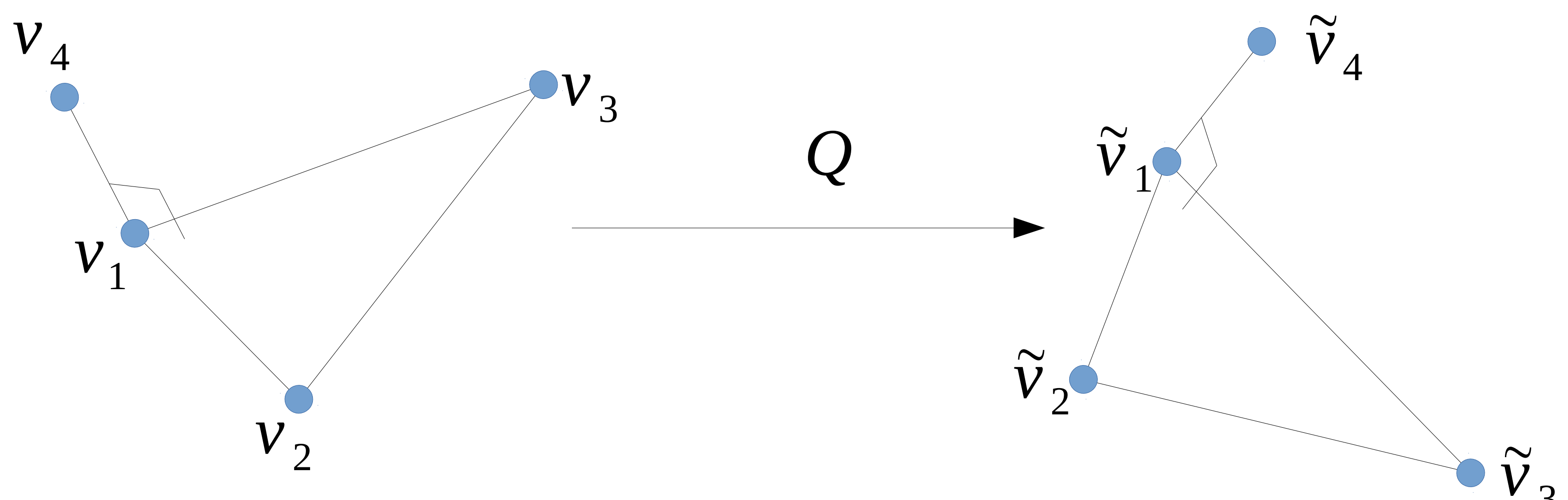


- Find transformation (3x3 matrix) of each triangle
- Transform similarly the triangles… but they are disconnected :(
- If you also transfer the translation… they are still disconnected :_(
- If you reconstruct the shape in the least-squares sense… magic happens

Seems familiar ? :)

- We just saw before, how one can reconstruct a shape, is the local transformations around the vertices are specified (ARAP)
- In the paper, they use a triangle-based formulation, but it is similar in spirit

Triangle transformation



$$\mathbf{Q}\mathbf{v}_i + \mathbf{d} = \tilde{\mathbf{v}}_i, \quad i \in 1 \ldots 4$$

$$\mathbf{V} = \left[\mathbf{v}_2 - \mathbf{v}_1 \;\; \mathbf{v}_3 - \mathbf{v}_1 \;\; \mathbf{v}_4 - \mathbf{v}_1\right]$$
$$\tilde{\mathbf{V}} = \left[\tilde{\mathbf{v}}_2 - \tilde{\mathbf{v}}_1 \;\; \tilde{\mathbf{v}}_3 - \tilde{\mathbf{v}}_1 \;\; \tilde{\mathbf{v}}_4 - \tilde{\mathbf{v}}_1\right]$$

$$\mathbf{Q} = \tilde{\mathbf{V}}\mathbf{V}^{-1}$$

Vertex reconstruction

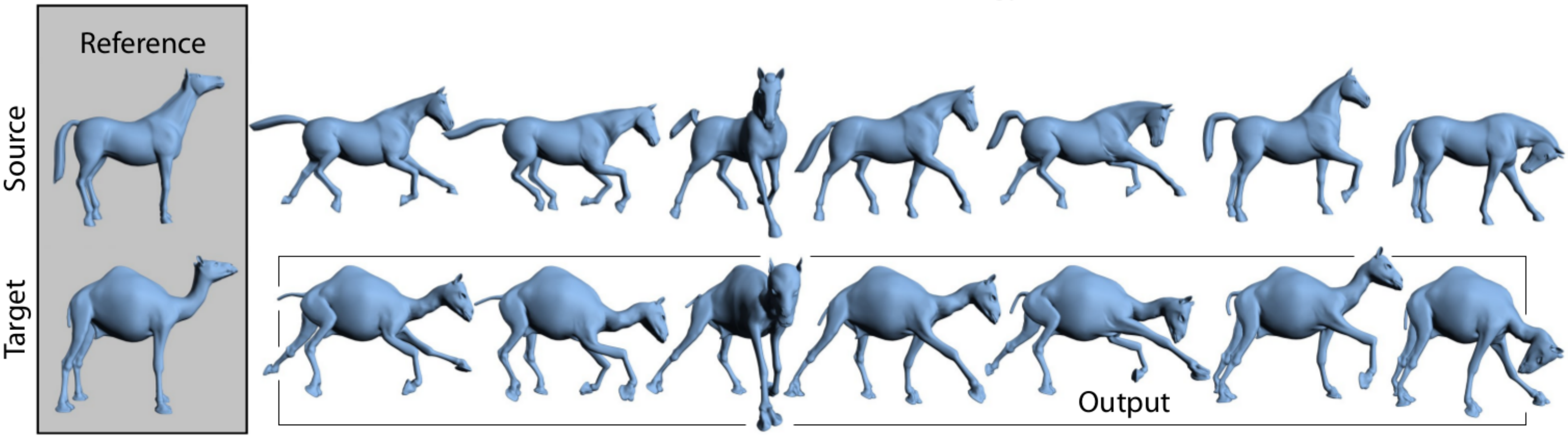
- Each triangle t should be oriented according to Q_t
- We need to « glue » the triangles (find the new vertex positions v’)
- We minimize (for example) the energy :

$$\sum_t \|(v'_{t_2}-v'_{t_1})-Q_t.(v_{t_2}-v_{t_1})\|^2+\|(v'_{t_3}-v'_{t_2})-Q_t.(v_{t_3}-v_{t_2})\|^2+\|(v'_{t_1}-v'_{t_3})-Q_t.(v_{t_1}-v_{t_3})\|^2$$

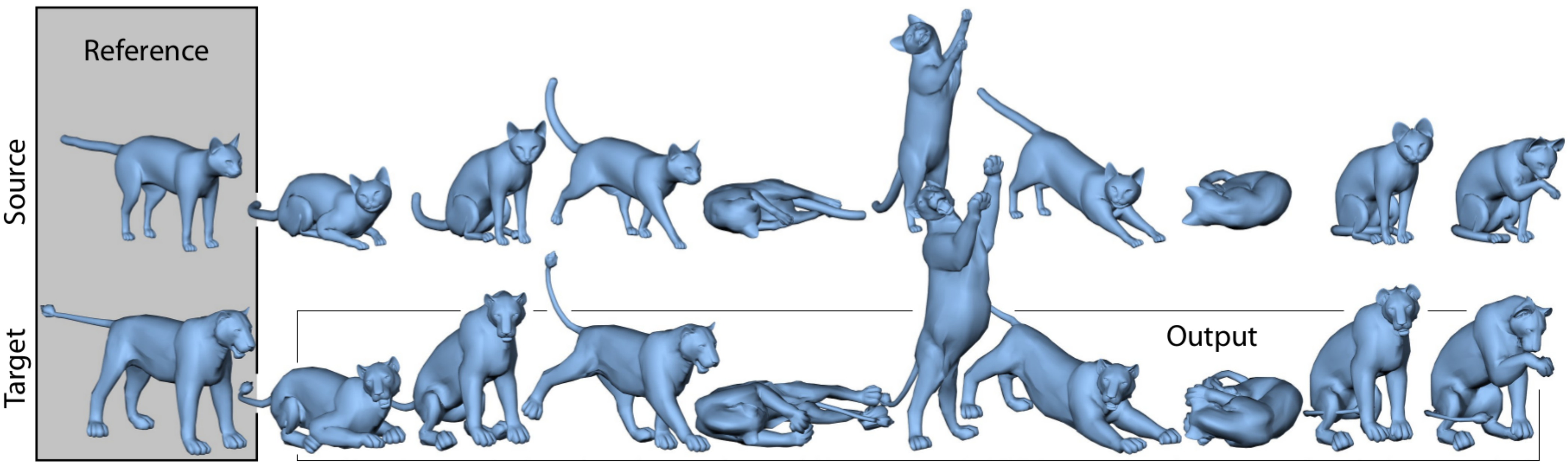
It’s a linear system ! Again !

Don’t forget to specify one vertex position per component

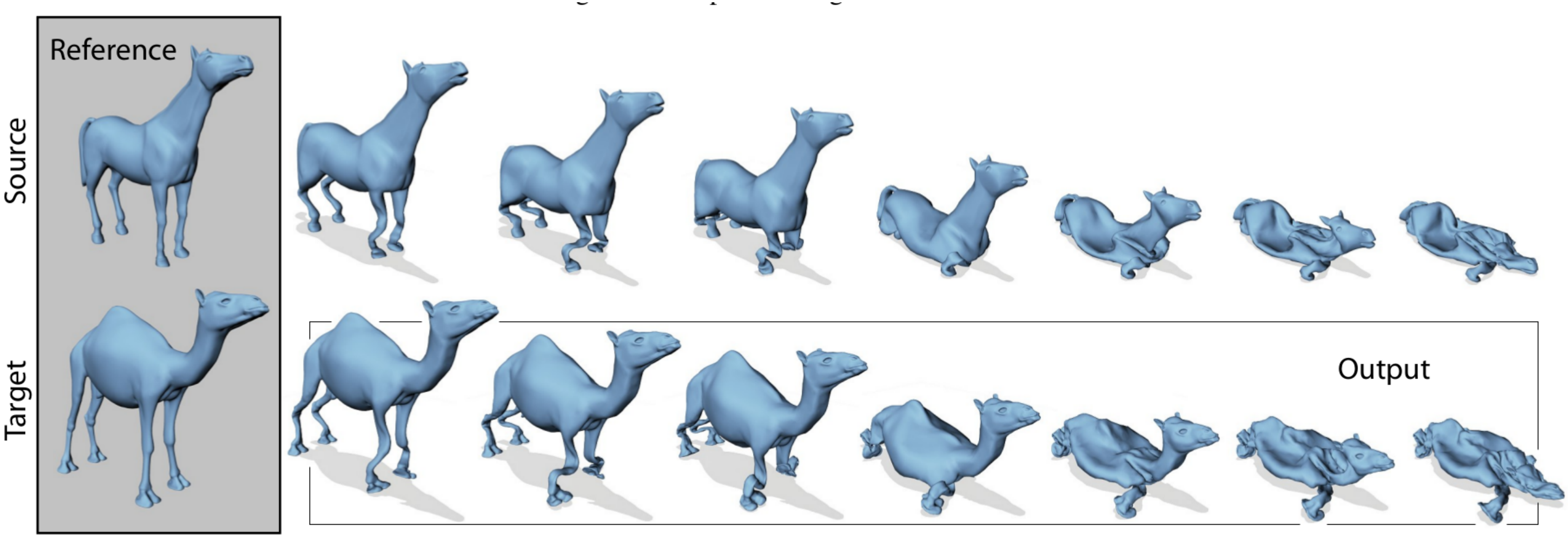
Results



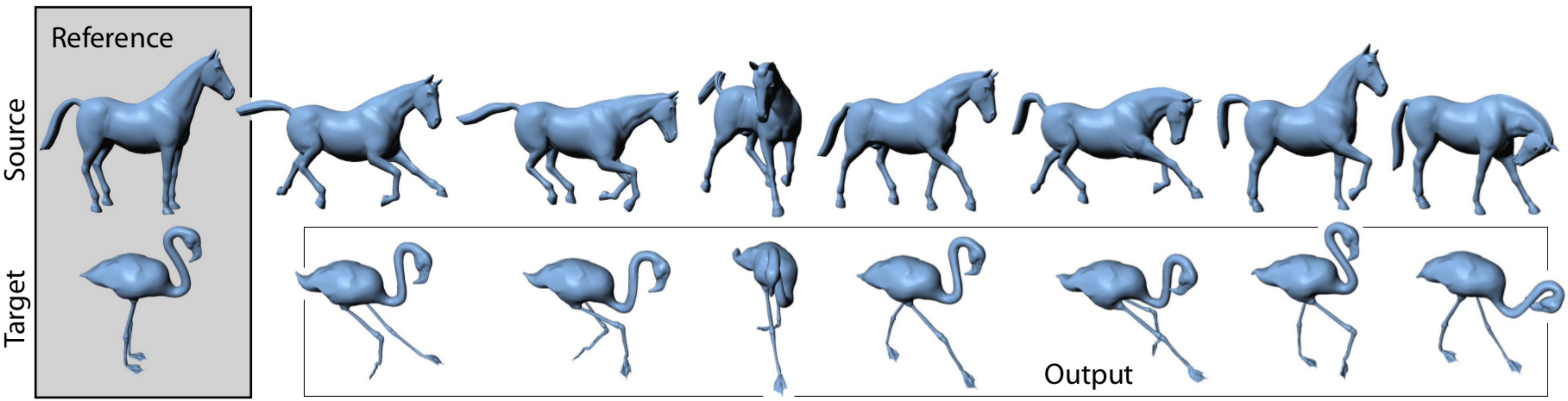
Results



Results



Results



Results

Similarities with Poisson editing

- Poisson problem on a mesh :
  - Specify per-triangle gradients {g_i} of a per-vertex function f
  - Reconstruct function f (linear system)

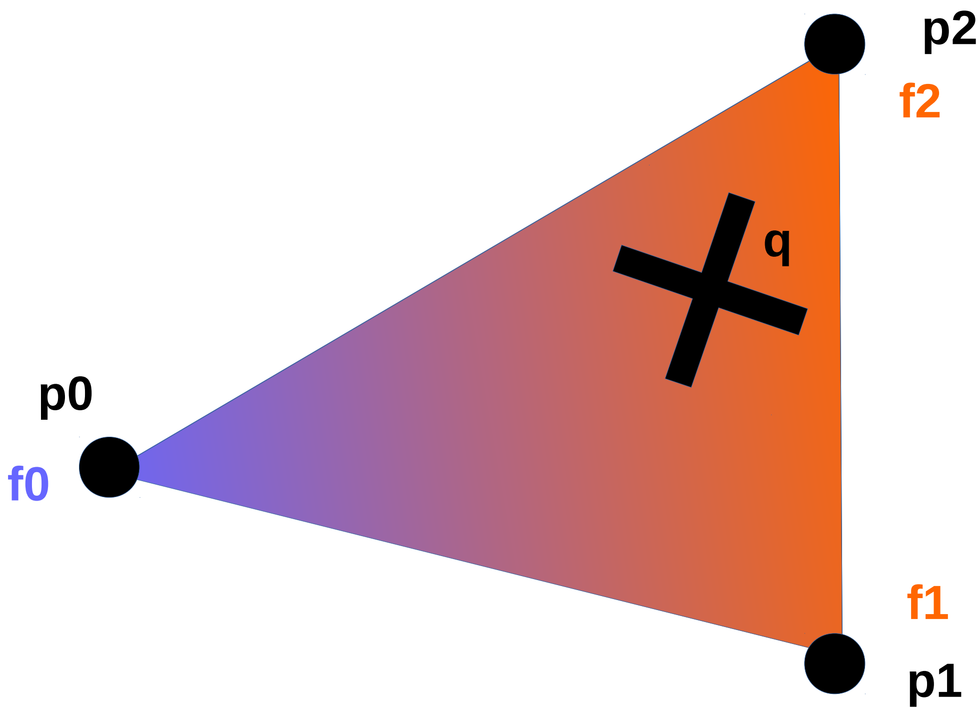
- Ex : specify gradients of x,y,z → similar to what we have seen :
  - Gradients are computed on the target and source base meshes
  - Transformations per triangle are computed just like before
  - Gradients of x,y,z are transformed on the target mesh
  - Positions can be recovered by integrating them

- More general : you can integrate plenty of different functions !

[Yu et al.] : Mesh Editing with Poisson-Based Gradient Field Manipulation

Gradient of a scalar function on a triangle

- Scalar functions are interpolated linearly on the triangles



Value of f at point q :

$$f(q)=\phi_0(q).f_0+\phi_1(q).f_1+\phi_2(q).f_2$$

$$(\phi_0(q),\phi_1(q),\phi_2(q))$$

Barycentric coordinates of q inside t!

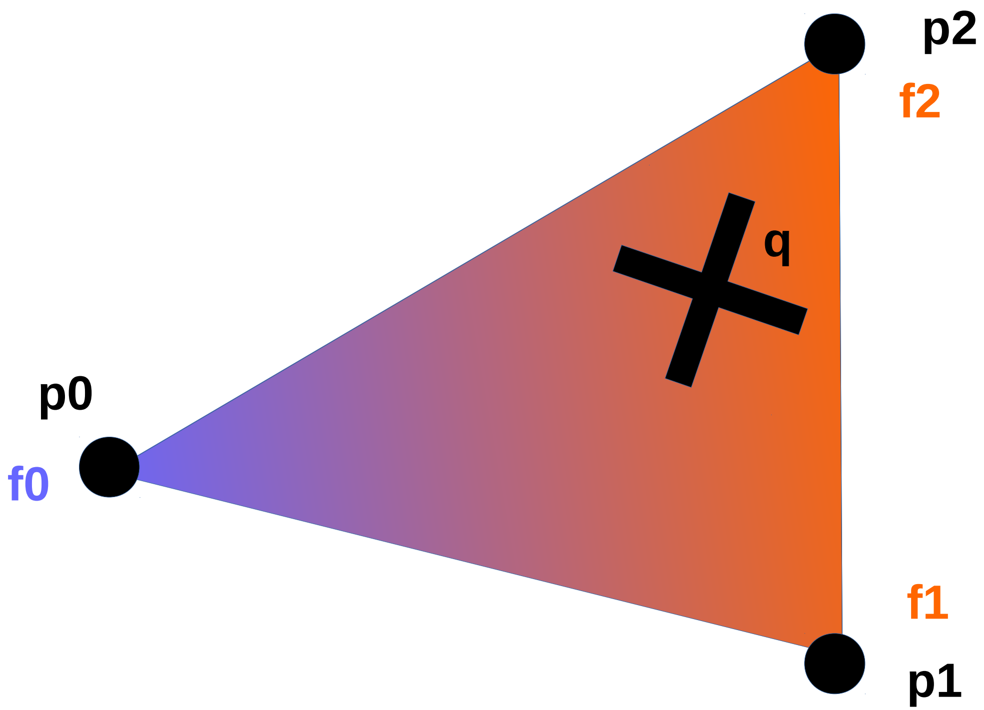
$$f(p0)=\phi_0(p0).f_0+\phi_1(p0).f_1+\phi_2(p0).f_2$$

$$f(p0)=1.f_0+0.f_1+0.f_2$$

$$f(p0)=f_0$$

Gradient of a scalar function on a triangle

- Gradients are constant inside a triangle, and easy to compute



Value of f at point q :

$$f(q)=\phi_0(q).f_0+\phi_1(q).f_1+\phi_2(q).f_2$$

Gradient of f at point q :

$$\nabla f(q)=\nabla\phi_0(q).f_0+\nabla\phi_1(q).f_1+\nabla\phi_2(q).f_2$$

Gradient is constant inside the triangle :

$$\nabla f=\nabla\phi_0.f_0+\nabla\phi_1.f_1+\nabla\phi_2.f_2$$

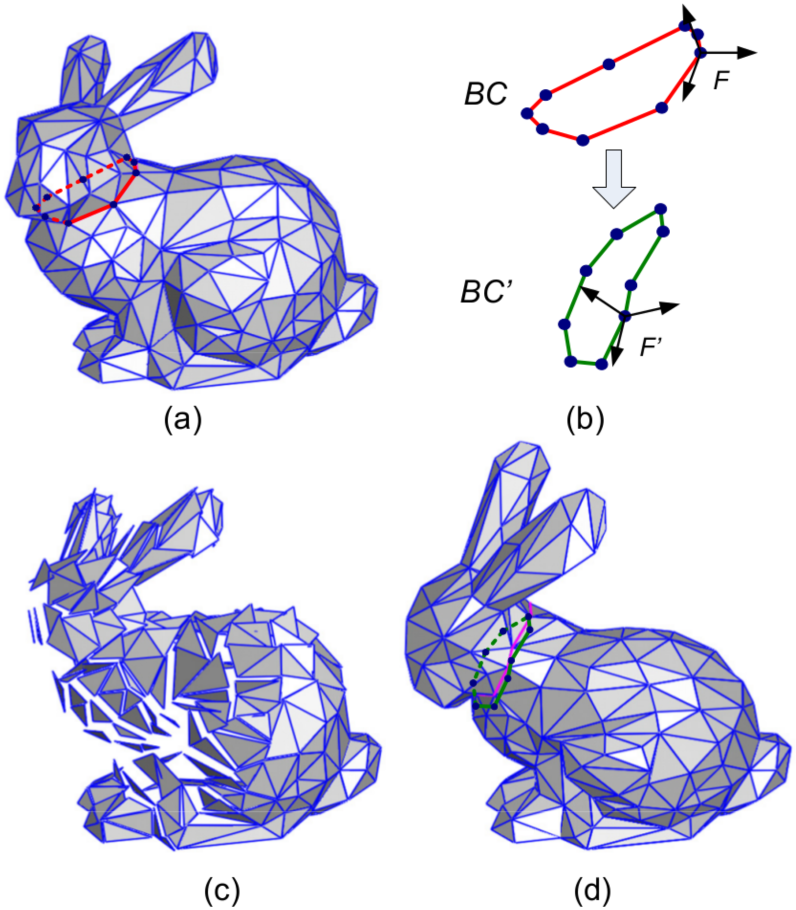
Solving a Poisson equation on a mesh

$$\nabla \phi = \mathbf{w}$$

$$\longrightarrow \quad \mathrm{Div}(\nabla \phi) = \mathrm{Div}\mathbf{w}$$

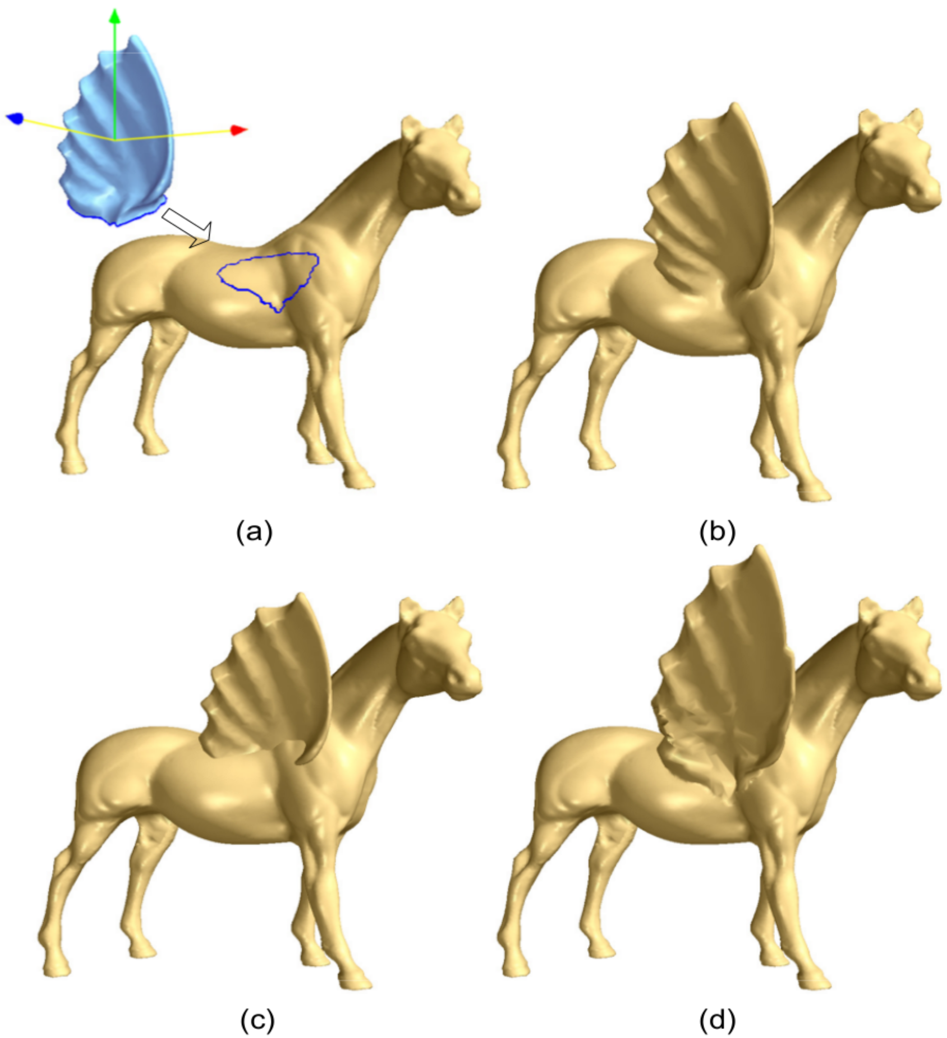
$$(\mathrm{Div}\mathbf{w})(\mathbf{v_i}) = \sum_{T_k \in N(i)} \nabla B_{ik} \cdot \mathbf{w} |T_k|$$

« Boundary » based deformations



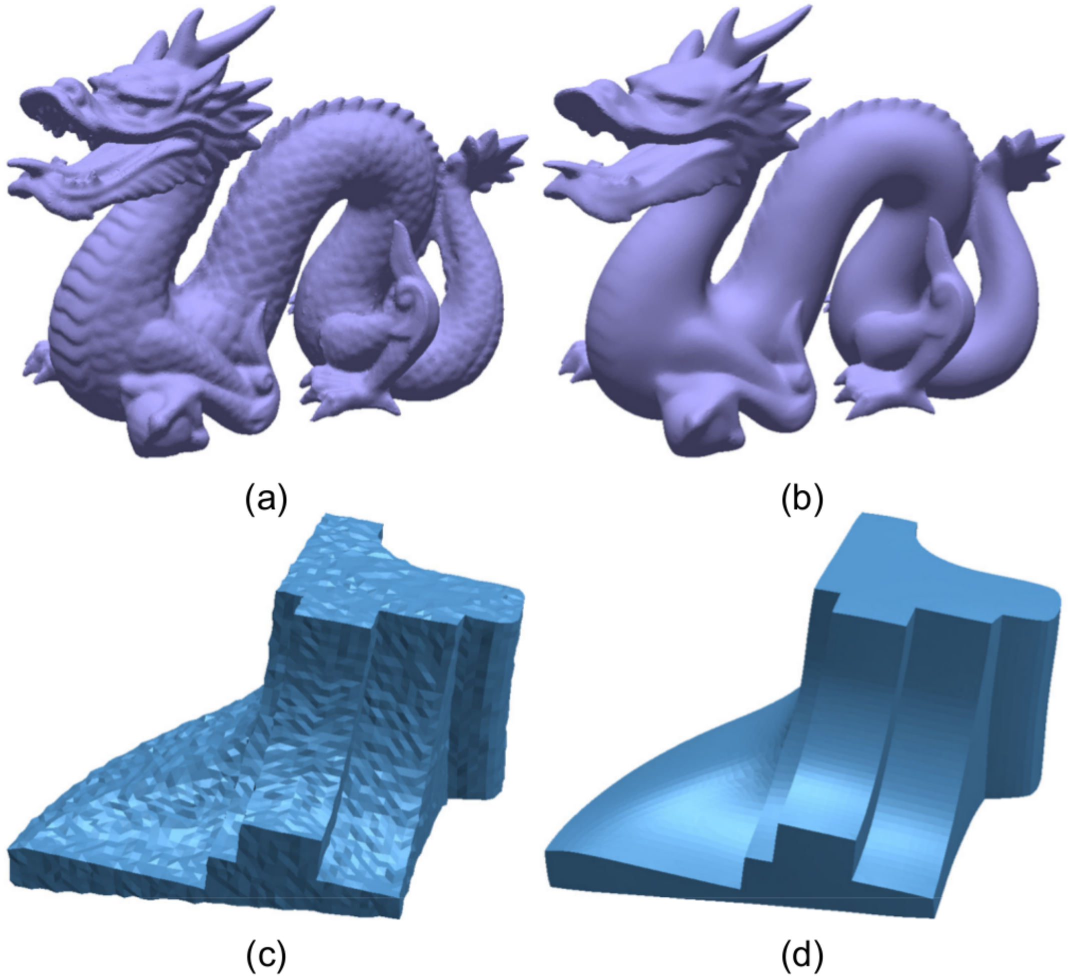
Propagate smoothly the transformations (weighted by distance)
Solve Poisson equation.

Merging



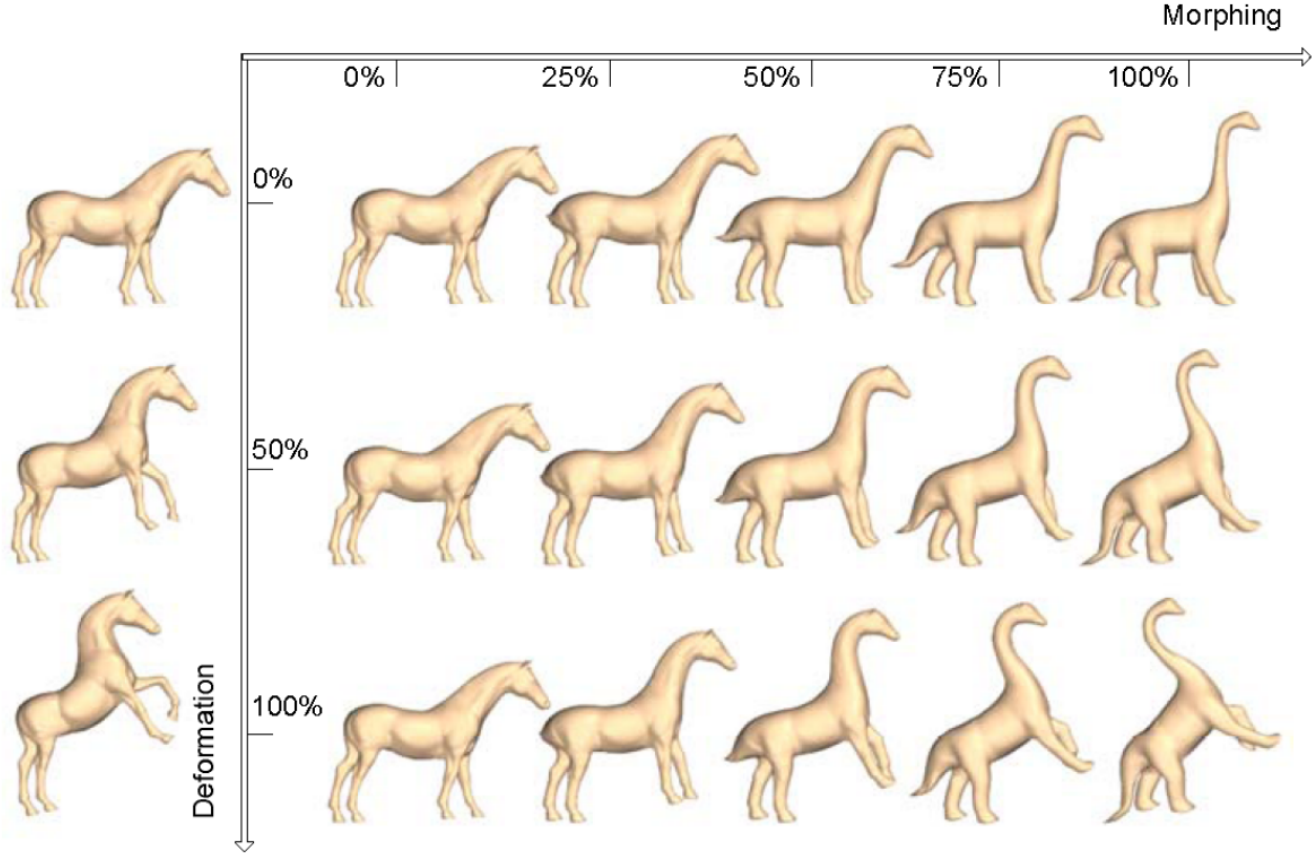
Propagate smoothly the transformations (weighted by distance)
Solve Poisson equation.

Denoising



Filter normals.
Reconstruct mesh from the normals using Poisson equation.

Shape interpolation



Each triangle has k transformation matrices (for k meshes).
Interpolate these transformation matrices, then solve Poisson eq.

[Xu et al.] : Poisson Shape Interpolation

Problem with Poisson shape interpolation

- Interpolating transformations is ill-posed
- Transformations are simply not the right quantity to interpolate !!!
- What is the right quantity to interpolate ?

Shape interpolation

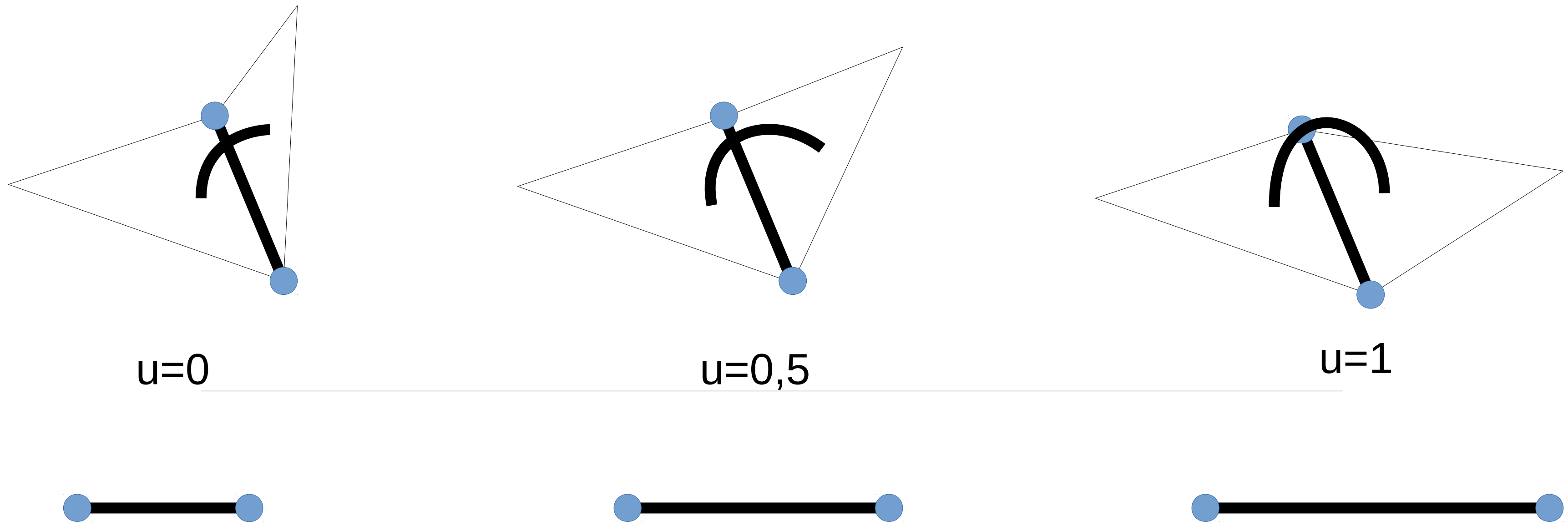
« Averaging 2 poses »

u=0 u=0,25 u=0,5 u=0,75 u=1

[Winkler et al.] : Multi-Scale Geometry Interpolation

« Averaging poses »

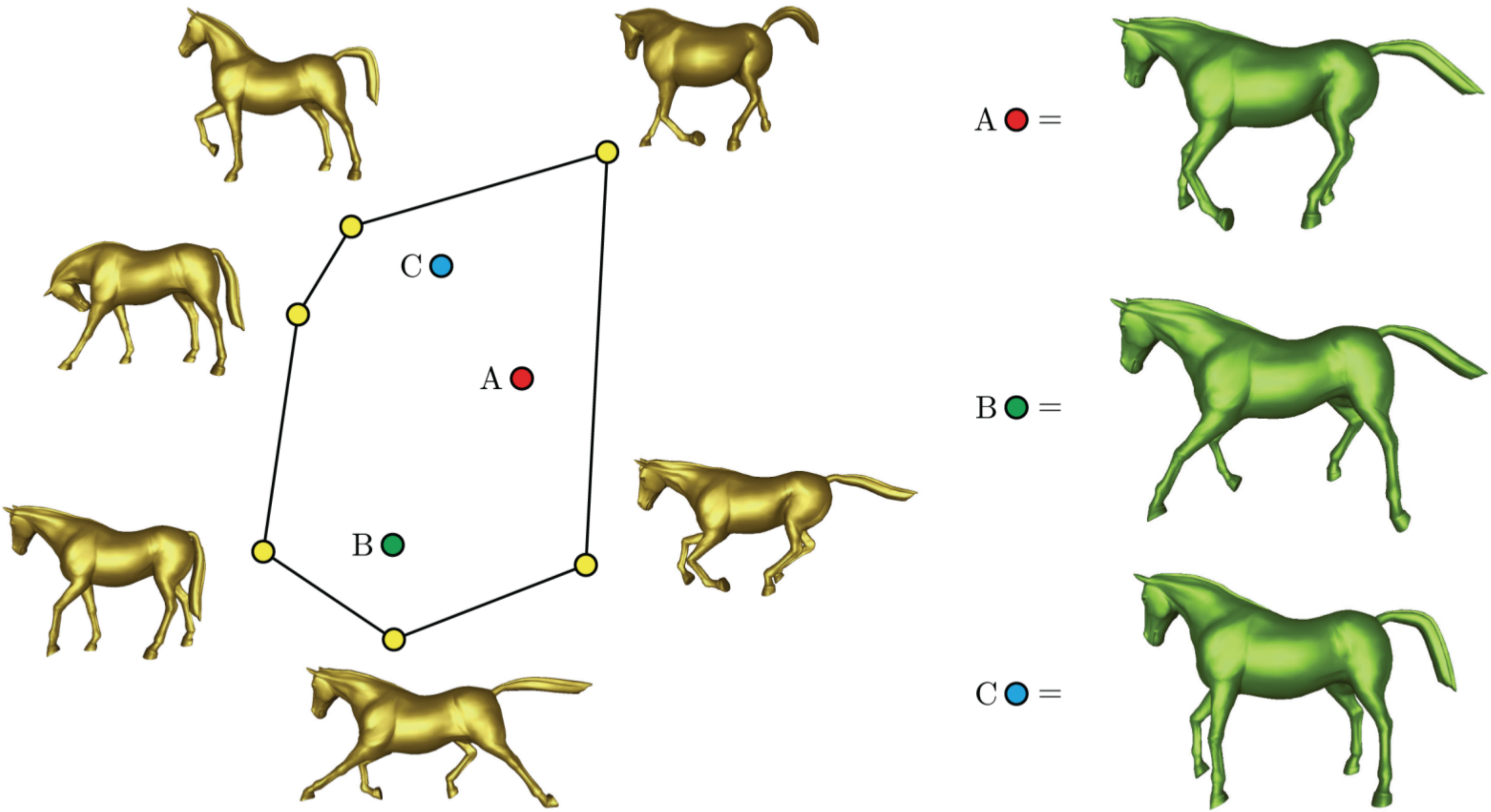
- K meshes with one-to-one vertex and edge connectivity
- Interpolate dihedral angles (curvature)
- Interpolate edge lengths (metric)



Extrapolation

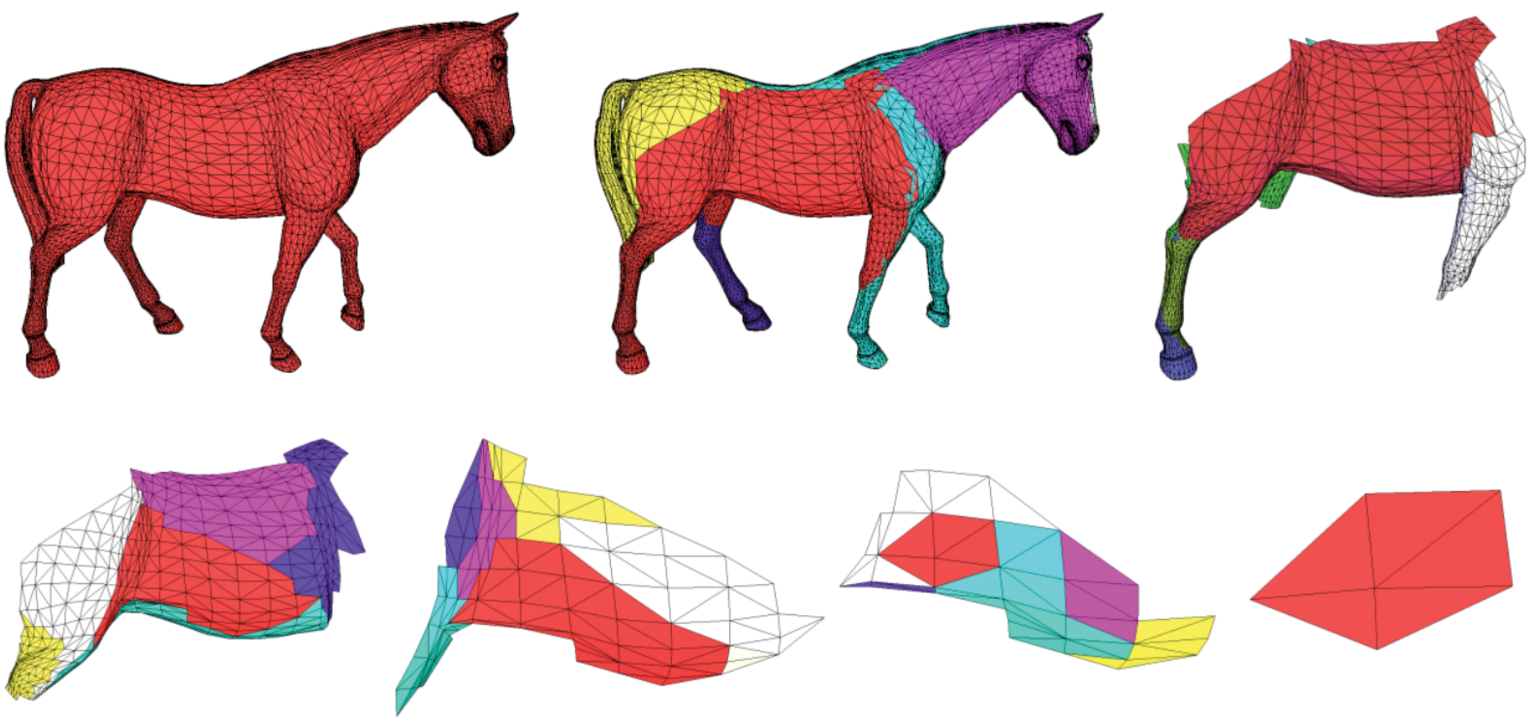
U=-0,25 u=0 u=0,5 u=1 U=1,25

« Averaging K poses »



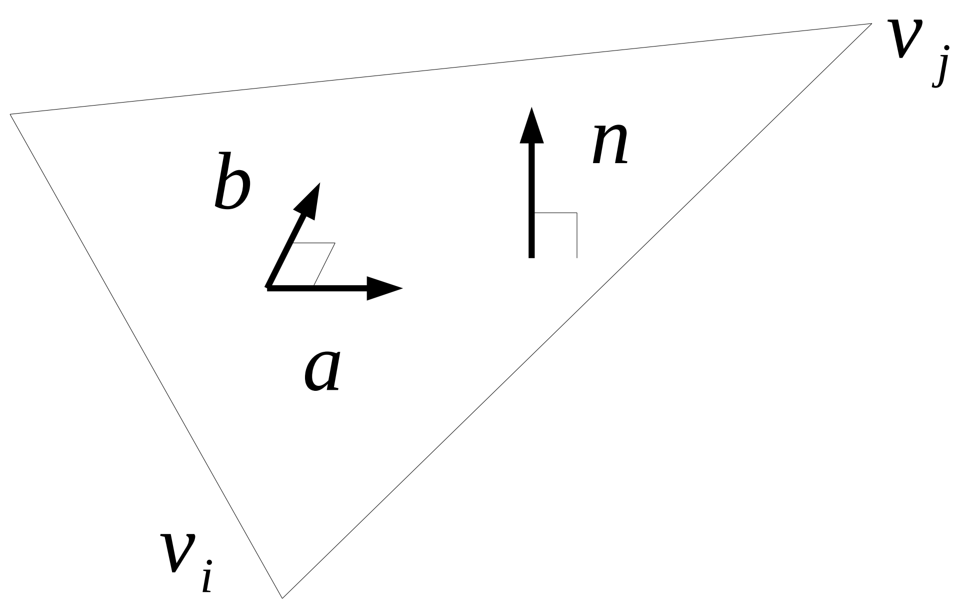
Geometrical construction

- Reconstruct small patches,
- Blend small patches to create bigger patches,
- Etc etc. (tedious, complex to implement, slow)



Algebraic construction

- Equip each triangle T with a local orthogonal frame f = (a, b, n)
- Express the three edges in this basis (2 coordinates per edge in this basis)



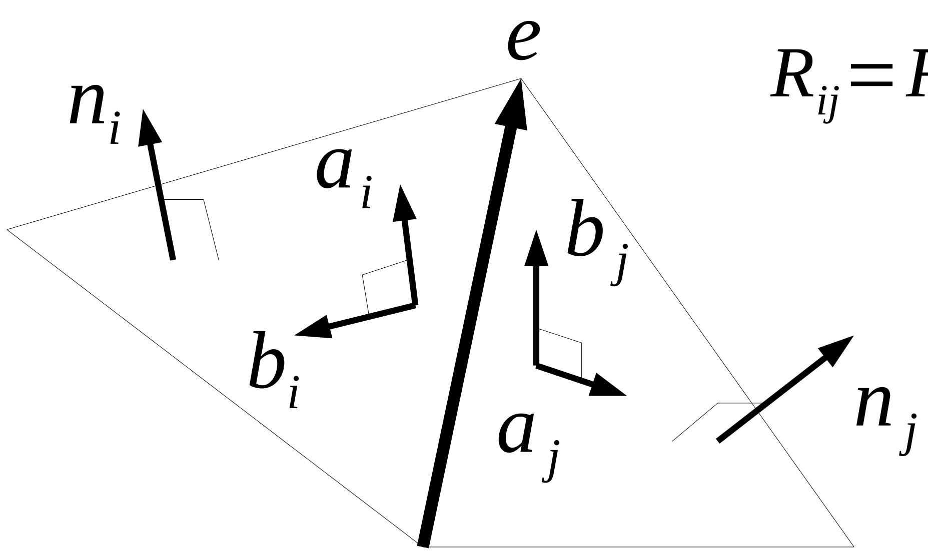
$$v_j - v_i = \alpha_{ij} . a + \beta_{ij} . b$$

$$\begin{pmatrix} \alpha_{ij} \\ \beta_{ij} \\ 0 \end{pmatrix} = f_t^T . (v_j - v_i)$$

Inspired from **[Wang et al.]** : Linear Surface Reconstruction from Discrete Fundamental Forms on Triangle Meshes

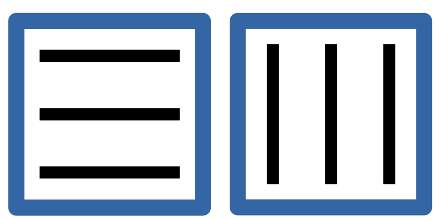
Algebraic construction

- From dihedral angles, construct respective orientation of adjacent triangles :



$$R_{ij}=R_z\left(\widehat{(a_i,e)}\right).R_x\left(\widehat{(n_i,n_j)}\right).R_z\left(\widehat{(e,a_j)}\right)$$

$$R_{ij}=f_i^{-1}.f_j$$



Key property : orientation-insensitive

Frames from dihedral angles

$$R_{ij} = f_i^{-1} . f_j$$

- Computed from desired dihedral angles

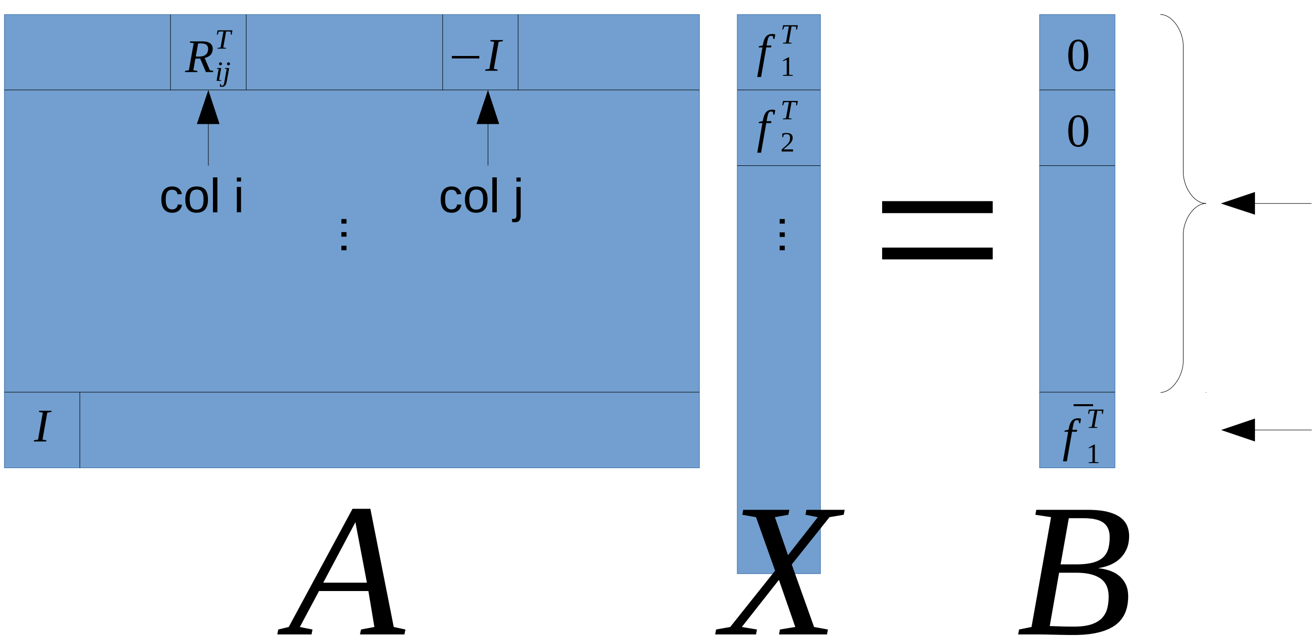
$$E_f = \sum_{e=(t_i \wedge t_j)} \| f_i . R_{ij} - f_j \|^2$$

- Enforces preservation of transitions $\{R_{ij}\}$
- Not practical in this form

$$E_f = \sum_{e=(t_i \wedge t_j)} \| R_{ij}^T . f_i^T - f_j^T \|^2$$

- Practical in this form (linear system $\|A . X - B\|^2$)
- Unknowns are **tranposed** frames

Linear system construction (each element is a 3x3 matrix) :



Setup equations for each pair of adjacent triangles i,j

Add a line to fix global orientation (one line per component actually)

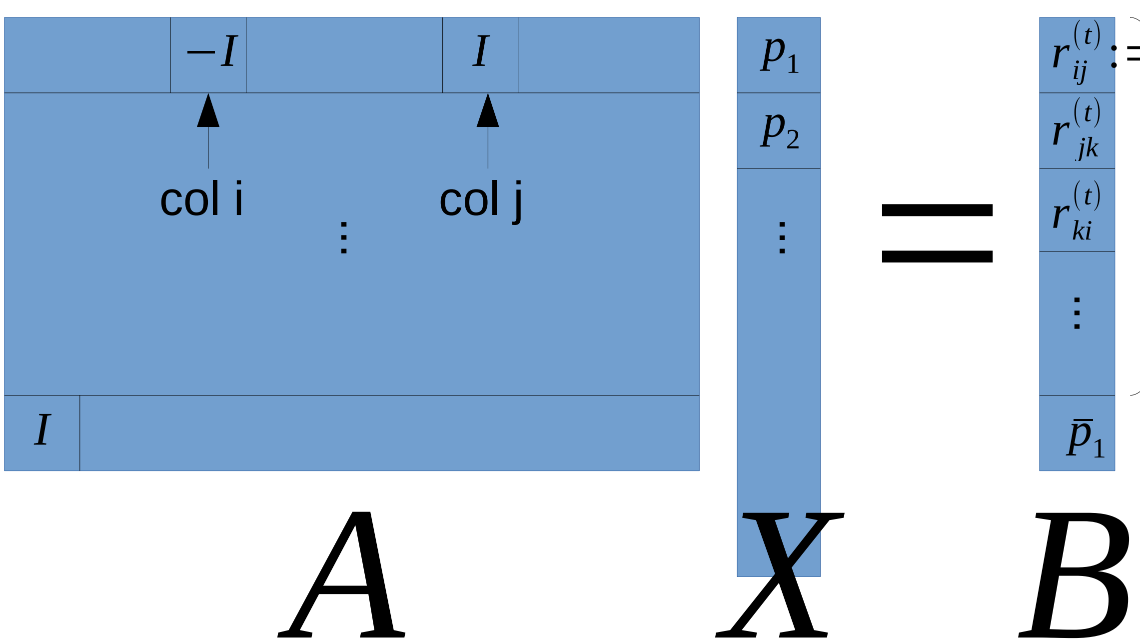
Positions from frames and edge lengths

New length New frame Similar for edges jk and ki

$$E_p = \sum_{t=(i,j,k)} \left\| (p_j - p_i) - \frac{l_{ij}'}{l_{ij}} (\alpha_{ij} . a_t + \beta_{ij} . b_t) \right\|^2 + \ \dots \ + \ \dots$$

Encoded length Encoded coeffs

Linear system construction is straightforward



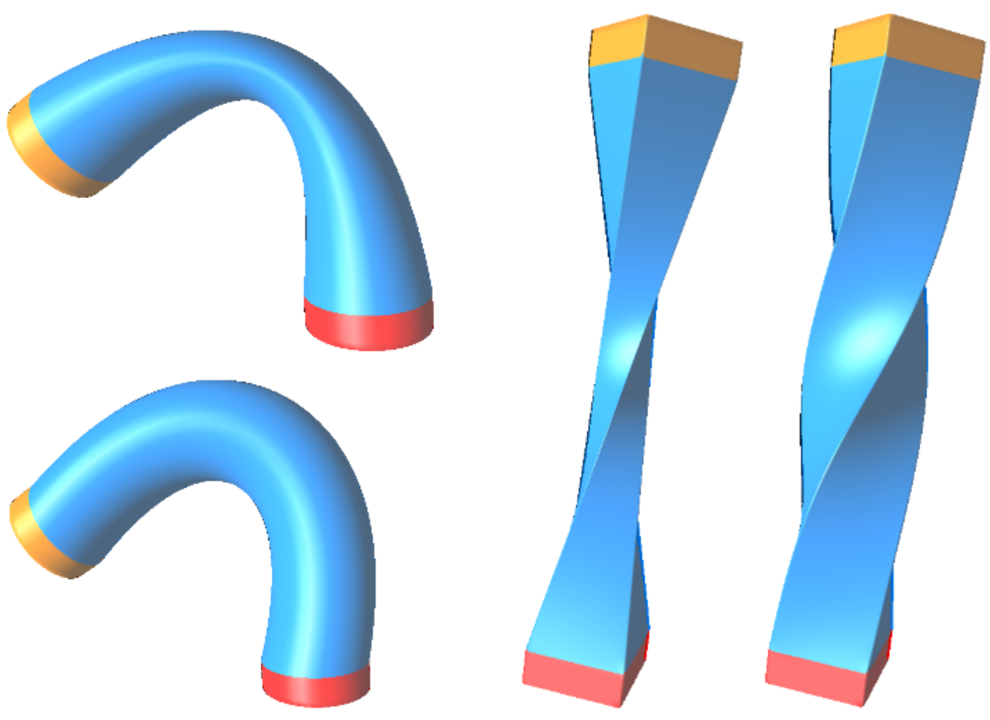
$$r_{ij}^{(t)} := \frac{l_{ij}'}{l_{ij}} (\alpha_{ij} . a_t + \beta_{ij} . b_t)$$

Setup 3 equations for each triangle (i,j,k)

Add a line to fix global position (one line per component actually)

With and without frame orthogonalization

- First solve for new frames in triangles, then for new positions for vertices
- After solving the frames system, the variables are not rotation matrices. Compare without and with orthonormalization :



Current research in Shape Deformation

Conformal 3d surface deformations

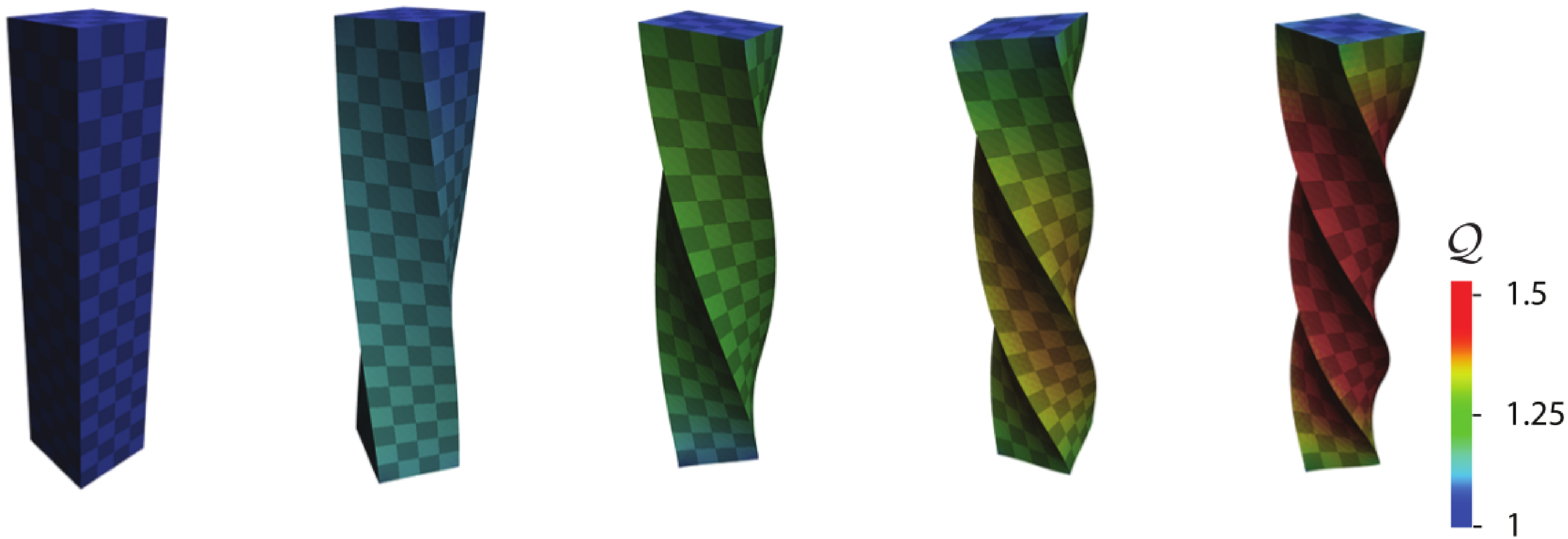
Conformal 3d surface deformations

[Crane et al.] : Spin Transformations of Discrete Surfaces

Conformal 3d surface deformations

[Crane et al.] : Spin Transformations of Discrete Surfaces

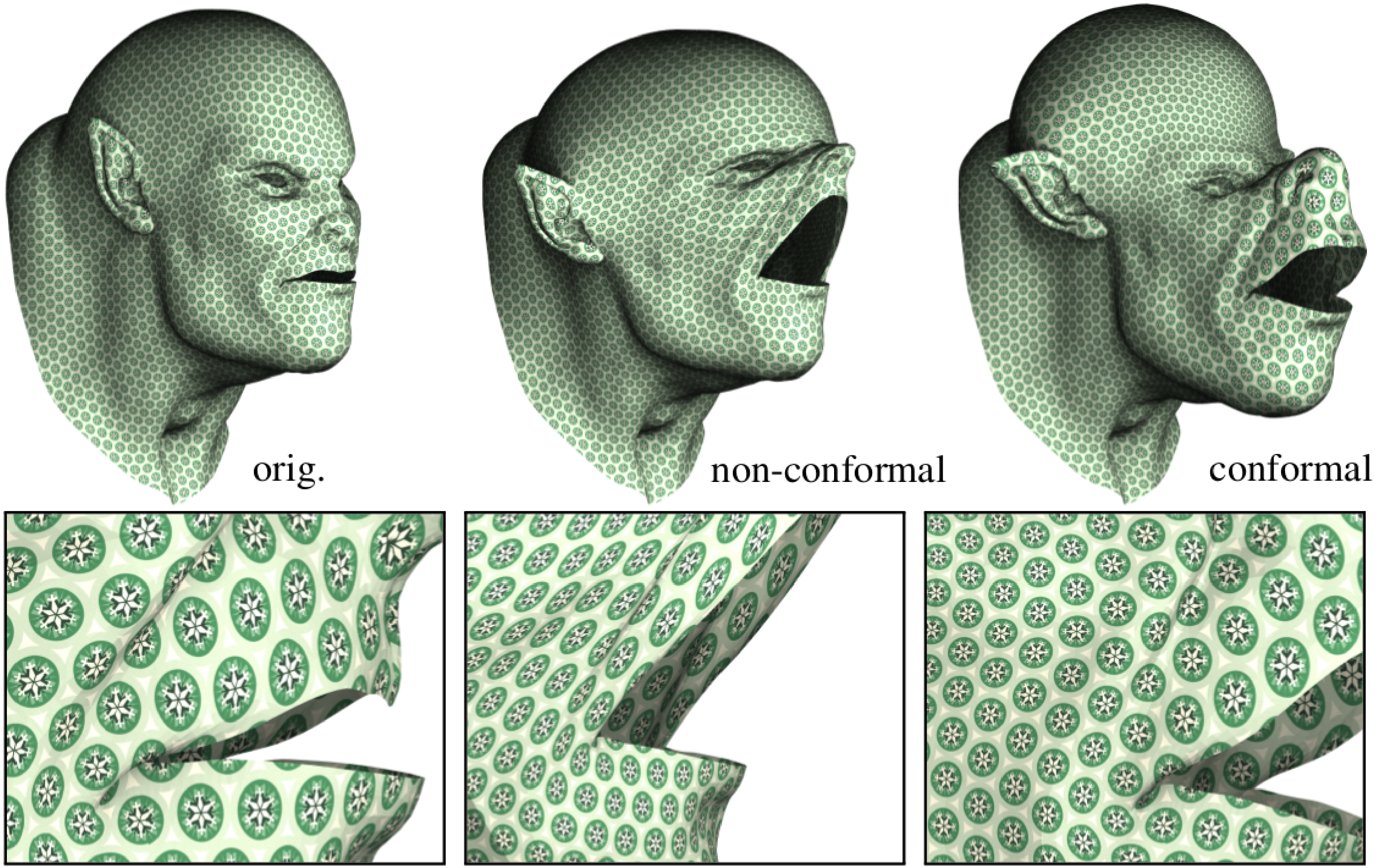
Conformal 3d surface deformations



[Crane et al.] : Spin Transformations of Discrete Surfaces

Conformal 3d surface deformations

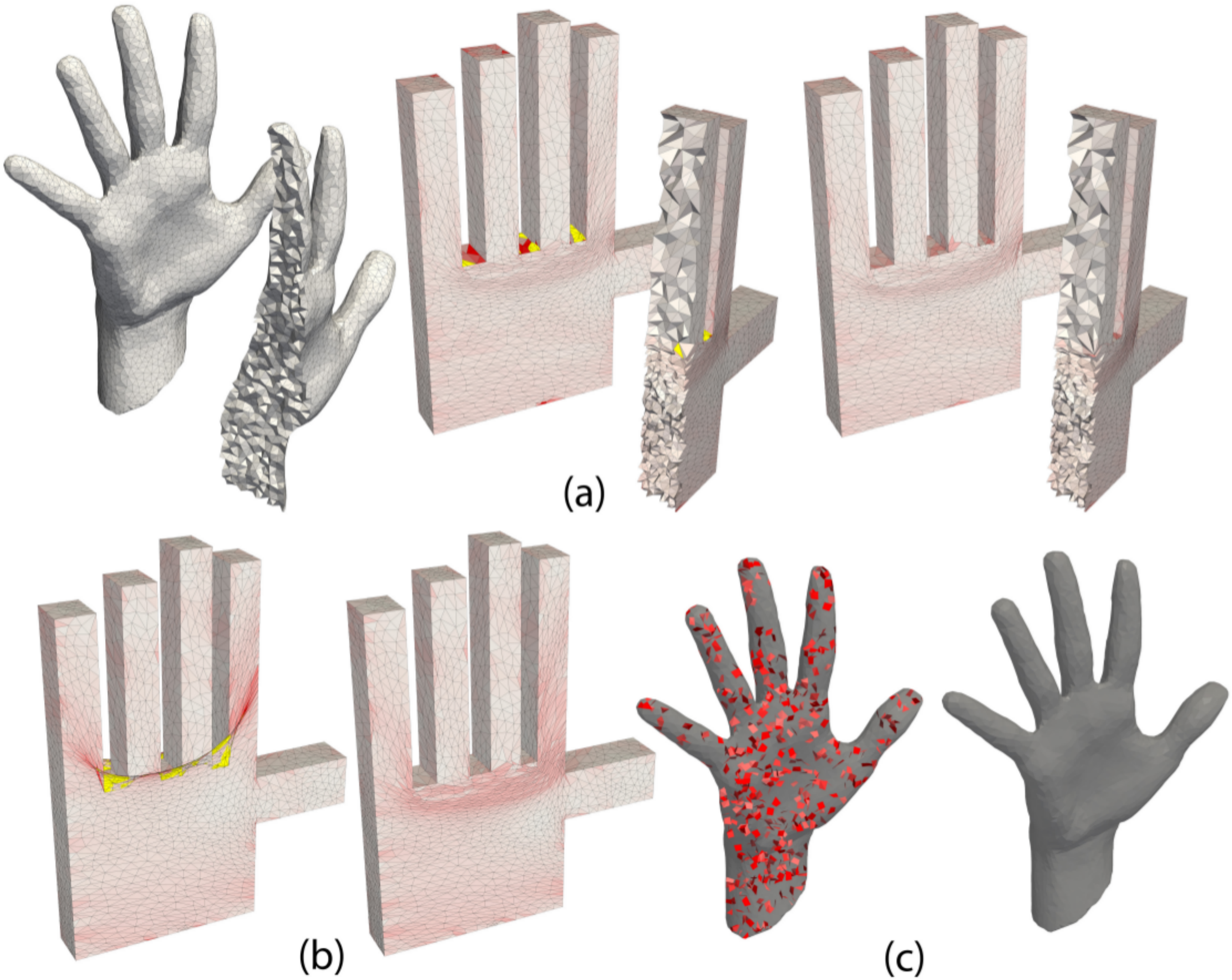
Conformal 3d surface deformations



[Vaxman et al.] : Conformal Mesh Deformations with Möbius Transformations

Bounded 3d volume distortion

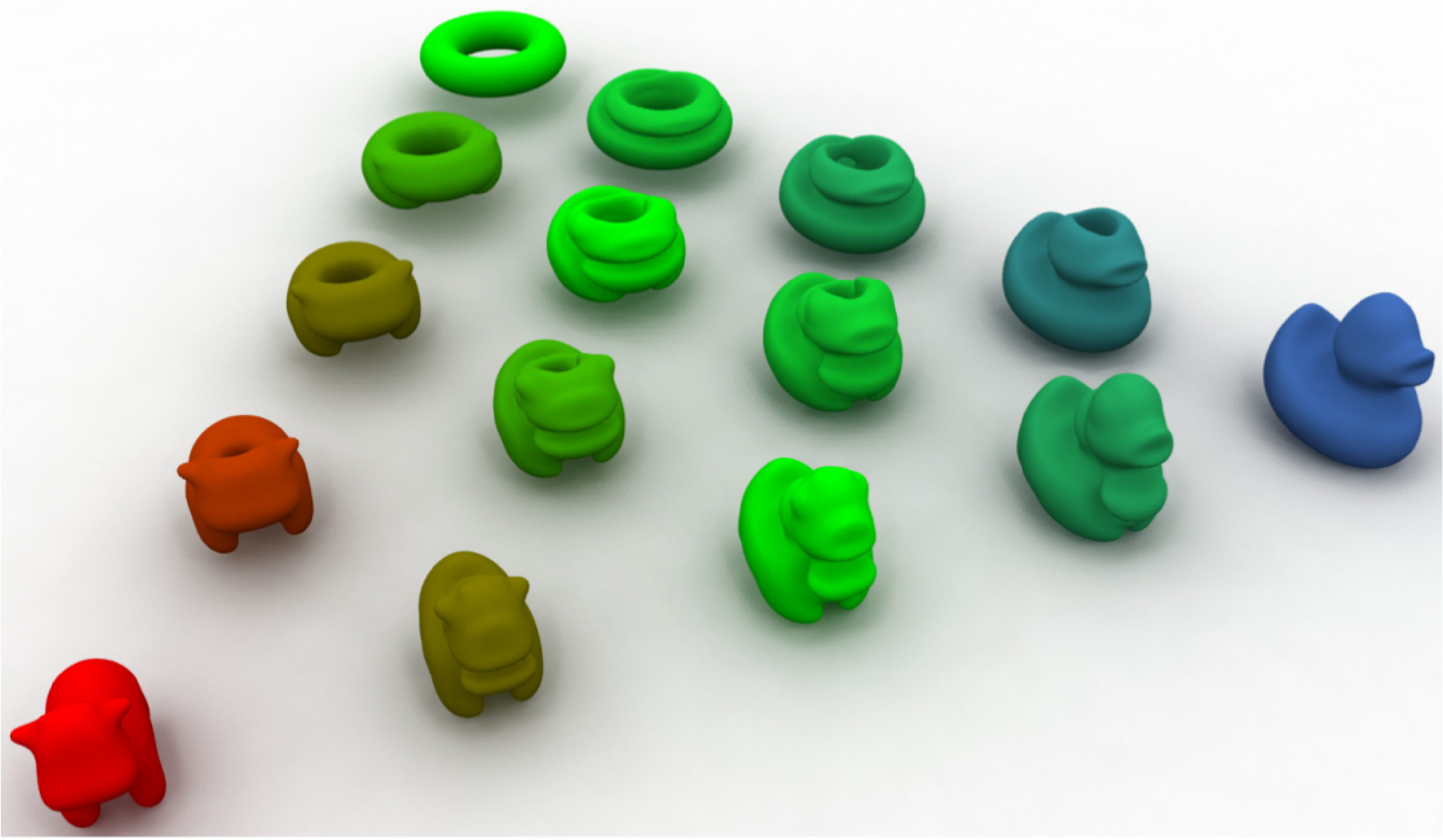
Bounded 3d volume distortion



[Aigerman et al.] :Injective and Bounded Distortion Mappings in 3D

Shape interpolation using optimal transport

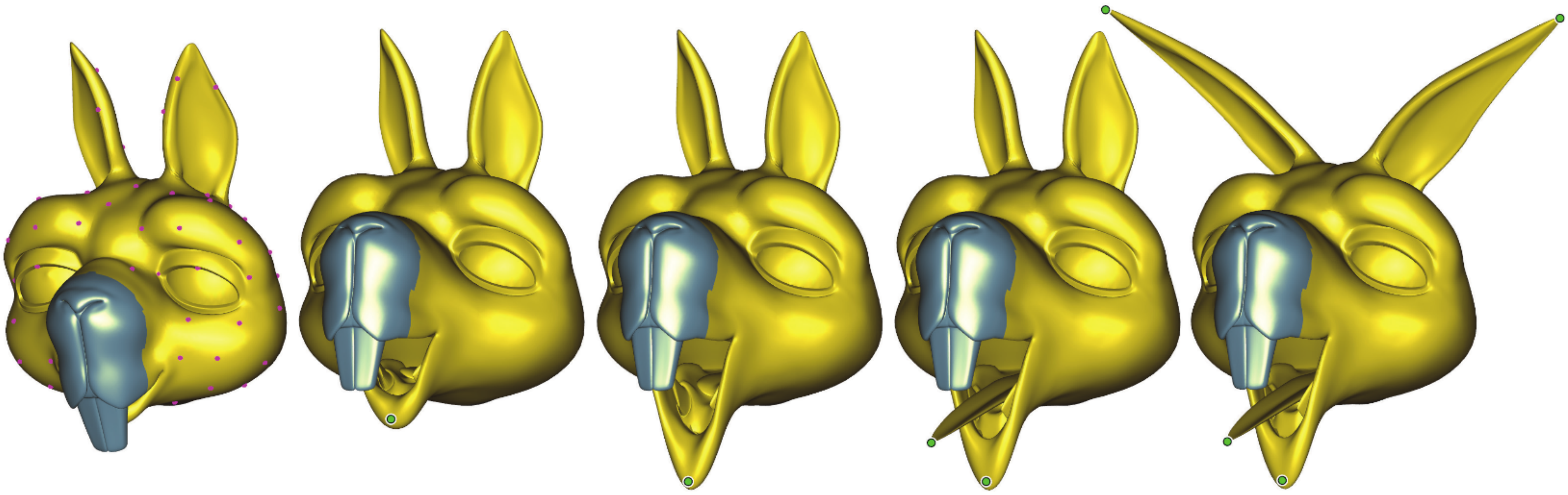
Shape interpolation using optimal transport



[Solomon et al.] :Convolutional Wasserstein Distances: Efficient Optimal Transportation on Geometric Domains

Real-time ARAP

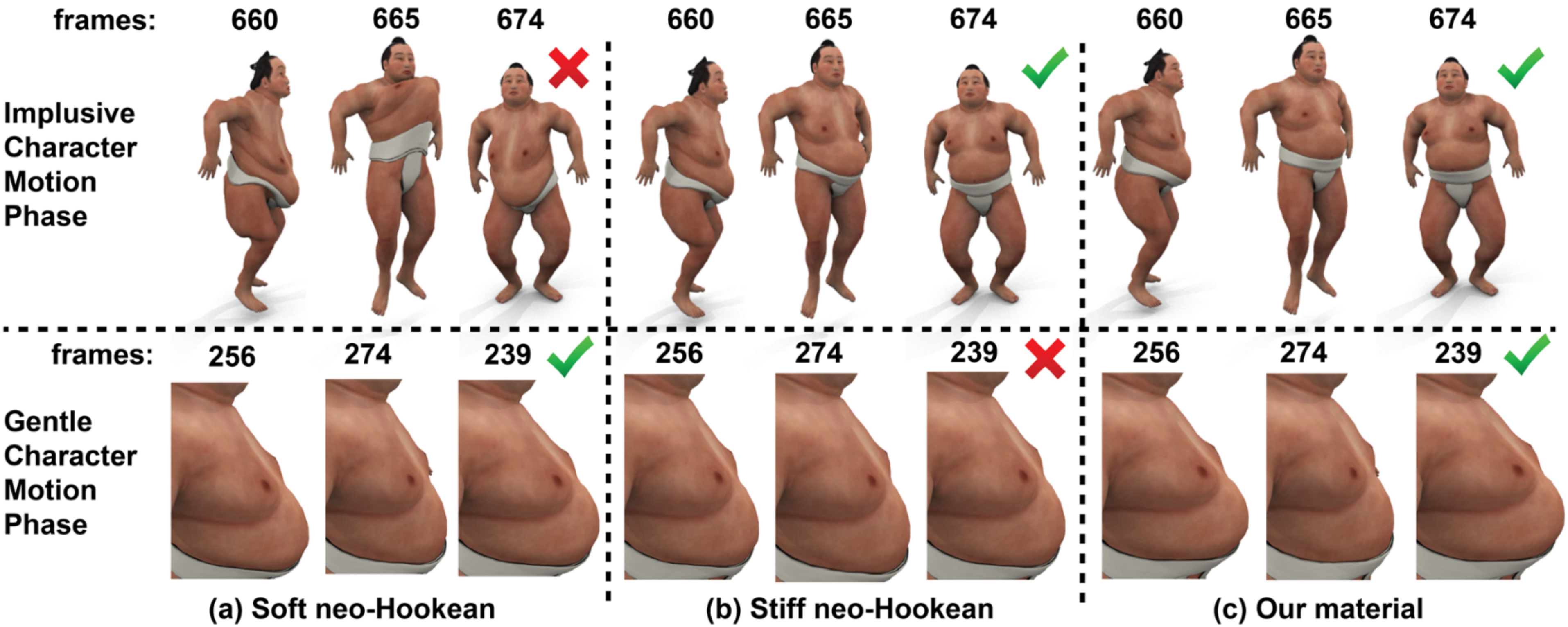
Real-time ARAP



[Wang et al.] :Linear Subspace Design for Real-Time Shape Deformation

Material design

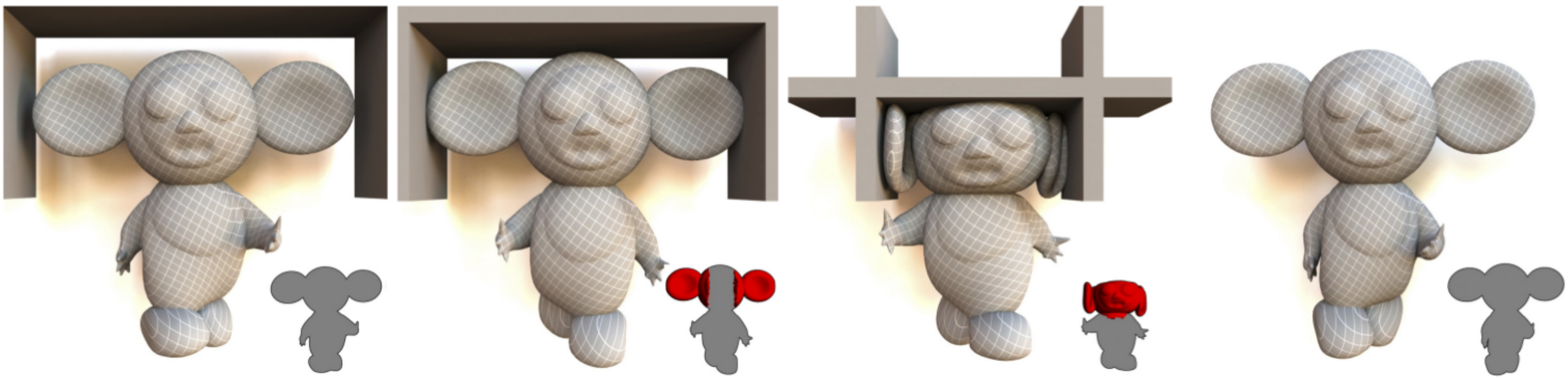
Material design



[Xu et al.] :Nonlinear Material Design Using Principal Stretches

Physically-correct animations

Physically-correct animations



(a) (b) (c) (d)

[Teng et al.] :Subspace Condensation: Full Space Adaptivity for Subspace Deformations

So many more...