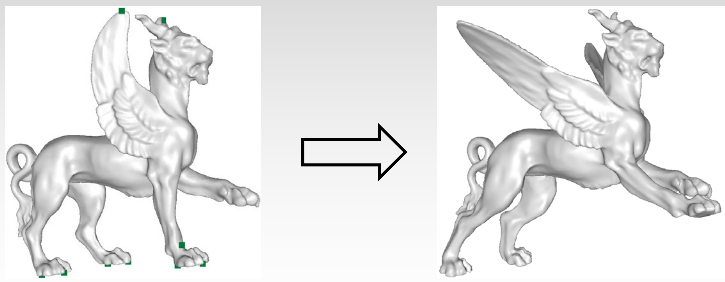
Mesh Editing: Deformation & Other Operations



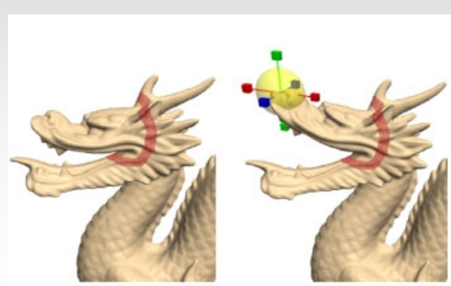
Deformation

Modify global shape

- preserve local features & global continuity

Control mechanism

- anchors (triangles/vertices) – moved by user
- Region of influence (ROI)

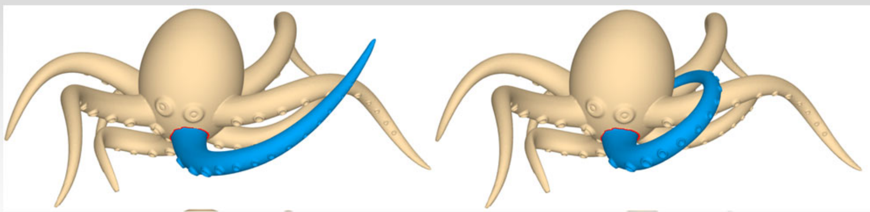


Capturing Geometry - Local Coordinates

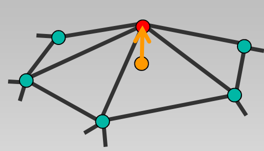
Define local geometry

- Vertex Based
- Triangle Based

Preserve under deformation



Local coordinates - Laplacian



$$\boldsymbol{\delta}_i = \mathbf{v}_i - \frac{1}{d_i}\sum_{j\in N(i)} \mathbf{v}_j$$

$$\boldsymbol{\delta}_i = \sum_{j\in N(i)} \frac{1}{d_i}\left(\mathbf{v}_i - \mathbf{v}_j\right)$$

Can always add weights:

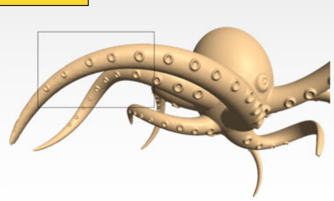
$$\boxed{\delta_i = \frac{\sum_{j\in N(i)} w_{ij}\left(\mathbf{v}_i - \mathbf{v}_j\right)}{\sum_{j\in N(i)} w_{ij}}}$$

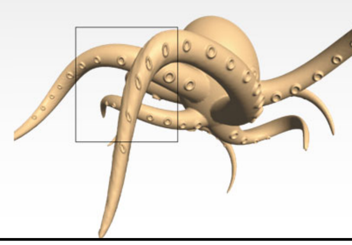
v correlates to δ via

$$\mathbf{L}\ \mathbf{v} = \delta$$

$$\mathrm{L} = \mathrm{I} - \mathrm{D}^{-1}\mathrm{A}$$

$$A_{ij} = \begin{cases} 1 & i \in N(j) \\ 0 & otherwise \end{cases} \qquad D_{ij} = \begin{cases} d_i & i = j \\ 0 & otherwise \end{cases}$$





Surface Reconstruction

Pose new constraints on mesh

- $\mathbf{v}_i = \mathbf{u}_i; i \in c$
- c = set of constraints

Minimize error in reconstructed surface

- In least square fashion

$$E(\mathbf{V}') = \sum_{i=1}^{n} \left\| \delta_i - L(\mathbf{v}'_i) \right\|^2 + \sum_{i \in c} \left\| \mathbf{v}'_i - \mathbf{u}_i \right\|^2$$

Least Square Fitting

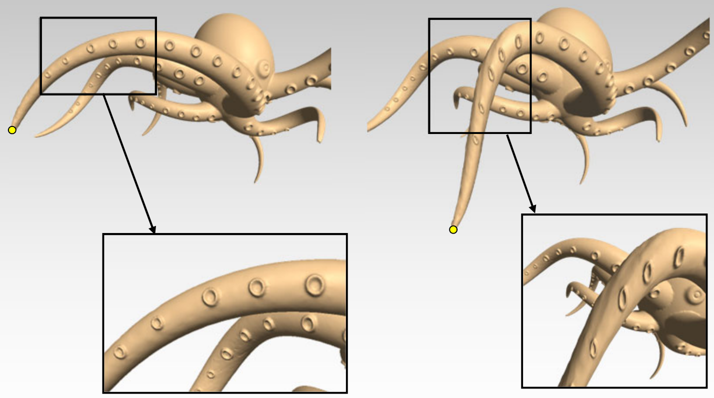
$Ax=b$ with m equations, n unknowns

Least square solution: $x=(A^T A)^{-1} A^T b$

Use your favourite solver to solve $A^T A\, x = A^T b$

Problem

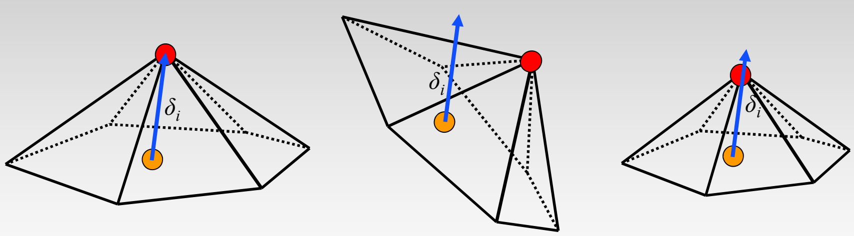
Only translation invariant



Laplacian Coordinates

Translation invariant

Not rotation/scale invariant



$$\delta_i = L(\mathbf{v}_i) = L(\mathbf{v}_i + \mathbf{t}); \forall \mathbf{t} \in \Re^3$$

Solution

Add rotations into framework

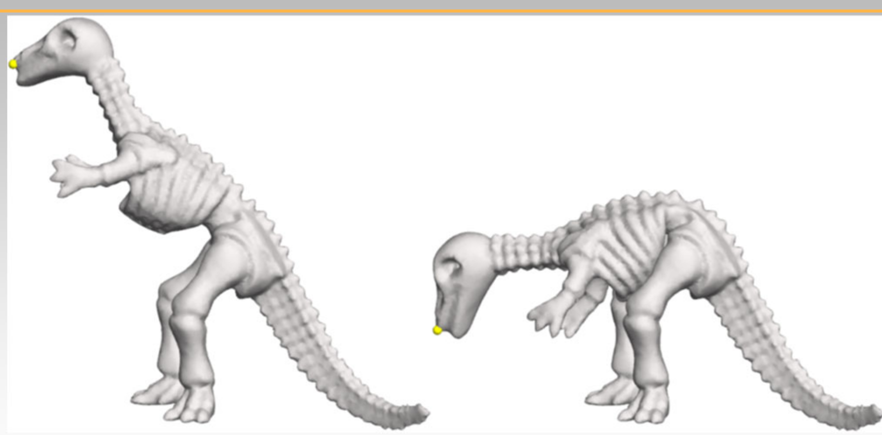
$$E(\mathbf{V}') = \sum_{i=1}^{n} \left\| R_i \delta_i - L(\mathbf{v}'_i) \right\|^2 + \sum_{i \in c} \left\| \mathbf{v}'_i - \mathbf{u}_i \right\|^2$$

- Interleave rotate/position (local/global) iterations

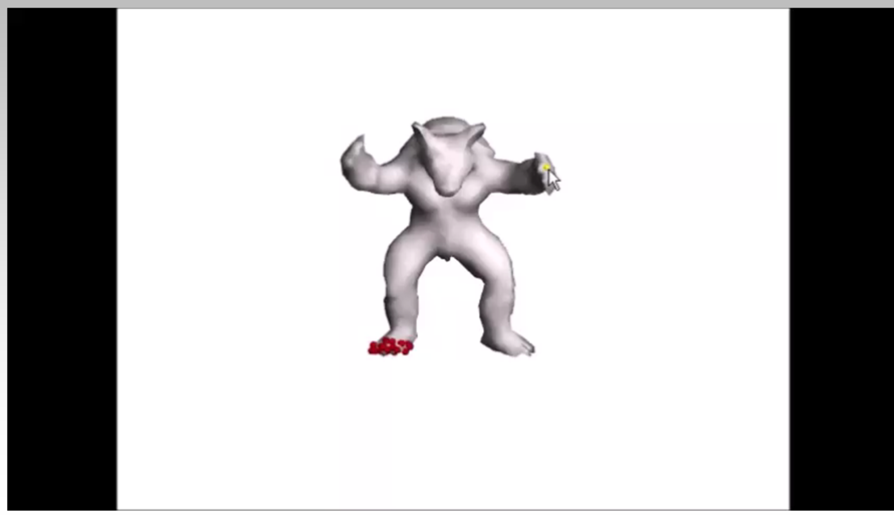
Challenge

- Too local… (consistency between adjacent umbrellas)

As-Rigid-As-Possible Surface Modelling



As-rigid-as-possible (ARAP)



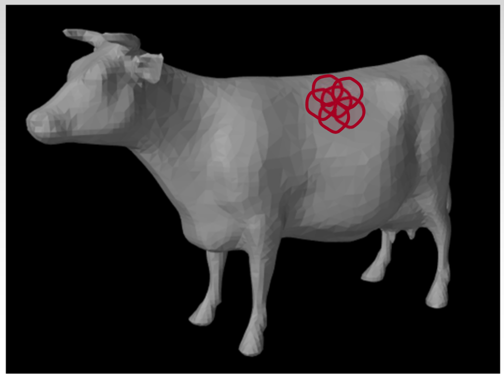
ARAP in a nutshell…

- Decompose surface into small **overlapping** "cells"
- Measure local rigidity => define local rotations
  - Non-linear but small
- Use to solve globally
  - Quadratic

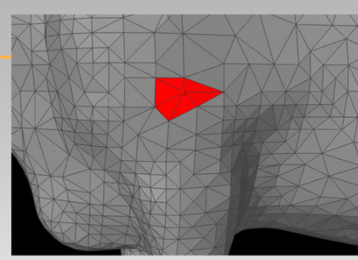
Cell Construction

Desired Properties:

Characterize local shape
Used to enforce local rigidity constraints
Overlapping, to prevent shearing/stretching at cell boundaries



Cell Construction



Simplest logical choice for cells?

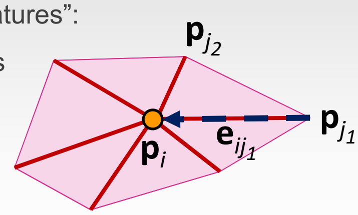
Vertex Umbrella

- Covers entire surface
- One cell per vertex
- All triangles exist in 3 cells

Within cell, define translation-invariant "features":

- Vectors from central vertex to neighbours

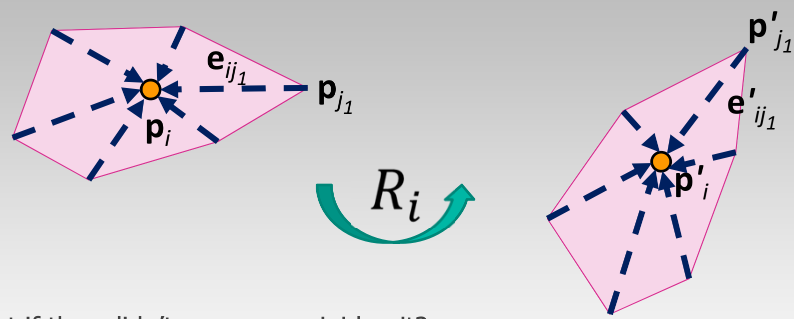
$$\boldsymbol{e}_{ij} = \boldsymbol{p}_i - \boldsymbol{p}_j$$



Local "Rigidity"

If cell i moved as a rigid unit, we could write:

$$\boldsymbol{e}'_{ij} = R_i\, \boldsymbol{e}_{ij} \quad \forall\, j \in N(i)$$

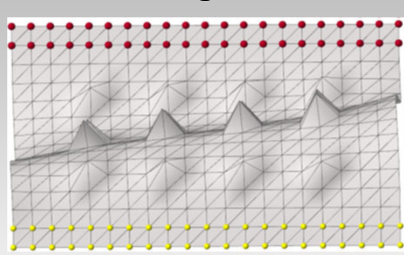


- What if they didn't move as a rigid unit?

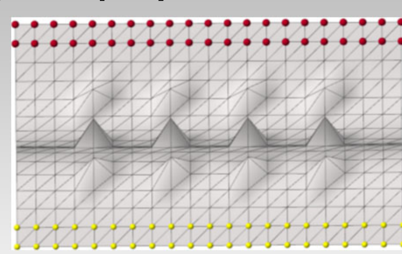
$$\text{Rigid Error}^2\,(C_i, C'_i) = \sum_{j \in N(i)} \left\| \boldsymbol{e}'_{ij} - R_i\, \boldsymbol{e}_{ij} \right\|^2$$

Edge Weights

Should all edge vectors be weighted equally?



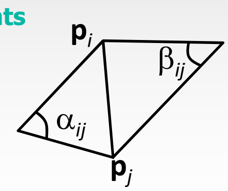
Uniform Weights



Cotangent Weights

Cotangent weights:

$$w_{ij} = \frac{1}{2}\left(\cot \alpha_{ij} + \cot \beta_{ij}\right)$$



Global Rigidity

Local Rigidity: $\text{Rigid Error}^2\,(C_i, C'_i) = \sum_{j \in N(i)} w_{ij} \left\| (\boldsymbol{p}'_i - \boldsymbol{p}'_j) - R_i(\boldsymbol{p}_i - \boldsymbol{p}_j) \right\|^2$

- Given $\{\boldsymbol{p}_i\}$ and $\{\boldsymbol{p}'_i\}$, what are the optimal rigid transforms $\{R_i\}$?
  - Least squares + SVD!!!

Global Rigidity:

All local cells as-rigid-as-possible ➔ global shape is as-rigid-as-possible

Global energy function:

$$\text{Energy} = \sum_i \sum_{j \in N(i)} w_{ij} \left\| (\boldsymbol{p}'_i - \boldsymbol{p}'_j) - R_i(\boldsymbol{p}_i - \boldsymbol{p}_j) \right\|^2$$

© Alla Sheffer

Mesh Deformation

- Positional constraints: $\boldsymbol{p}'_i = \boldsymbol{u}_i, \; i \in C$
- Determine locations $\{\boldsymbol{p}'_i\}$ for all points by minimizing global energy

$$\text{Energy} = \sum_i \sum_{j \in N(i)} w_{ij} \left\| (\boldsymbol{p}'_i - \boldsymbol{p}'_j) - R_i(\boldsymbol{p}_i - \boldsymbol{p}_j) \right\|^2 + \underbrace{\mu \sum_{i \in C} w_{ij} \left\| \boldsymbol{p}'_i - \boldsymbol{u}_i \right\|^2}_{\text{soft constraints}}$$

Caveats:

- $\{\boldsymbol{p}'_i\}$ and $\{R_i\}$ are unknown
- Non-linear optimization problem

© Alla Sheffer

Mesh Deformation

Solution:

- Start with initial guess of $\{\boldsymbol{p'}_i\}$, solve for $\{R_i\}$
  - *Compute for each cell independently (L.S. + SVD)*
  - *Embarrassingly parallel*
- Given $\{R_i\}$, minimize energy to find $\{\boldsymbol{p'}_i\}$

$$\sum_{j\in N(i)} w_{ij}\left(\boldsymbol{p'}_i - \boldsymbol{p'}_j\right) = \sum_{j\in N(i)} \frac{w_{ij}}{2}(R_i + R_j)(\boldsymbol{p}_i - \boldsymbol{p}_j)$$

$$\boxed{L\boldsymbol{p}' = \boldsymbol{b}}$$

Advantages

"L" is only a function of the cotangent weights

- Depends only on original mesh
- Only needs to be factored ONCE!!
- Sparse linear system

Rotations can be computed in parallel

FAST!!

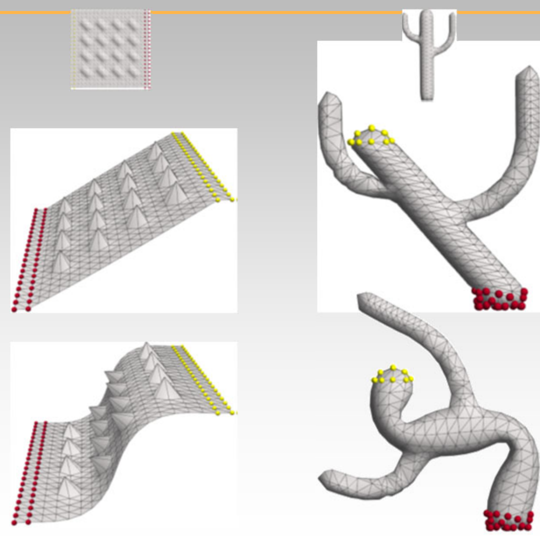
- Each iteration reduces energy
  - Updating rotations guaranteed to reduce cell-error
  - Updating positions guaranteed to reduce global error

Gauranteed Convergence!!

Results (vs Poisson)

Poisson:

ARAP:



ARAP summary

Method tries to keep "cells" as-rigid-as-possible

Requires:

- Estimating rotation per umbrella
- Minimizing global energy iteratively

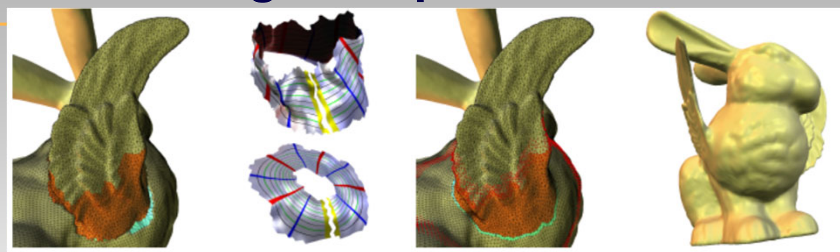
Advantages:

- Fast
- Guaranteed convergence (to something...)
- (Almost) Edge-length preserving
- Easy to implement

Disadvantages

- Non-linear optimization
- Depends on mesh resolution
- Not volume-preserving

More Model Editing: Composition

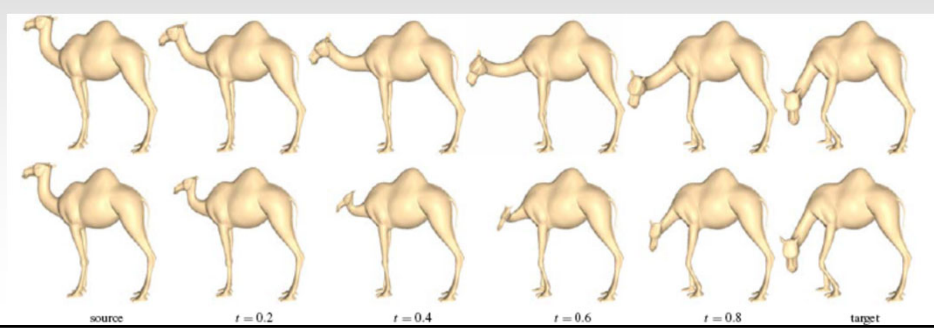


Specify parts to glue – cut boundaries
Align/specify correspondence
Create common connectivity
Define smooth geometry transition

More Model Editing: Morphing

Require common connectivity & feature correspondence
Vertex trajectories

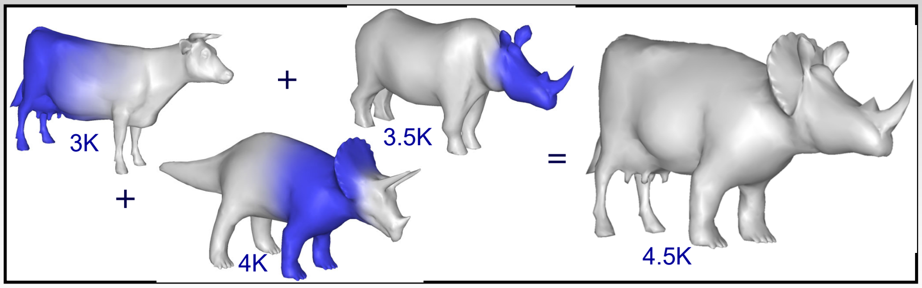
- preserve shape
- avoid (as much as possible) self-intersections



More Model Editing: Blending

Special case of morphing

- Single frame with non-uniform (smooth) time parameter



Numerical Issues

Minimization (Unconstrained)

To find x that minimizes $F(x)$ – find x such that $F'(x)=0$

- Check if got minimum/maximum/saddle point
- Note: finds **LOCAL** minimum

Typically no need for explicit check (assume function does not have maxima/saddles

Translate problem into: find x such that $f(x)=0$

Minimization with Constraints

Need to

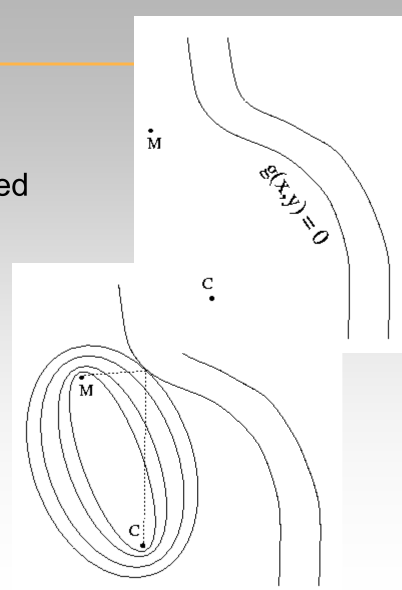
- Find x such that $F(x)$ minimal
- WHEN constraints $c(x) = 0$ satisfied

Achieved when

- $F'(x)=\mu c'(x)$
- for unknown μ

General formulation

- $F^*(x,\mu)=F(x)+\mu c(x)$
- Find x,μ which extremize F^*
- Known as min-max
  - *min on x*
  - *max on μ*



Solution

Use Lagrange Multipliers

$$F^*(x,\mu)=F(x)+\mu c(x)$$

Solve the min-max problem (minimum on x, maximum on μ)

Reached when all derivatives are zero

Have linear (or non-linear) system of equations

- If non linear - use Newton method to solve

Solving Linear System

Solve $Ax=B$ (A $n \times n$ matrix)

Choice I: Compute A^{-1} $O(n^3)$ TERRIBLY expensive

Choice II: Iterative (Gauss/Gauss-Seidel)

- Set x to initial guess
- Solve one equation at a time
  - *$A_i x=B_i$ - consider all x_j ($j \neq i$) as constant and compute x_i*
    - $x_i = (b_i - \Sigma a_j x_j)/a_i$
  - *Repeat (for all i) till convergence*
- Works only for a very small set of matrices

Solving Linear System

Choice III: LU (or LDL^T) decomposition

- Compute matrices L & U such that
  - $LU=A$
  - L – *lower matrix (has 1's on diagonal & 0's above)*
  - U – *upper matrix (has 0's below diagonal*
  - *Use off-the-shelf algorithm/code*
    - Take advantage of sparsity (if applicable)
- Solve:
  - *Solve* $Ly=B$ *(use Gauss iterations)*
    - Works (at each point add ONE variable)
  - *Solve* $Ux=y$ *(use Gauss iterations)*
    - Start from $i=n-1$ and go "up"
    - Works (at each point add ONE variable)