CENG 789 – Digital Geometry Processing

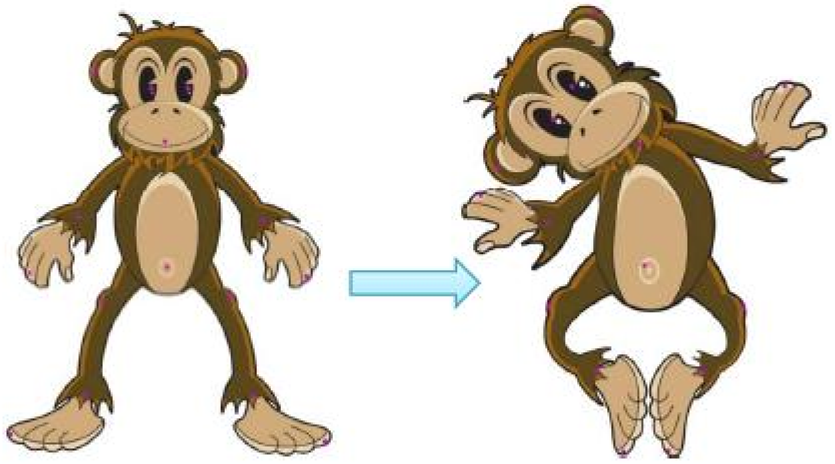
11- Mesh Deformation

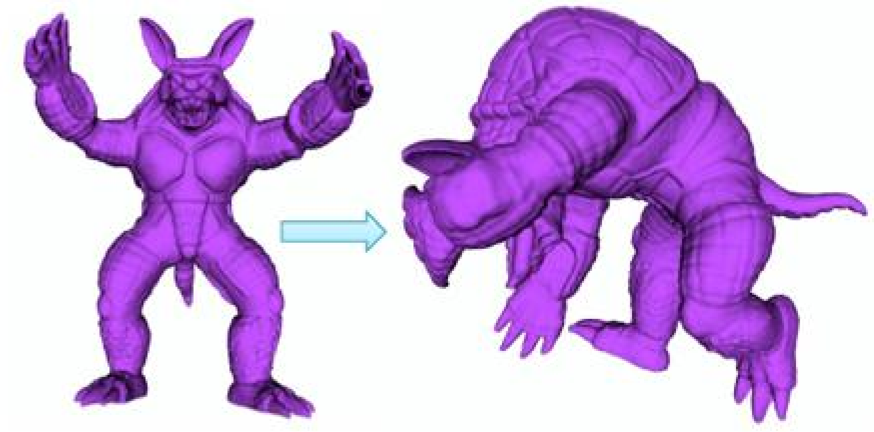
Prof. Dr. Yusuf Sahillioğlu

Computer Eng. Dept, MIDDLE EAST TECHNICAL UNIVERSITY, Turkey

Goal

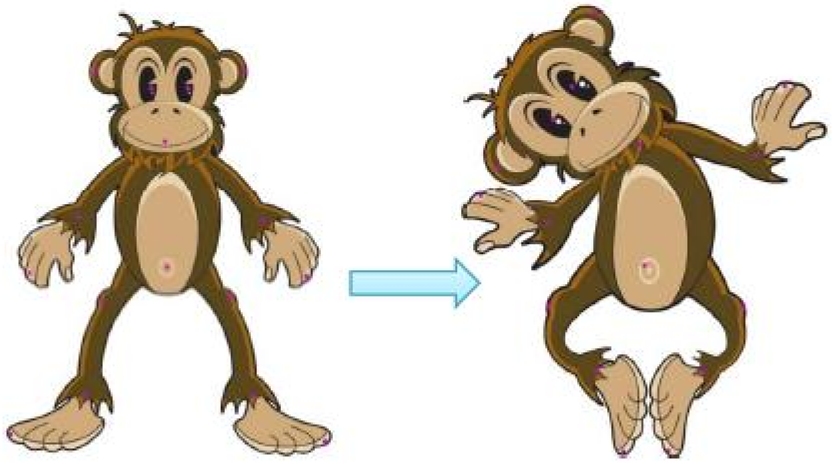
✓ Find the new vertex coordinates for a plausible manipulation/motion.





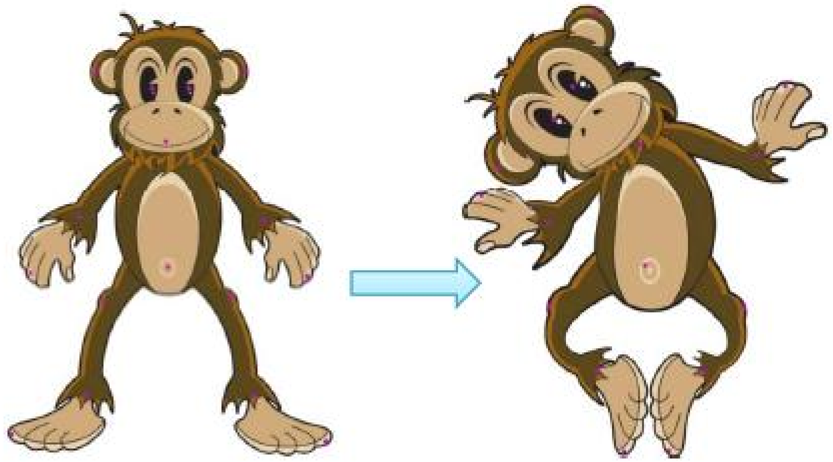
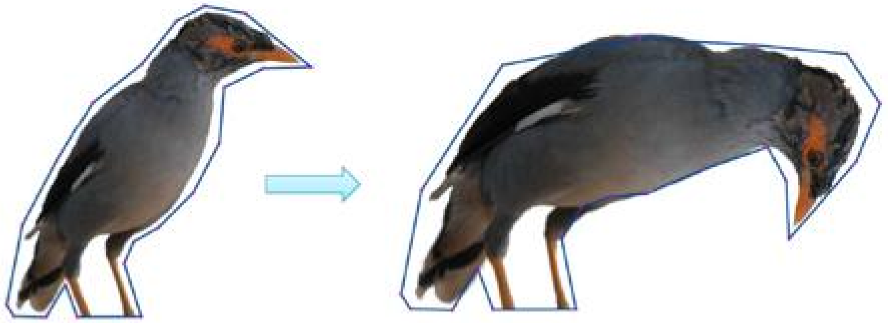
Goal

✓ Complex math hidden behind an intuitive user interface.



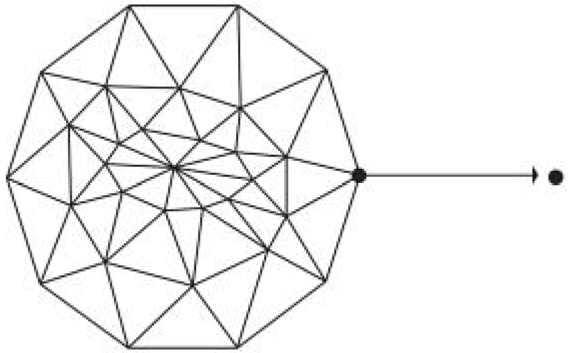
Motivation

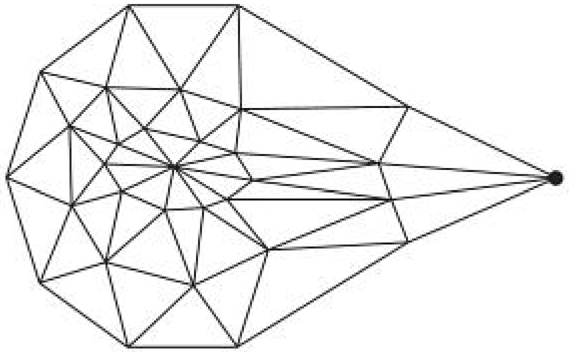
- ✓ Provides easy modeling; i.e., generate new shapes by deforming existing ones.
  - ✓ Create a basketball that deforms when bounced.
  - ✓ Deform Toy Story characters as they walk, talk, been hit, etc.

Deformation Types

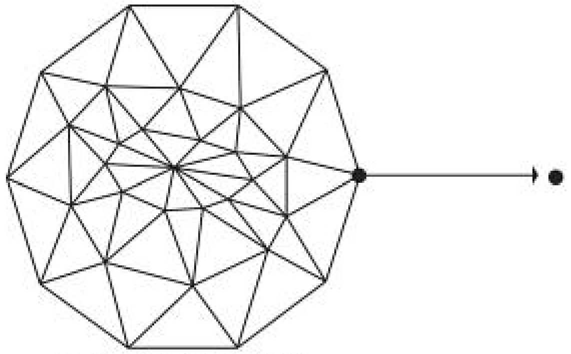
- ✓ Vertex-by-vertex displacement is easy but tedious ☹.
- ✓ Group a number of vertices and displace them uniformly is easy but too restrictive ☹.
- ✓ Allow user to displace 1+ seed/handle vertex and propagate the displacement to close vertices while attenuating the amount of displacement ☺.

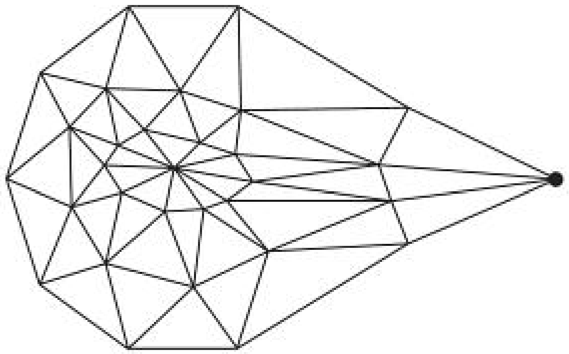

Displacement of seed vertex


Attenuated displacement propagated to adjacent vertices

Deformation Types

- ✓ Attenuation functions:
  - ✓ Trade off quality vs. computational complexity.
  - ✓ Euclidean distance.
  - ✓ Min # of edges connecting seed to the vertex to be displaced.
  - ✓ Geodesic distance.

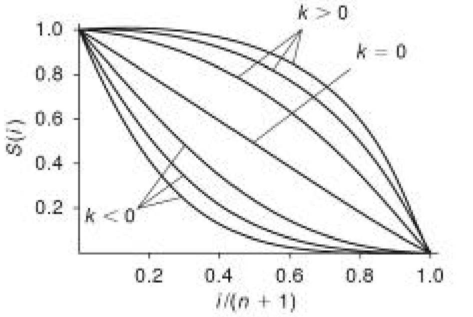

Displacement of seed vertex

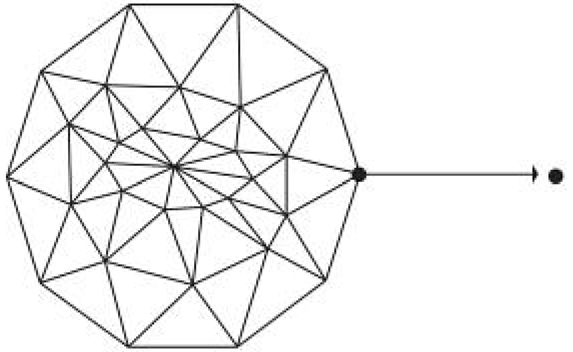

Attenuated displacement propagated to adjacent vertices

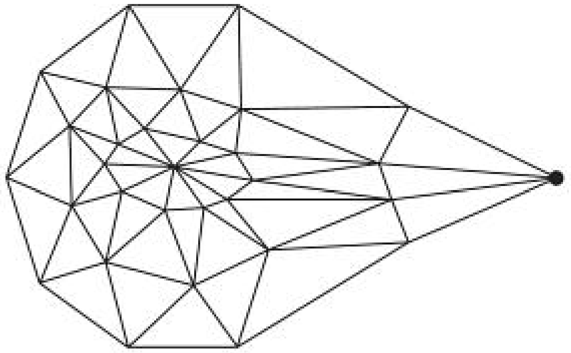
Deformation Types

- ✓ Attenuation functions:
  - ✓ Min # of edges connecting seed to the vertex to be displaced.
  - ✓ User specifies max range of effect to be verts within n edges of seed.
  - ✓ k=0 makes linear attenuation.
  - ✓ k<0 elastic, k>0 rigid displacements.

$$
\begin{aligned}
S(i) &= 1 - \left(\frac{i}{n+1}\right)^{k+1} && k \geq 0 \\
&= \left(1 - \left(\frac{i}{n+1}\right)\right)^{-k+1} && k < 0
\end{aligned}
$$

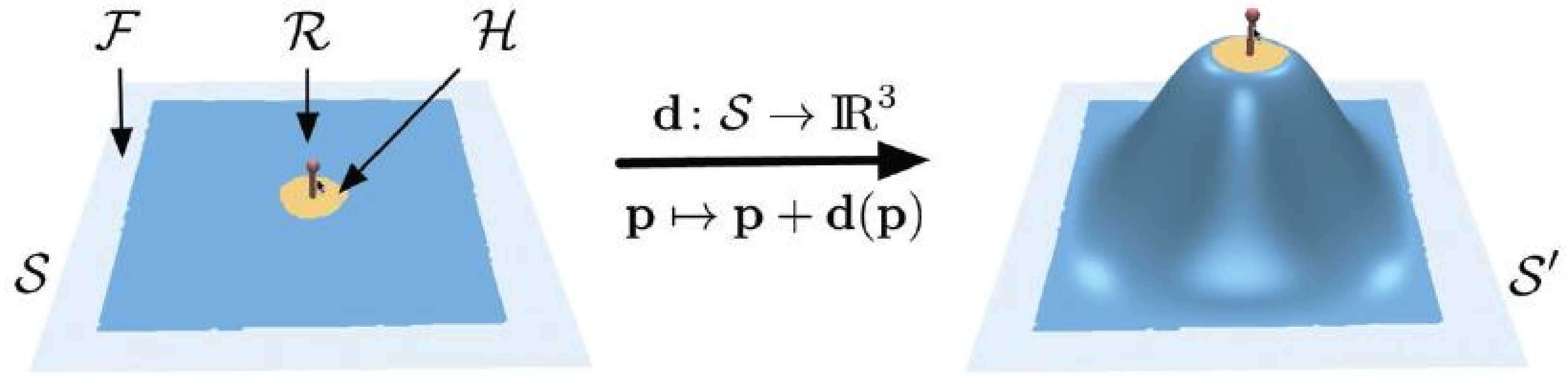



Displacement of seed vertex


Attenuated displacement propagated to adjacent vertices

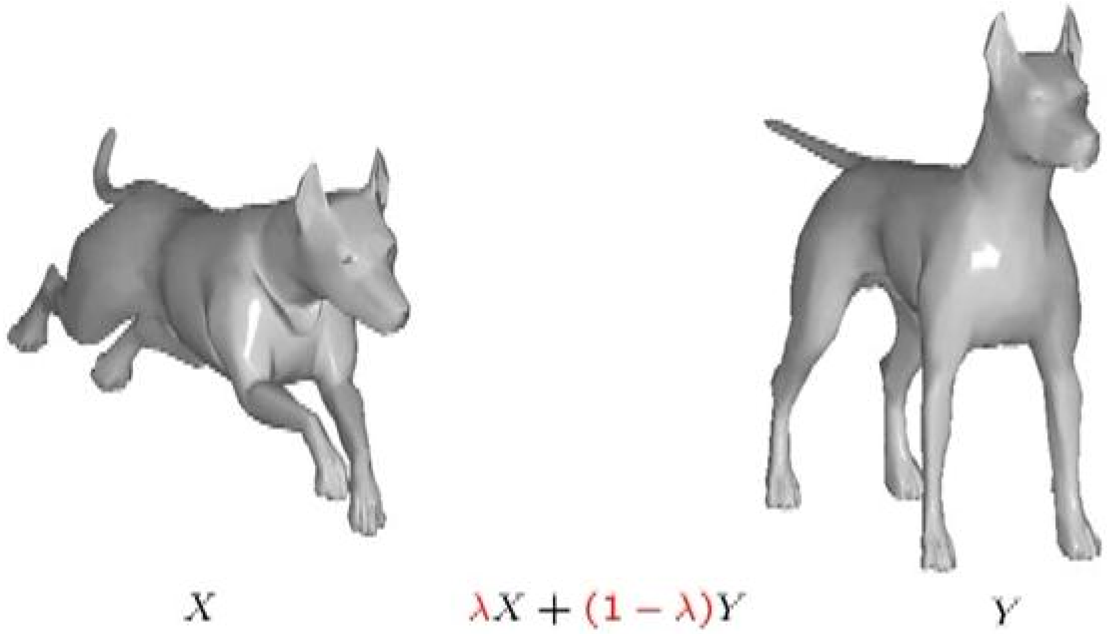
Deformation Types

✓ User controls the deformation by moving a handle region H and keeping the region F fixed. The unconstrained deformation region R should deform in an intuitive manner.


$\mathcal{F}$
$\mathcal{R}$
$\mathcal{H}$
$\mathcal{S}$
$\mathbf{d}: \mathcal{S} \to \mathbb{R}^3$
$\mathbf{p} \mapsto \mathbf{p} + \mathbf{d}(\mathbf{p})$
$\mathcal{S}'$

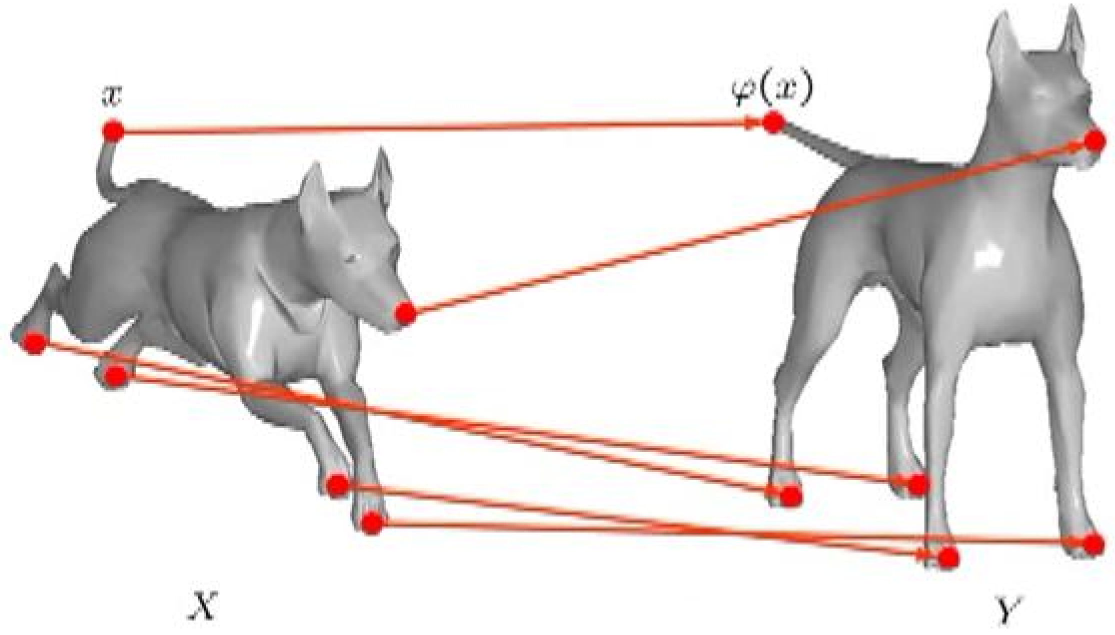
Deformation Types

✓ Creating the in-betweens in shape interpolation is also a deformation.



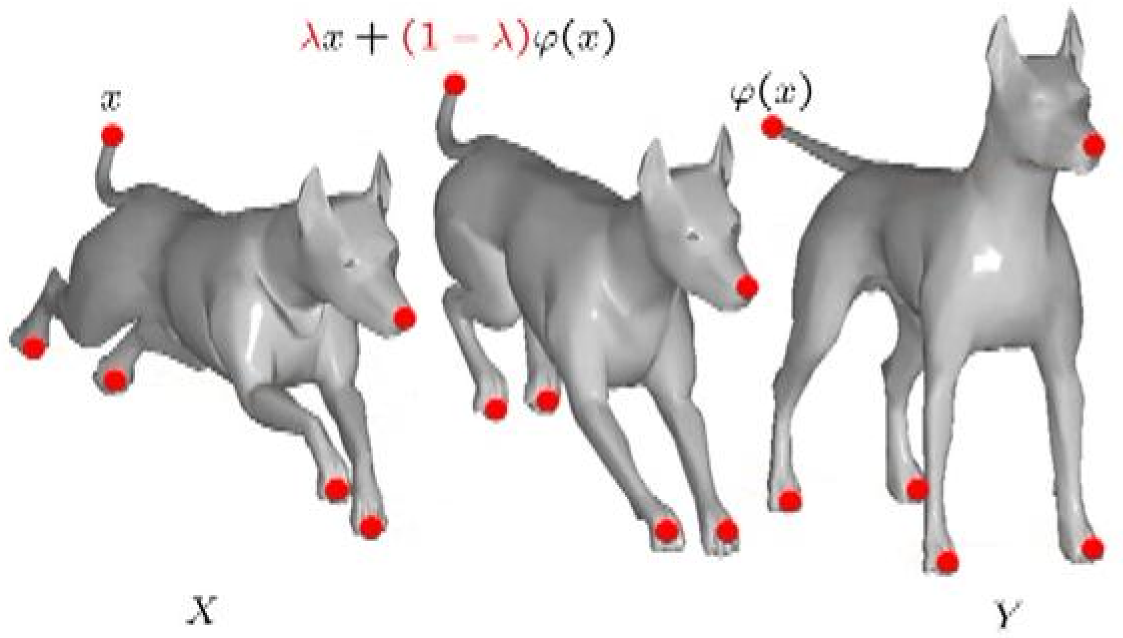
Deformation Types

✓ Creating the in-betweens in shape interpolation is also a deformation.



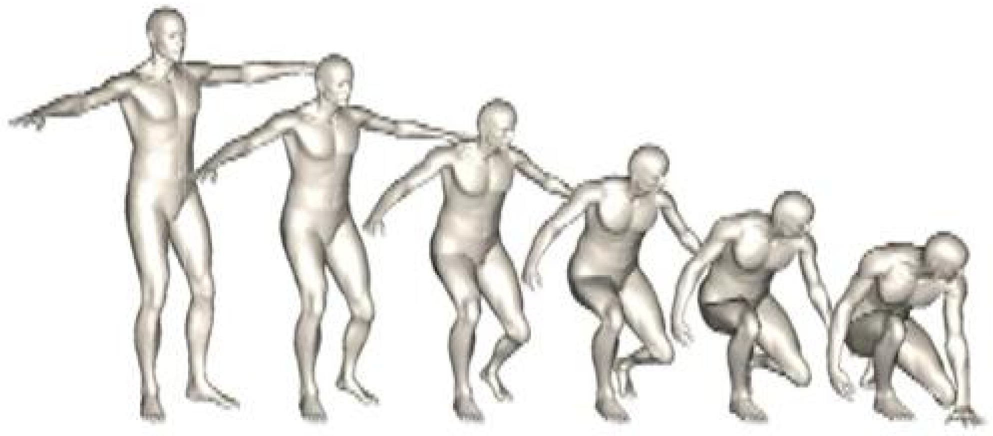
Deformation Types

✓ Creating the in-betweens in shape interpolation is also a deformation.



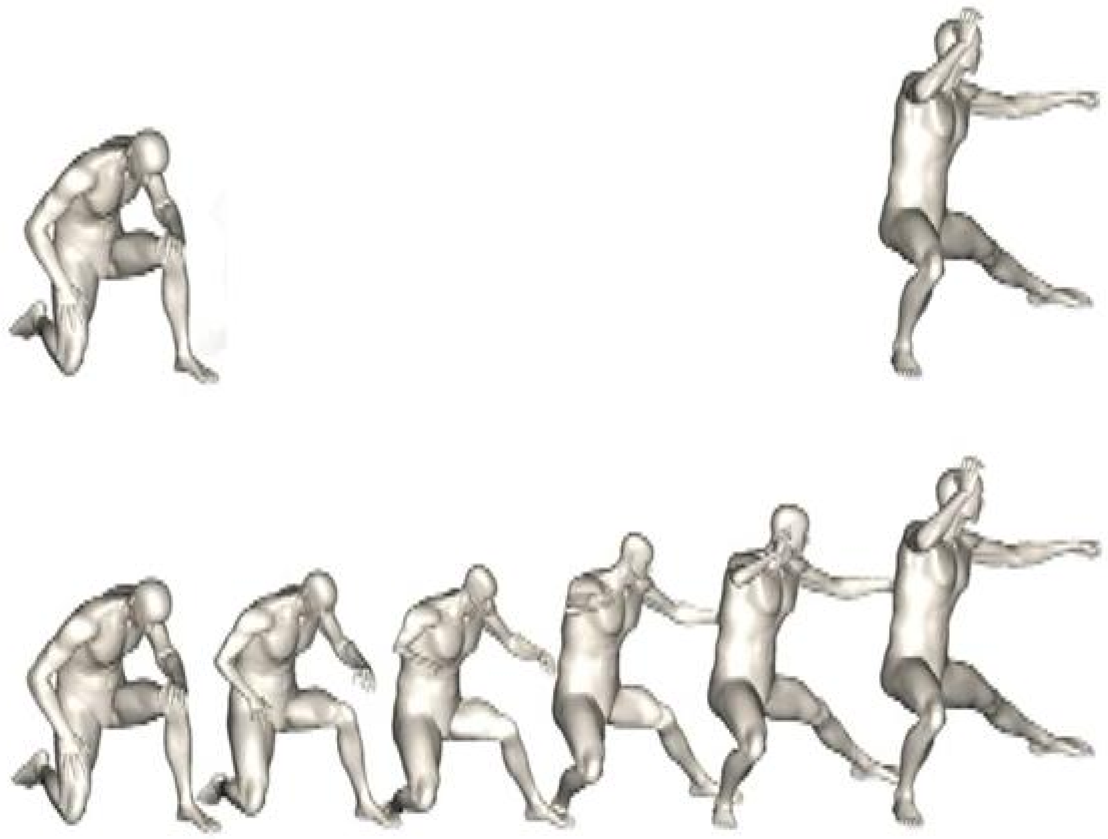
Deformation Types

- ✓ Creating the in-betweens in shape interpolation is also a deformation.
- ✓ No problem:



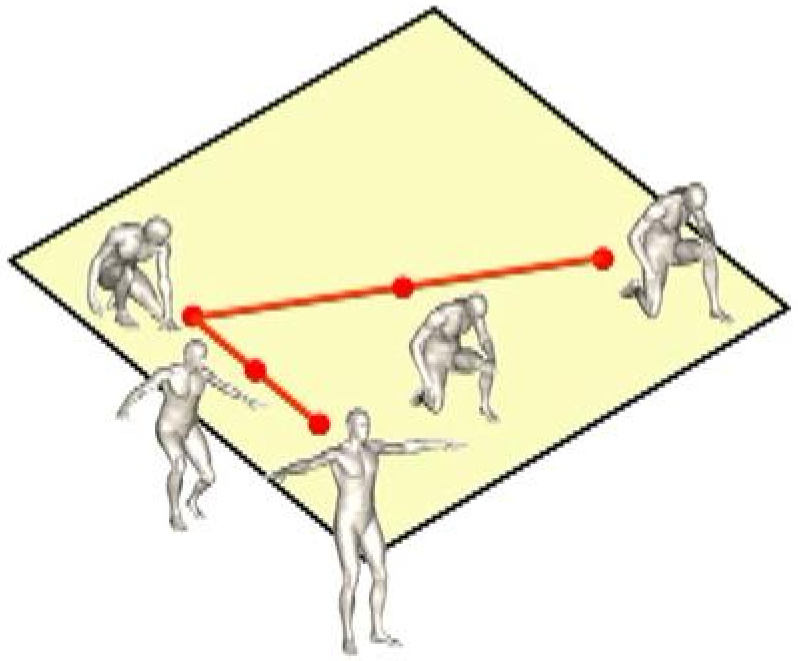
Deformation Types

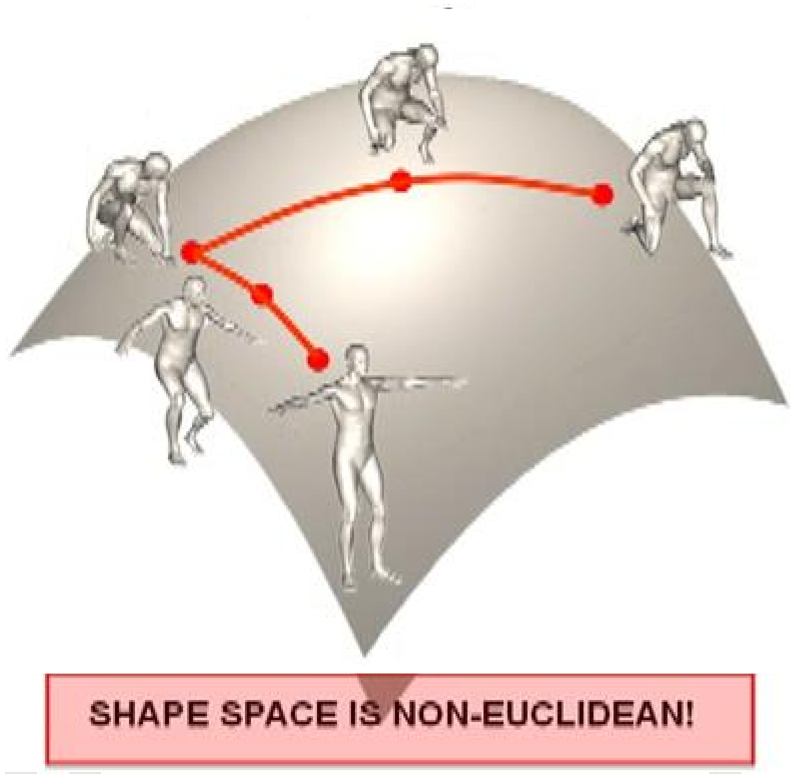
- ✓ Creating the in-betweens in shape interpolation is also a deformation.
- ✓ Problem at hands:



Deformation Types

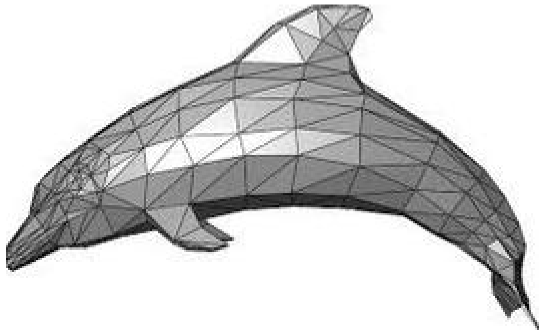
- ✓ Creating the in-betweens in shape interpolation is also a deformation.
- ✓ Reason: shape space is not linear, e.g., distance b/w 2 shapes nonliner





Deformation Types

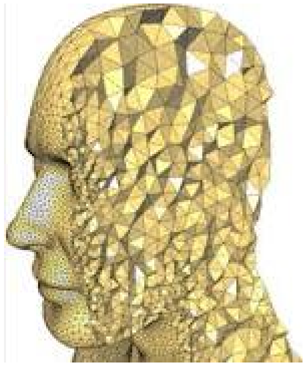
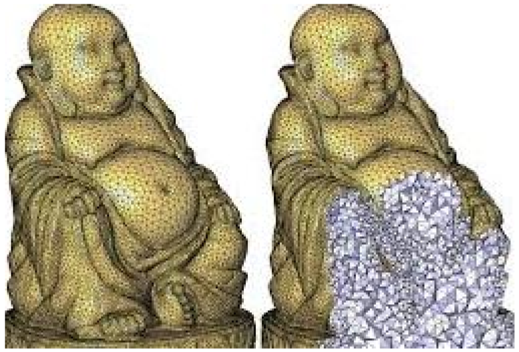
- ✓ Shell-based deformation: polygon meshes.



- ✓ Volume-based deformation: tetrahedral meshes.
- ✓ Multi-scale deformation: efficiency
- ✓ Free-form deformation: lattice-based and cage-based.

Volume-based Deformation

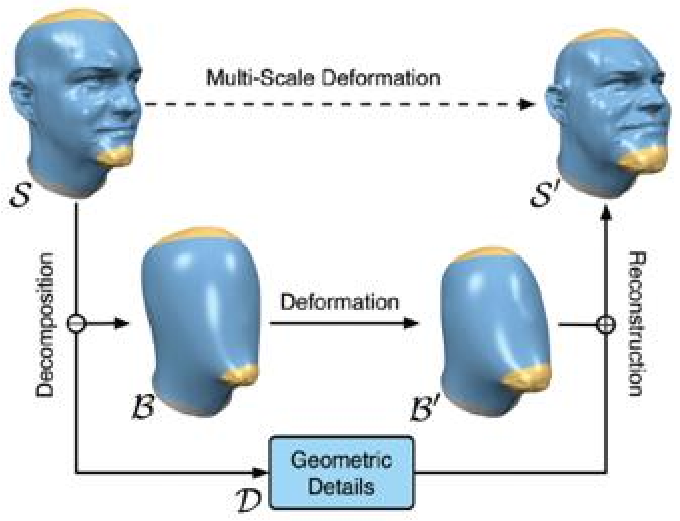
- ✓ Shell-based deformation: polygon meshes.
- ✓ Volume-based deformation: tetrahedral meshes.

- ✓ Multi-scale deformation: efficiency.
- ✓ Free-form deformation: lattice-based and cage-based.

Multi-Scale Deformation

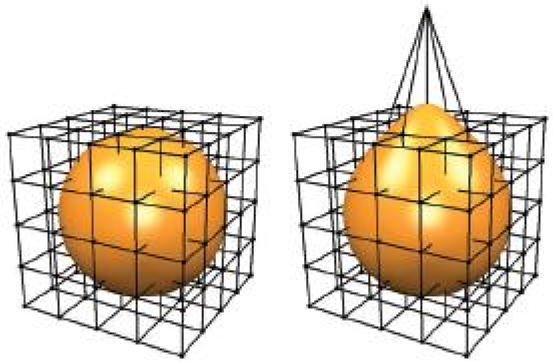
- ✓ Shell-based deformation: polygon meshes.
- ✓ Volume-based deformation: tetrahedral meshes.
- ✓ Multi-scale deformation: efficiency

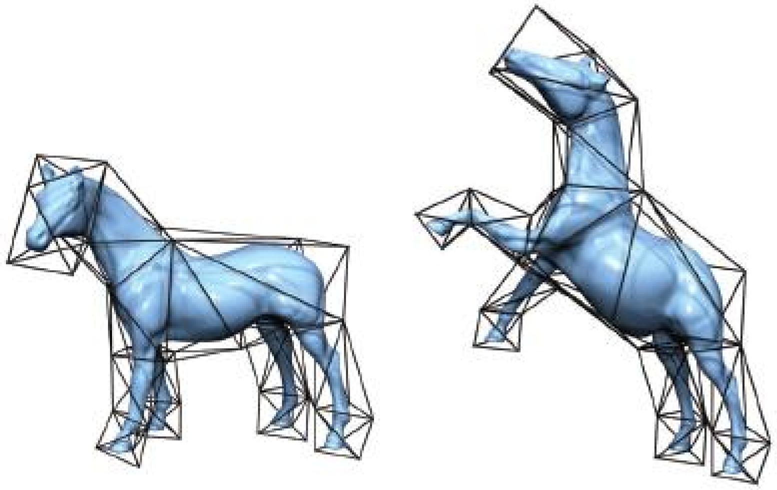


- ✓ Free-form deformation: lattice-based and cage-based.

Free-Form Deformation (FFD)

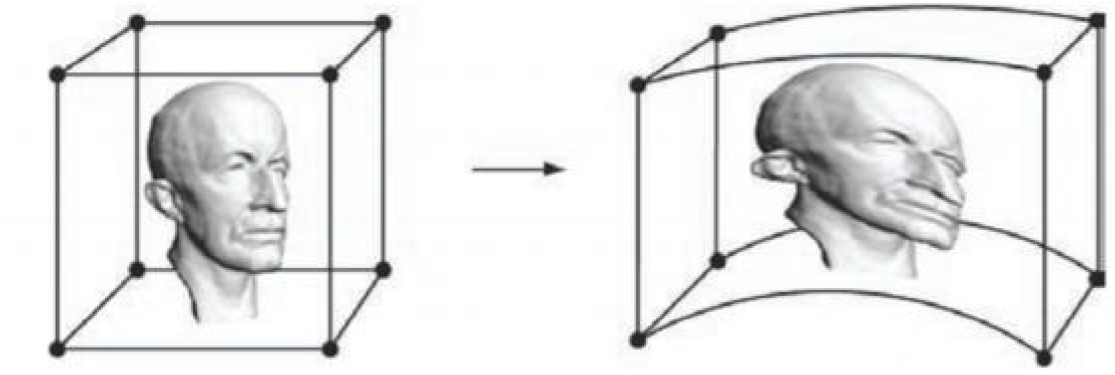
- ✓ Shell-based deformation: polygon meshes.
- ✓ Volume-based deformation: tetrahedral meshes.
- ✓ Multi-scale deformation: efficiency
- ✓ Free-form deformation: lattice-based or cage-based.





FFD

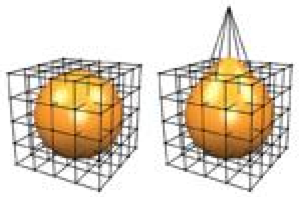
- ✓ Free-form deformation: lattice-based or cage-based.

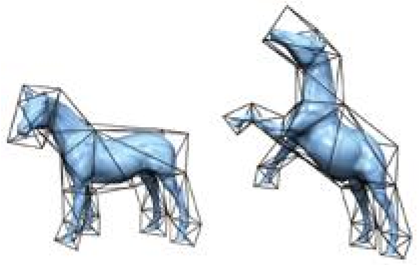


- ✓ Deform the ambient space (grid/cage), and thus implicitly deform the embedded objects.
- ✓ For each vertex of the object, coordinates relative to local grid are determined that register the vertex to the grid.
- ✓ Low-reso grid is then manipulated by the user.
- ✓ Using its relative coords, each vertex is mapped back to modified grid, which relocates the vertex in global space.
- ✓ Parallel execution is possible since each cell is independent.

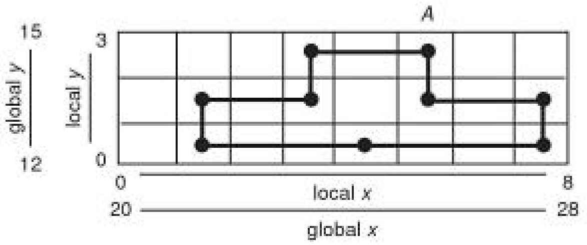
FFD

- ✓ Free-form deformation: lattice-based or cage-based.





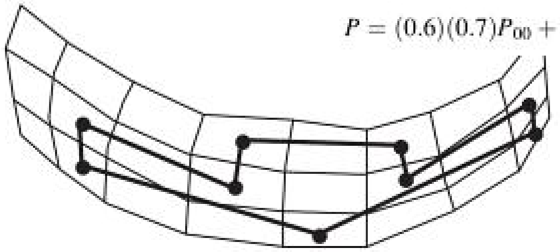
- ✓ Vertex A has global coordinates of (25.6, 14.7), local coordinates of (5.6, 2.7).

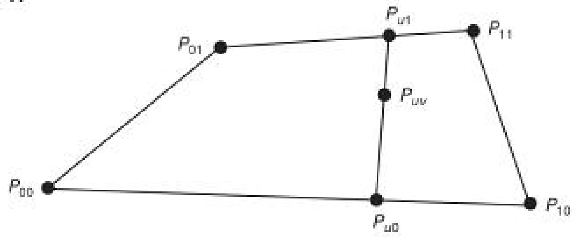


- ✓ Vertex A (and others) are relocated in the modified grid by bilinear interpolation relative to the containing cell of the grid: interpolants u&v=.6 & .7

$$P_{u0} = (1 - u)P_{00} + uP_{10}$$
$$P_{u1} = (1 - u)P_{01} + uP_{11}$$
$$P_{uv} = (1 - v)P_{u0} + vP_{u1}$$
$$= (1 - u)(1 - v)P_{00} + (1 - u)vP_{01} + u(1 - v)P_{10} + uvP_{11}$$

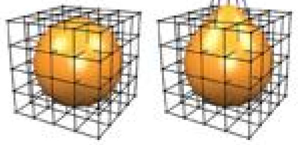
$$P = (0.6)(0.7)P_{00} + (0.6)(1.0 - 0.7)P_{01} + (1.0 - 0.6)(0.7)P_{10} + (1.0 - 0.6)(1.0 - 0.7)P_{11}$$

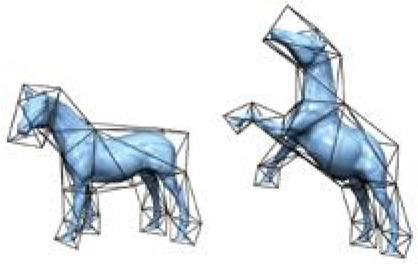




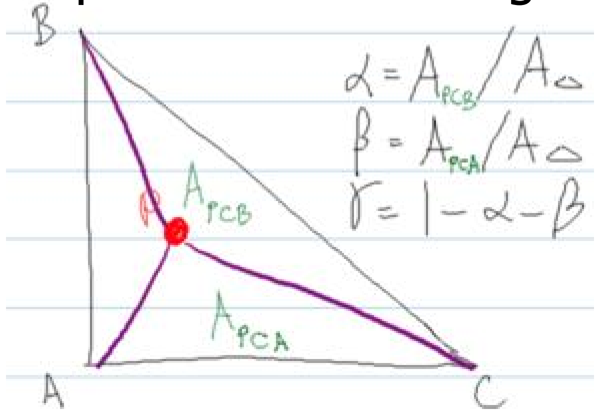
FFD

- ✓ Free-form deformation: lattice-based or cage-based.





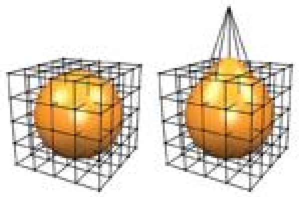
- ✓ Other grid cells also possible: triangle cells.
- ✓ Barycentric Coordinates: any point P in a triangle can be expressed as:
  P = alpha A + beta B + gamma C

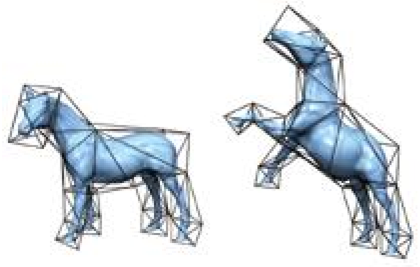

B
A
C
P
A_{PCB}
A_{PCA}
$\alpha = A_{PCB} / A_{\triangle}$
$\beta = A_{PCA} / A_{\triangle}$
$\gamma = 1 - \alpha - \beta$

- ✓ Idea: If we move any of the triangle's vertices or if we apply any transformation to the triangle, we can use the Barycentric Coordinates of any point on the original triangle to compute their corresponding position on the new triangle.

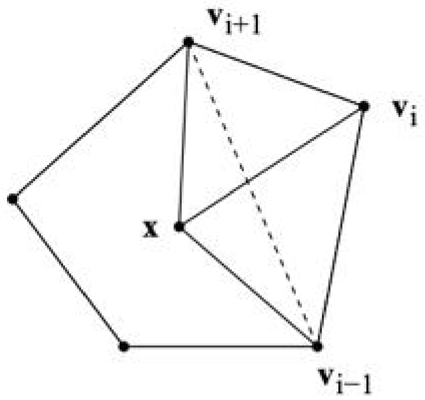
FFD

- ✓ Free-form deformation: lattice-based or cage-based.





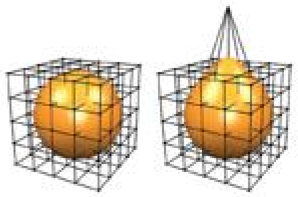
- ✓ Other grid cells also possible: convex polygon cells.
- ✓ Wachspress Coordinates: any point x in a convex polygon can be expressed.

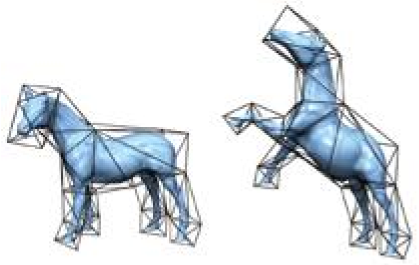


- ✓ Idea: If we move any of the polygon's vertices or if we apply any transformation to the polygon, we can use the Wachspress Coordinates of any point on the original polygon to compute their corresponding position on the new polygon.

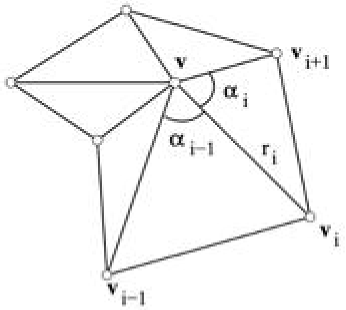
FFD

- ✓ Free-form deformation: lattice-based or cage-based.





- ✓ Other grid cells also possible: arbitrary polygon cells.
- ✓ Mean Value Coordinates: any point v in any polygon can be expressed.

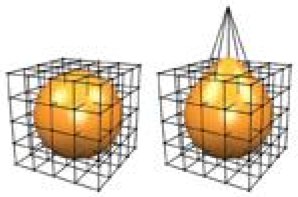


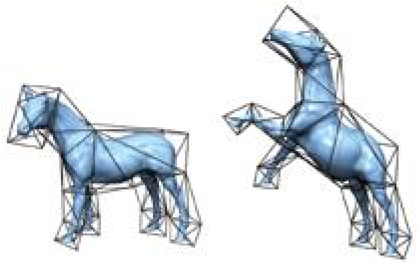
$$\lambda_i = w_i \Big/ \sum_{j=1}^{n} w_j, \qquad w_i = \frac{1}{r_i}\left(\tan\frac{\alpha_i}{2} + \tan\frac{\alpha_{i-1}}{2}\right),$$

- ✓ Idea: If we move any of the polygon’s vertices or if we apply any transformation to the polygon, we can use the Mean Value Coordinates of any point on the original polygon to compute their corresponding position on the new polygon.

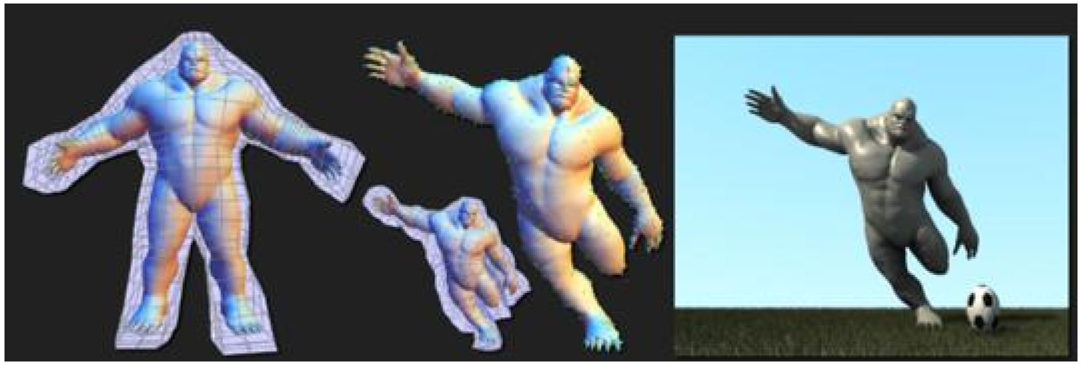
FFD

- ✓ Free-form deformation: lattice-based or cage-based.





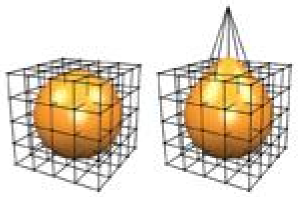
- ✓ Other grid cells also possible: arbitrary polyhedron cells.
- ✓ 3D Mean Value Coordinates: any point in any polyhedron can be expressed.

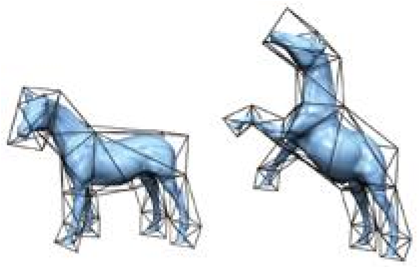


- ✓ Idea: If we move any of the polyhedron's vertices or if we apply any transformation to the polyhedron, we can use the 3D Mean Value Coordinates of any point on the original polyhedron to compute their correspondng position on new polyhedron.

FFD

- ✓ Free-form deformation: lattice-based or cage-based.

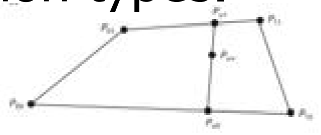




- ✓ Cool sources for details on Barycentric, Wach, Mean Value Coordinates.
  - ✓ http://cgvr.cs.uni-bremen.de/teaching/cg_literatur/barycentric_floater.pdf
  - ✓ https://www.mn.uio.no/math/english/people/aca/michaelf/papers/wach_mv.pdf
  - ✓ http://folk.uio.no/martinre/Publications/mv3d.pdf
  - ✓ https://www.youtube.com/watch?v=PDJxjXCjiis
  - ✓ http://hecodes.com/2016/07/mesh-manipulation-using-mean-values-coordinates-in-three-js/

Interpolation

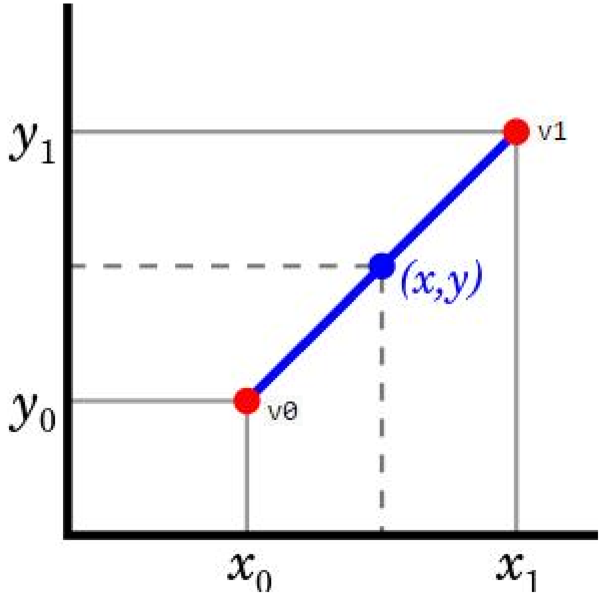
- ✓ FFD is all about interpolation. Let’s see some interpolation types.

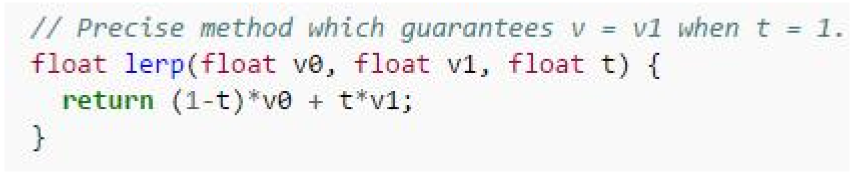


- ✓ Linear interpolation: interpolation over a 1D line.
  - ✓ Find the value of f at point x: f(x)
- ✓ Bilinear interpolation: interpolation over a 2D plane.
  - ✓ Find the value of f at point (x, y): f(x, y)
- ✓ Trilinear interpolation: interpolation over a 3D volume.
  - ✓ Find the value of f at point (x, y, z): f(x, y, z)

Interpolation

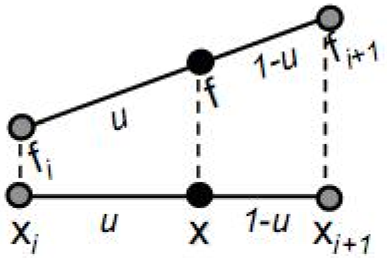
- ✓ Linear interpolation: interpolation over a 1D line.
- ✓ Interpolate points (or other attributes) on a line from v_0 to v_1 using t.




```
// Precise method which guarantees v = v1 when t = 1.
float lerp(float v0, float v1, float t) {
  return (1-t)*v0 + t*v1;
}
```

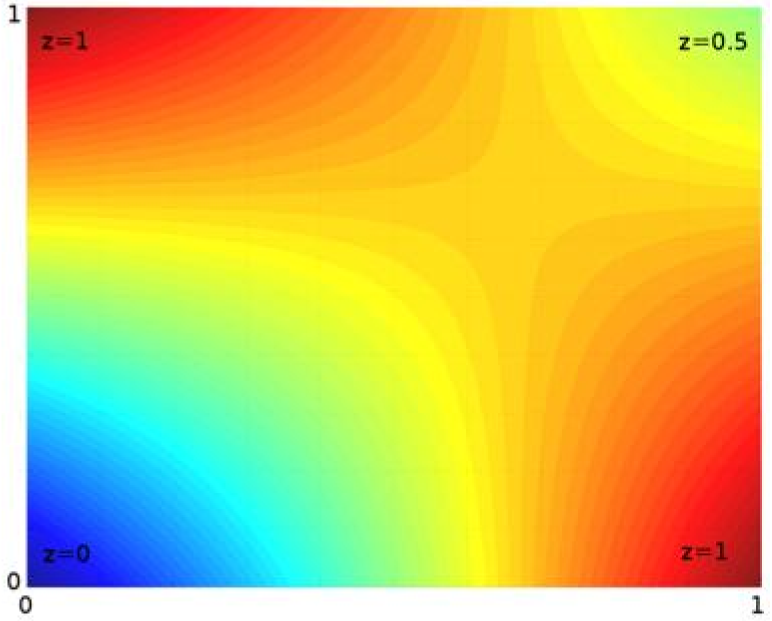
$x = x_i + u(x_{i+1} - x_i) = (1-u)x_i + ux_{i+1}$;

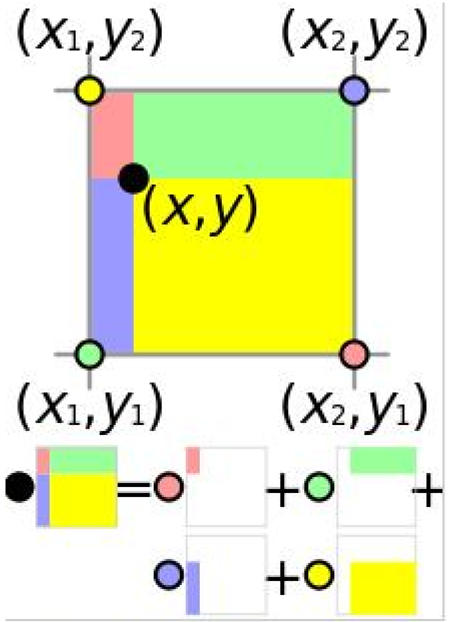
$f(x) = (1-u)f_i + uf_{i+1}$



Interpolation

- ✓ Bilinear interpolation: interpolation over a 2D plane.
- ✓ Interpolate .. using u and v.

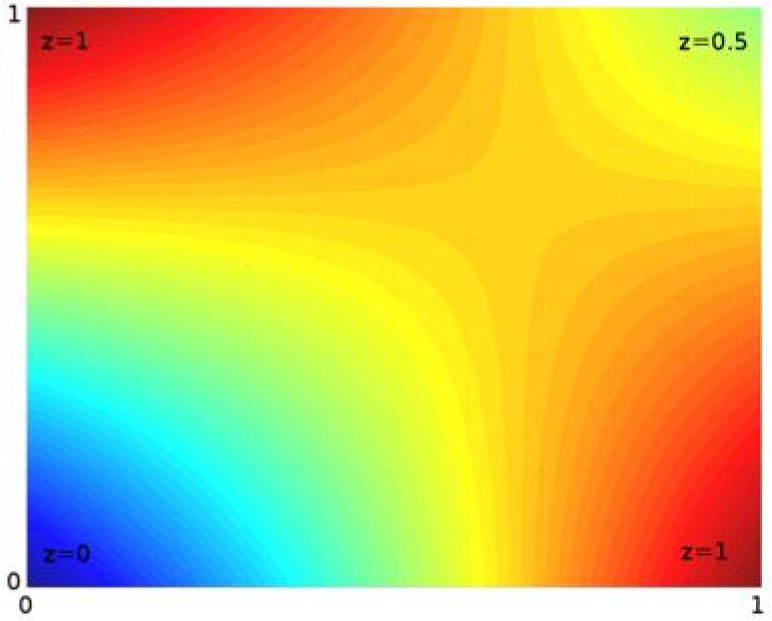




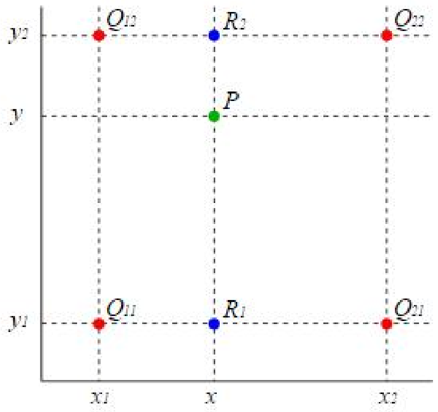
In this geometric visualisation, the value at the black spot is the sum of the value at each coloured spot multiplied by the area of the rectangle of the same colour, divided by the total area of all four rectangles.

Interpolation

- ✓ Bilinear interpolation: interpolation over a 2D plane.
- ✓ Interpolate .. using u and v.



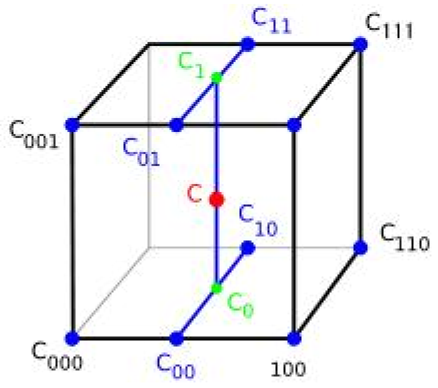
more formally, we do lerp in 1 direction, and then again in the other direction.

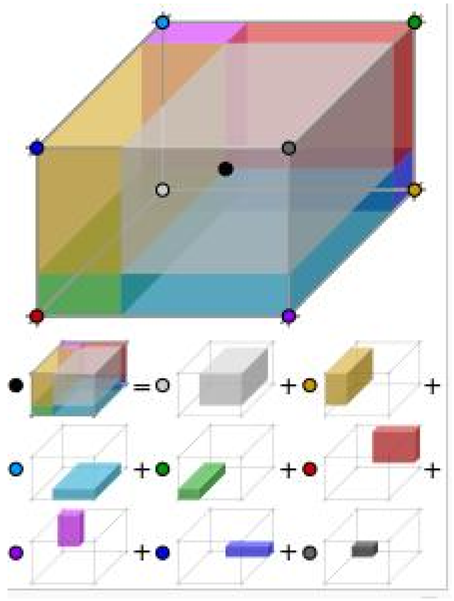


do lerp to find R1, then R2.
Then do lerp using R1, R2; get P.

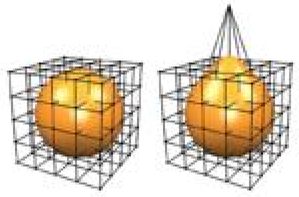
Interpolation

- ✓ Trilinear interpolation: interpolation over a 3D cell.
- ✓ Interpolate .. using u, v, and w.





- ✓ More formally, lerp of 2 bilerps (one for the bottom face, one for top).
- ✓ u and v for the bilerps, w for the final lerp.
- ✓ FFD in 3D employs trilinear interpolation:

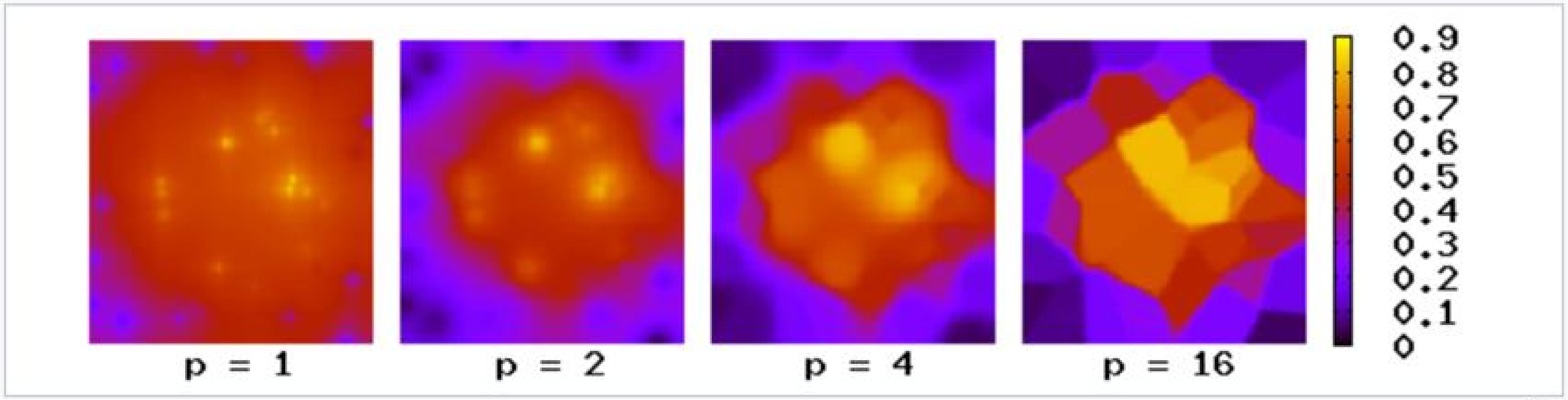


Interpolation

- ✓ Interpolation without cells, i.e., scattered data interpolation, can be done via Shepard interpolation (1968); inverse distance weighting.

$$f(\mathbf{x}) = \Sigma_i \phi_i(\mathbf{x})\, f_i \qquad \phi_i(\mathbf{x}) = \frac{\|\mathbf{x}-\mathbf{x}_i\|^{-p}}{\sum_j \|\mathbf{x}-\mathbf{x}_j\|^{-p}}$$

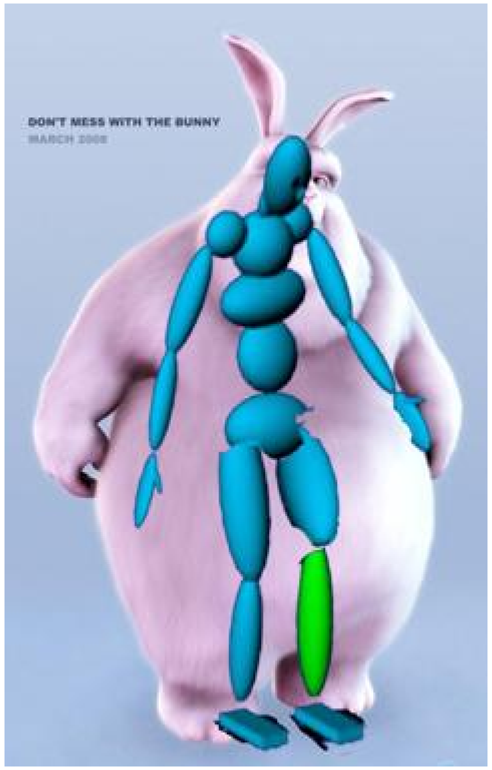
- ✓ x_i: scattered points, f_i: function values at scattered points.
- ✓ Neighborhood not given explicitly: specify in terms of radius, # points.

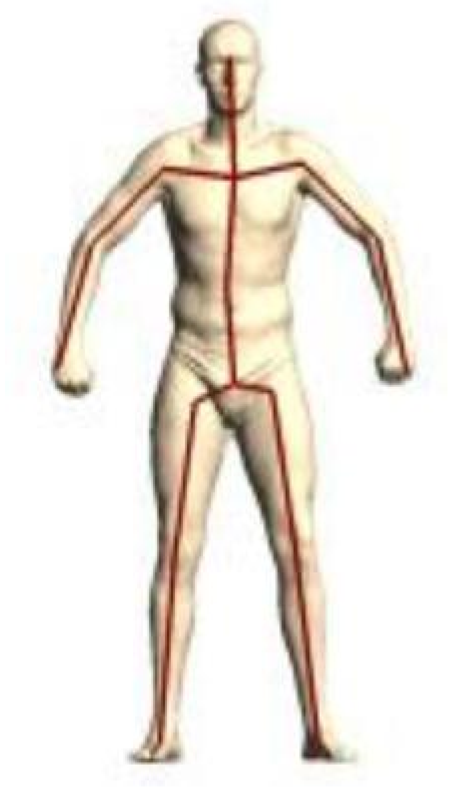


Shepard's interpolation for different power parameters *p*, from scattered points on the surface

Skinning

- ✓ A variant of FFD is skinning/enveloping, where you put the control "cage" inside the shape, i.e., skeleton controls the surfce deformation.



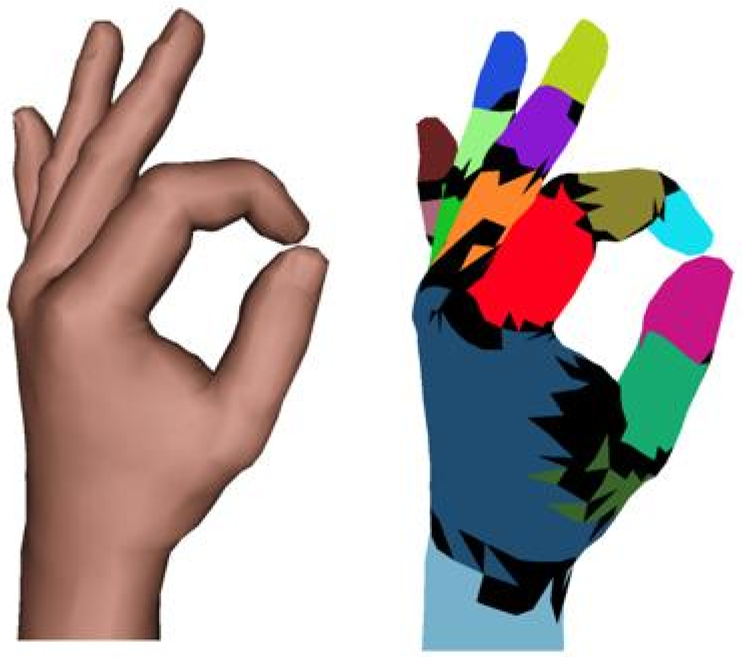


Skinning

- ✓ A variant of FFD is skinning/enveloping, where you put the control "cage" inside the shape, i.e., skeleton controls the surfce deformation.
- ✓ Bind skin vertices to bones.
- ✓ Animate skeleton (easy: change the joint angles), skin'll move with it.
- ✓ Skinning is aka
  - ✓ Skeletal subspace deformation (SSD).
  - ✓ Linear blend skinning (LBS).

Skinning

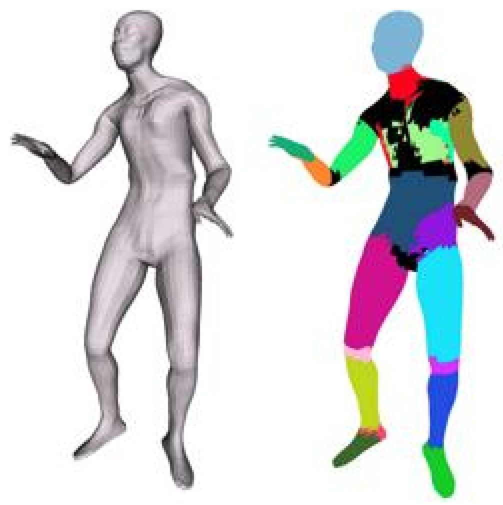
✓ Skinning.



Colored triangles are attached to 1 bone

Black triangles are attached to more than 1

Note how they are near joints



Skinning

- ✓ Skinning weights (manual or by using example poses & skins).
- ✓ w_{ij}: how much surface vertex p_i moves with bone j (1 means rigid).

“
$$\boxed{\begin{gathered} \boldsymbol{p}'_{ij} = \boldsymbol{T}_j \boldsymbol{p}_i \\ \boldsymbol{p}'_i = \sum_j w_{ij} \boldsymbol{p}'_{ij} \end{gathered}}$$
”

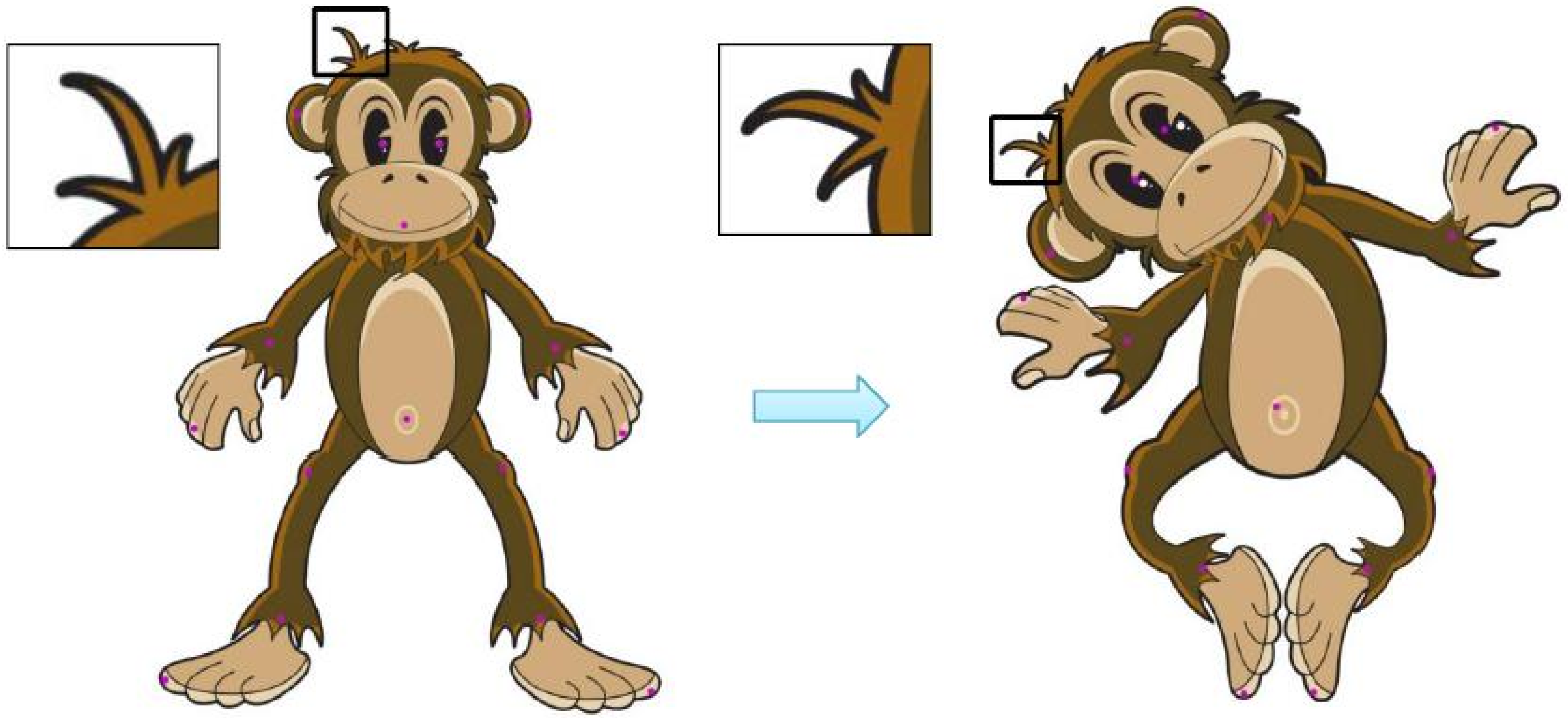
p’ij is the vertex i transformed using bone j.
Tj is the current transformation of bone j.
p’i is the new skinned position of vertex i.

- ✓ Some details involved: bind pose transformations B_j come in to express p_i in the local coordinate system of bone j. Only then T_j makes sense.

$$\boldsymbol{p}'_{ij} = \boldsymbol{T}_j \boldsymbol{B}_j^{-1} \boldsymbol{p}_i$$

Shell-based Deformation

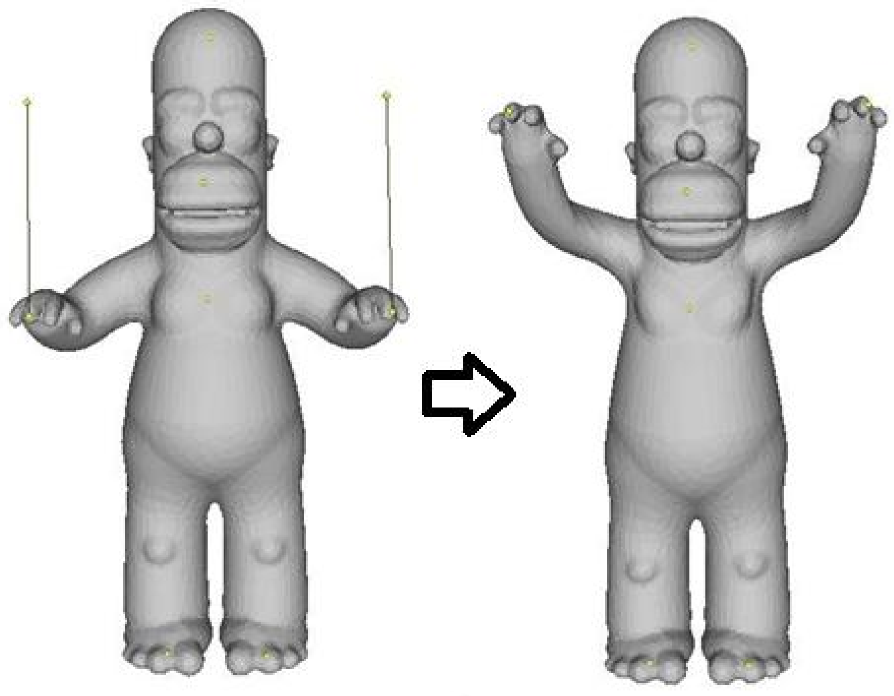
✓ Challenge: intuitive deformation requires global change + local detail preservation.



Shell-based Deformation

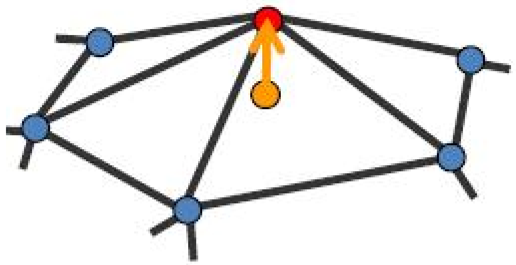
- ✓ Shell-based deformation using Laplacian coordinates
  - ✓ Laplacian deformation, in short.
  - ✓ Laplacian coordinates = differential coordinates = delta coordinates

- ✓ Control mechanism
  - ✓ Handles (vertices) moved by user.
  - ✓ Region of influence (ROI) is deformed accordingly.



Differential Coordinates

- ✓ Difficult to do operations on meshes that store the geometry in Absolute Coordinates.
  - ✓ Doing deformation that preserve overall shape is difficult operating on point-by-point basis.
- ✓ A solution: Use Differential Coordinates.
  - ✓ v.delta encapsulates local shape information around v.
  - ✓ = approximation of mean curvature around v times the normal direction at v.



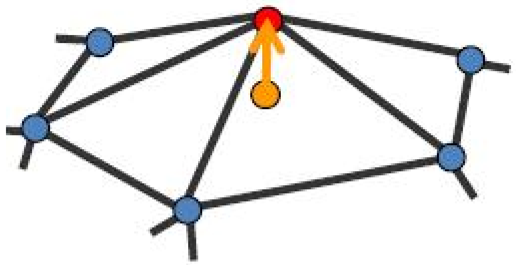
$$\vec{\delta_i} = \vec{v_i} - \frac{\sum_{j \epsilon N(i)} w_{ij}\vec{v_j}}{\sum_{j \epsilon N(i) w_{ij}}} \xrightarrow[\text{(umbrella weighting)}]{} \vec{\delta_i} = \vec{v_i} - \frac{1}{d_i}\sum_{j \epsilon N(i)} \vec{v_j}$$

weights=1

(umbrella weighting)

Differential Coordinates

- ✓ Difficult to do operations on meshes that store the geometry in Absolute Coordinates.
  - ✓ Doing deformation that preserve overall shape is difficult operating on point-by-point basis.
- ✓ A solution: Use Differential Coordinates.
  - ✓ v.delta encapsulates local shape information around v.
  - ✓ = approximation of mean curvature around v times the normal direction at v.



$$\vec{\delta_i} = \vec{v_i} - \frac{1}{d_i} \sum_{j \epsilon N(i)} \vec{v_j}$$

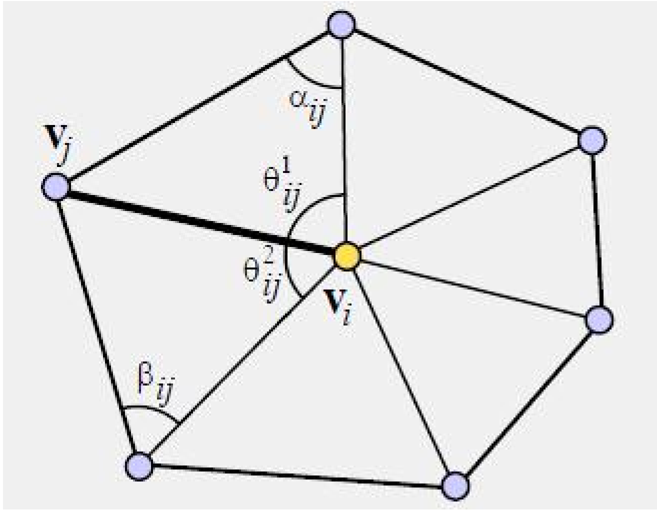
vector from center of 1-ring neighbors to vi

$$\vec{\delta_i} = \frac{1}{d_i} \sum_{j \epsilon N(i)} \vec{v_i} - \vec{v_j}$$

average those vectors

Differential Coordinates

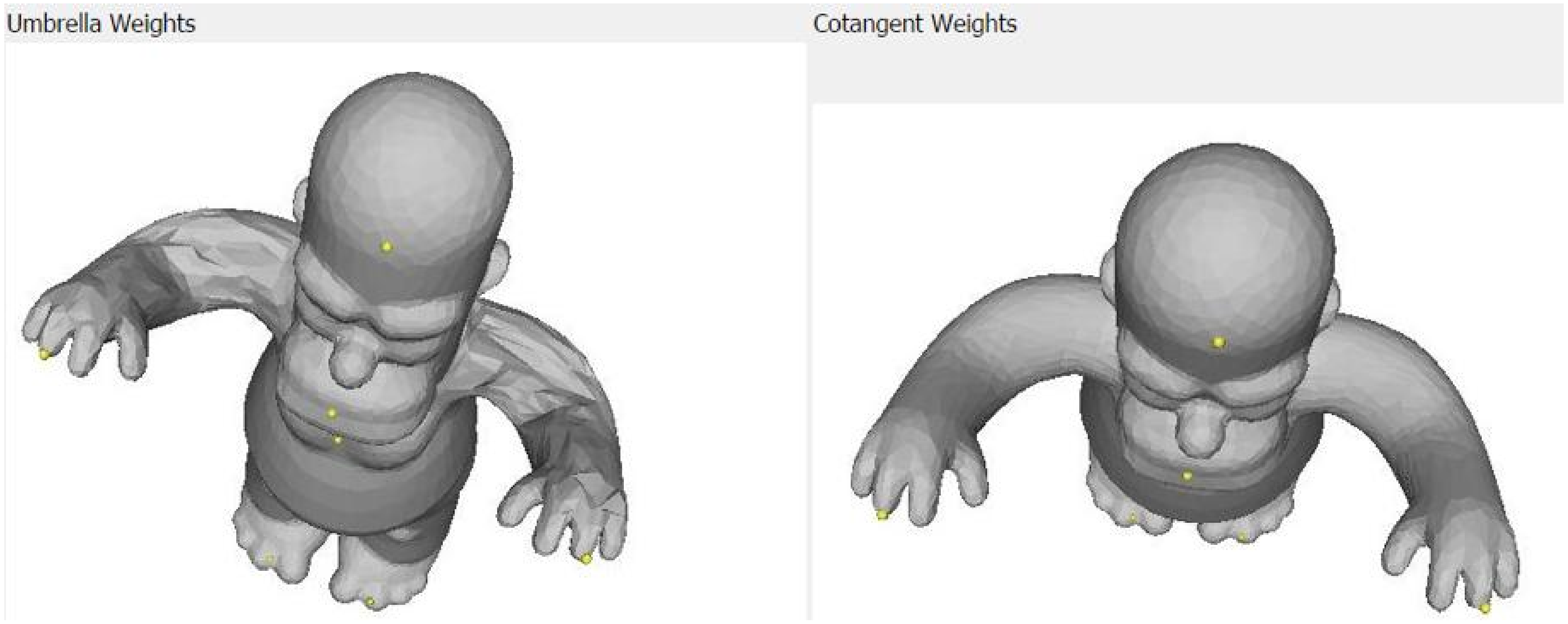
- ✓ Differential Coordinates by cotangent weighting.
  - ✓ A more geometry-aware weighting scheme than umbrella weighting.
  - ✓ Invariant to different tessellations.
  - ✓ Angle-based: preserves angles (and consequently areas) much better than umbrella wi = 1 weighting.

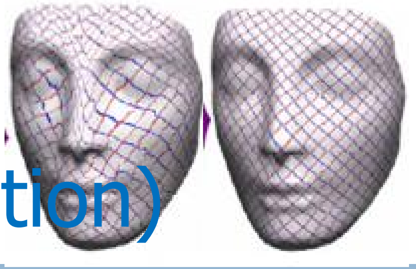


$$w_{ij} = \frac{1}{2}(cot\alpha_{ij} + cot\beta_{ij})$$

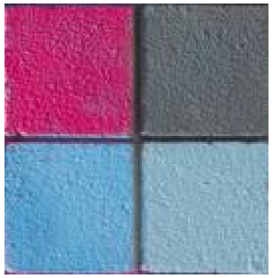
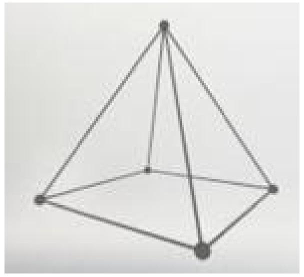
Differential Coordinates

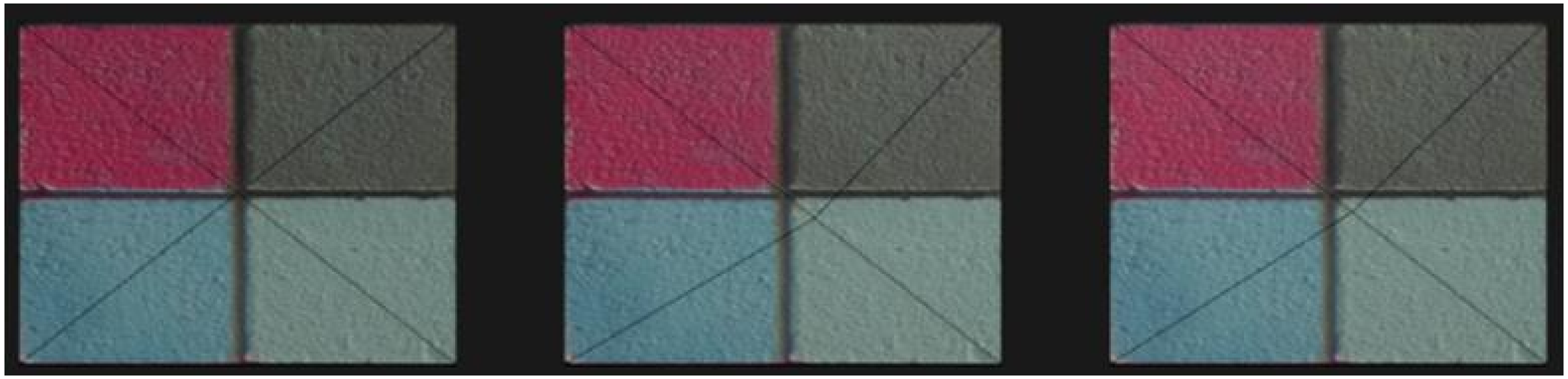
- ✓ Differential Coordinates by cotangent weighting.
  - ✓ Resulting deformation when umbrella (left) vs. cot (right) weighting in use:



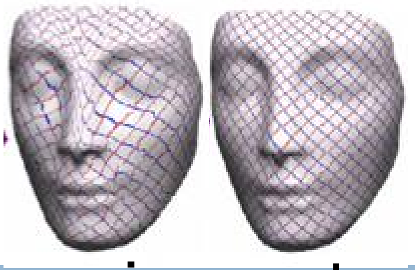


Differential Coordinates (Parameterization)

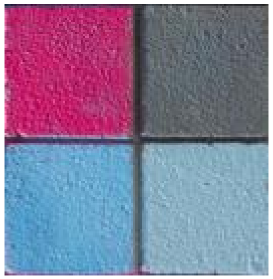
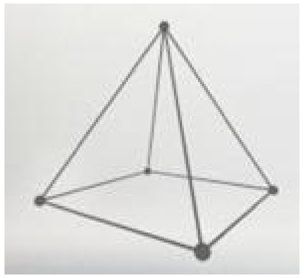
- ✓ Cotangent weighting advantage can be seen better w/ parameterization.
  - ✓ Copied from my Mesh Parameterization slides.
- ✓ Didactic example: texture and mesh are  and  .
- ✓ Pyramid's (no floor) pole slightly moved so its projection not in center.
- ✓ 2D parameterization overlayed w/ texture (uniform, cotan, mean-val):



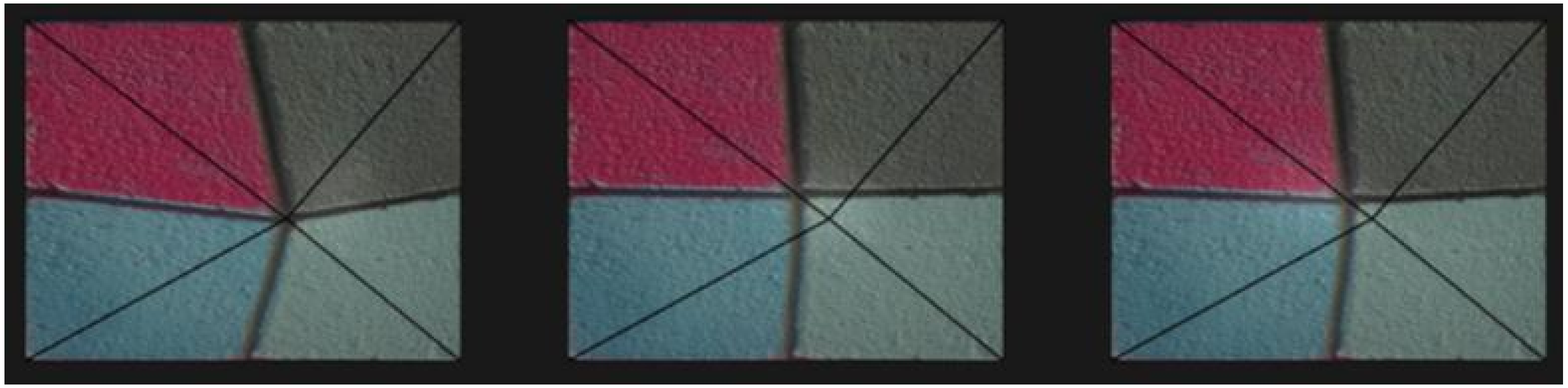
Differential Coordinates (uni. vs. cot.)



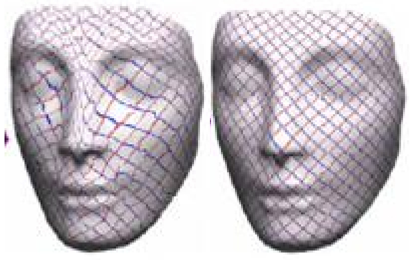
- ✓ Cotangent weighting advantage can be seen better w/ parameterization.
  - ✓ Copied from my Mesh Parameterization slides.

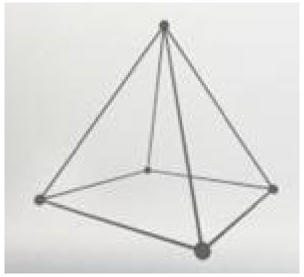
- ✓ Didactic example: texture and mesh are  and  .

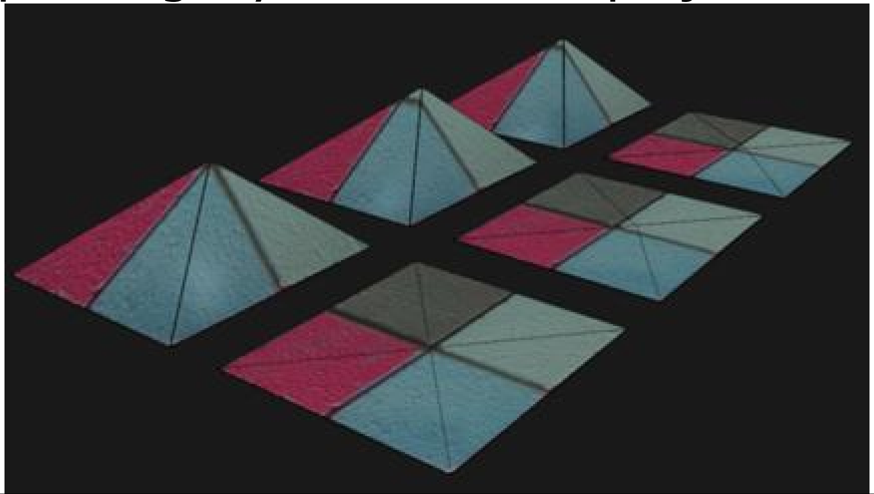
- ✓ Pyramid's (no floor) pole slightly moved so its projection not in center.
- ✓ 3D w/ pulled-back texture (birdeye view); note the distortion at the left.



Differential Coordinates (uni. vs. cot.)

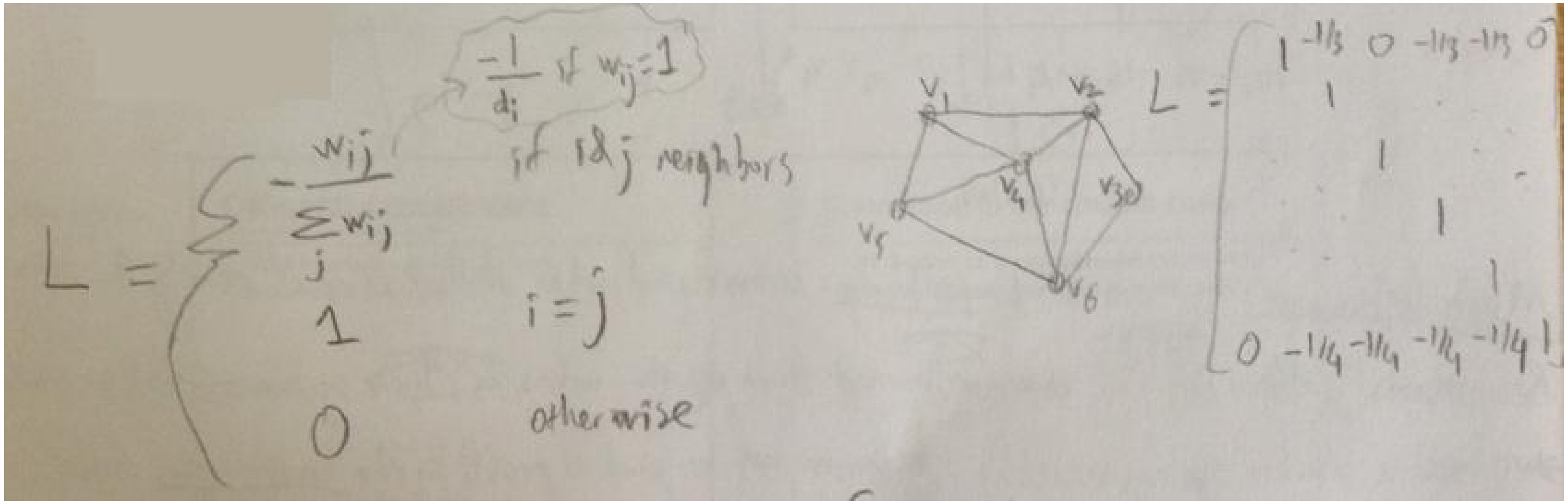


- ✓ Cotangent weighting advantage can be seen better w/ parameterization.
  - ✓ Copied from my Mesh Parameterization slides.
- ✓ Didactic example: texture and mesh are and  .
- ✓ Pyramid's (no floor) pole slightly moved so its projection not in center.
- ✓ Not so clear w/ non-birdeye view:



Differential Coordinates

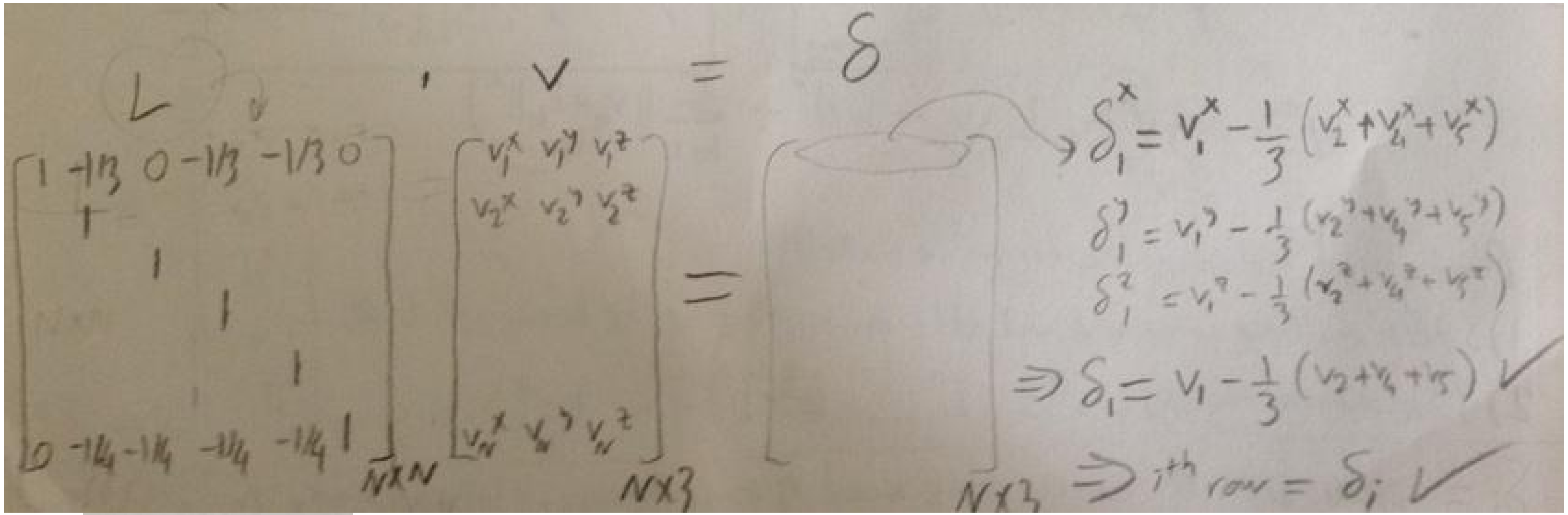
- ✓ Differential Coordinates by Laplacian Matrix **L.**
  - ✓ Since a differential coordinate is a linear combination of a vertex and its neighbors, the process of constructing differential coordinates for all vertices can be represented as a matrix **L** such that **L**v = delta.


$-\frac{1}{d_i}$ if $w_{ij}=1$
$L = \begin{cases} -\frac{w_{ij}}{\sum_j w_{ij}} & \text{if } i \& j \text{ neighbors} \\ 1 & i=j \\ 0 & \text{otherwise} \end{cases}$
v_1
v_2
v_3
v_4
v_5
v_6
$L =$
1 -1/3 0 -1/3 -1/3 0
0 -1/4 -1/4 -1/4 -1/4 1

$$L_{ij} = \begin{cases} 1 & i = j \\ -\frac{1}{d_i} & (i,j) \in E \\ 0 & \text{otherwise.} \end{cases}$$

Differential Coordinates

- ✓ Differential Coordinates by Laplacian Matrix **L.**
  - ✓ Since a differential coordinate is a linear combination of a vertex and its neighbors, the process of constructing differential coordinates for all vertices can be represented as a matrix **L** such that **L**v = delta.



$$L \cdot V = \delta$$

$$\begin{bmatrix} 1 & -1/3 & 0 & -1/3 & -1/3 & 0 \\ & 1 & & & & \\ & & 1 & & & \\ & & & 1 & & \\ & & & & 1 & \\ 0 & -1/4 & -1/4 & -1/4 & -1/4 & 1 \end{bmatrix}_{N\times N} \begin{bmatrix} v_1^x & v_1^y & v_1^z \\ v_2^x & v_2^y & v_2^z \\ & \vdots & \\ v_N^x & v_N^y & v_N^z \end{bmatrix}_{N\times 3} = \begin{bmatrix} \\ \\ \\ \end{bmatrix}_{N\times 3}$$

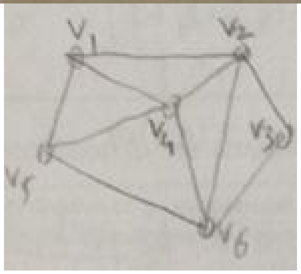
$$\delta_1^x = v_1^x - \frac{1}{3}(v_2^x + v_4^x + v_5^x)$$

$$\delta_1^y = v_1^y - \frac{1}{3}(v_2^y + v_4^y + v_5^y)$$

$$\delta_1^z = v_1^z - \frac{1}{3}(v_2^z + v_4^z + v_5^z)$$

$$\Rightarrow \delta_1 = v_1 - \frac{1}{3}(v_2 + v_4 + v_5) \checkmark$$

$$\Rightarrow i^{th} \text{ row} = \delta_i \checkmark$$



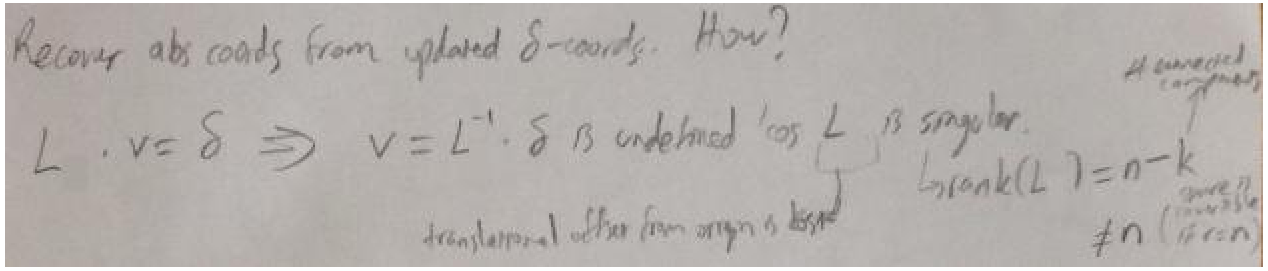
Recall, $\vec{\delta_i} = \vec{v_i} - \frac{1}{d_i} \sum_{j \in N(i)} \vec{v_j}$.

Differential Coordinates

- ✓ Differential Coordinates by Laplacian Matrix **L.**
  - ✓ NxN matrix **L** is extremely sparse: degree = 6 → (6+1) out of N non-zero entries per row where N is the number of vertices; typically N > 10000.
  - ✓ Use a sparse matrix package: Eigen http://eigen.tuxfamily.org/index.php?title=Main_Page
  - ✓ Eigen'll also be handy while solving linear system Ax = b for the deformation.
  - ✓ Useful for eigenanalysis too.
  - ✓ In short, a cool linear algebra library.

Differential Coordinates (Surface Reconstruction)

- ✓ Recovering Absolute Coordinates from Differential Coordinates.
  - ✓ delta stores differential coordinates, which throws away all info about Absolute positions → translation offset from origin is lost.
  - ✓ To recover/put mesh somewhere in space, try some basic linear algebra:



Recover abs coords from updated δ-coords. How?

$L \cdot v = \delta \Rightarrow v = L^{-1} \cdot \delta$ is undefined 'cos L is singular.

$\hookrightarrow \text{rank}(L) = n - k$ # connected components

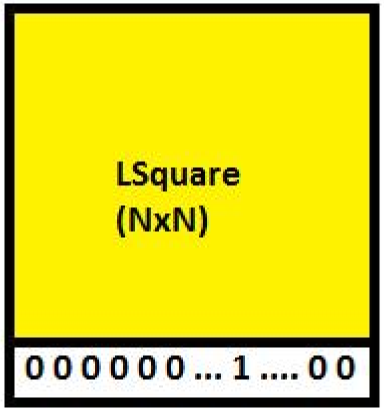
$\neq n$ (invertible iff r=n)

translational offset from origin is lost

  - ✓ This trial fails 'cos L is not full-rank → not invertible.

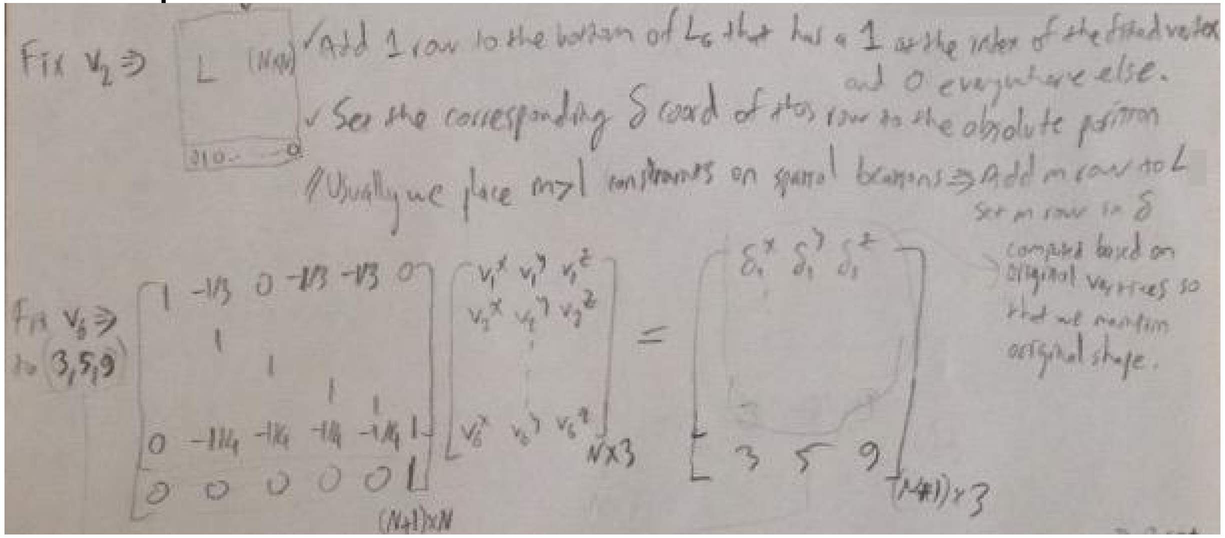
Differential Coordinates (Surface Reconstruction)

- ✓ Recovering Absolute Coordinates from Differential Coordinates.
  - ✓ delta stores differential coordinates, which throws away all info about Absolute positions → translation offset from origin is lost.
  - ✓ To recover/put mesh somewhere in space, specify the location of exactly 1 vertex (fixed).
    - ✓ To do this, simply add one row to the bottom of the **L** matrix that has a 1 at the index of this fixed vertex and zeros everywhere else, and set the corresponding differential coordinate of this row to be the desired absolute position of that vertex.



Differential Coordinates (Surface Reconstruction)

- ✓ Recovering Absolute Coordinates from Differential Coordinates.
  - ✓ delta stores differential coordinates, which throws away all info about Absolute positions → translation offset from origin is lost.
  - ✓ To recover/put mesh somewhere in space, specify the location of exactly 1 vertex (fixed).
    - ✓ To do this, simply add one row to the bottom of the **L** matrix that has a 1 at the index of this fixed vertex and zeros everywhere else, and set the corresponding differential coordinate of this row to be the desired absolute position of that vertex.



Fix $v_2 \Rightarrow$ L (N×N) 010…0

✓ Add 1 row to the bottom of L s.t. has a 1 at the index of the fixed vertex and 0 everywhere else.

✓ Set the corresponding δ coord of this row to the absolute position

// Usually we place m>1 constraints on spatial locations ⇒ Add m row to L, set m row in δ

Fix $v_6 \Rightarrow$ to (3,5,9)

$$\underbrace{\begin{bmatrix} 1 & -1/3 & 0 & -1/3 & -1/3 & 0 \\ & 1 & & & & \\ & & 1 & & & \\ & & & 1 & 1 & \\ 0 & -1/4 & -1/4 & -1/4 & -1/4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{(N+1)\times N} \underbrace{\begin{bmatrix} v_1^x & v_1^y & v_1^z \\ v_2^x & v_2^y & v_2^z \\ & \vdots & \\ v_6^x & v_6^y & v_6^z \end{bmatrix}}_{N\times 3} = \underbrace{\begin{bmatrix} \delta_1^x & \delta_1^y & \delta_1^z \\ & \vdots & \\ 3 & 5 & 9 \end{bmatrix}}_{(N+1)\times 3}$$

→ computed based on original vertices so that we maintain original shape.

Differential Coordinates (Surface Reconstruction)

- ✓ Recovering Absolute Coordinates from Differential Coordinates.
  - ✓ delta stores differential coordinates, which throws away all info about Absolute positions → translation offset from origin is lost.
  - ✓ To recover/put mesh somewhere in space, specify the location of exactly 1 vertex (fixed).
    - ✓ To do this, simply add one row to the bottom of the **L** matrix that has a 1 at the index of this fixed vertex and zeros everywhere else, and set the corresponding differential coordinate of this row to be the desired absolute position of that vertex.
    - ✓ In general, suppose m fixed vertices, leading to (n+m) x n sys mtrx $\tilde{L}$.

$$\left(\frac{L}{\omega I_{m\times m} \mid 0}\right)\mathbf{x} = \begin{pmatrix} \delta^{(x)} \\ \omega c_{1:m} \end{pmatrix}$$

Analytical solution:

$$\tilde{\mathbf{x}} = (\tilde{L}^T \tilde{L})^{-1} \tilde{L}^T \mathbf{b}$$

$$\mathbf{b} = (\delta, \omega c_1, \ldots, \omega c_m)^T$$

Least-squares solution:

$$\tilde{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} \left(\left\| L\mathbf{x} - \delta^{(x)} \right\|^2 + \sum_{j \in C} \omega^2 \left| x_j - c_j \right|^2 \right)$$

- ✓ See Section 2.2 in paper: Differential Representations for Mesh Processing.

Differential Coordinates (Deformation)

- ✓ Recovery/reconstruction can also be casted as mesh deformation through control points.
  - ✓ See Section 2.2 in paper: Differential Representations for Mesh Processing.
  - ✓ Or the older paper: Least-squares Meshes.

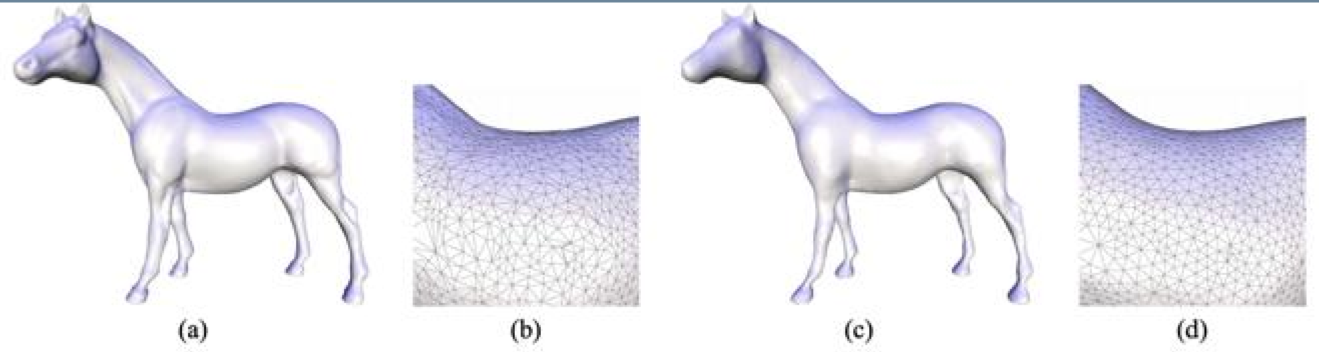


Figure 2. (a) The original horse model, 19851 vertices; (b) close-up on the original connectivity; (c) LS-mesh of the horse model with 1K control vertices; (d) close-up on the LS-mesh connectivity.

➔ Byproduct: remeshing, also Slide 62.

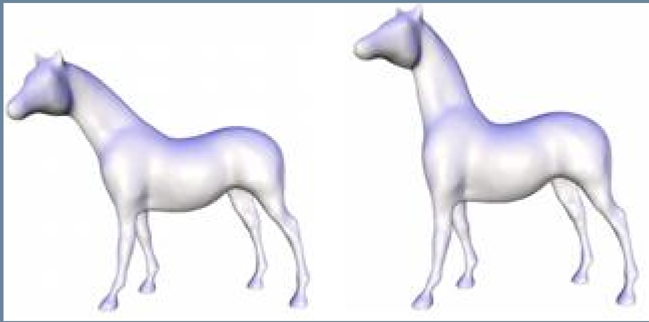


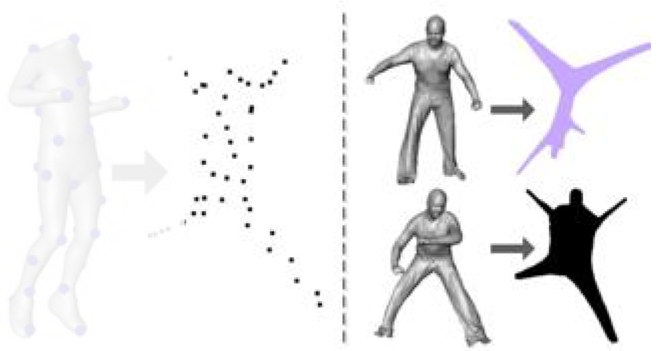
Figure 10. An example of editing the LS-mesh. On the left the original LS-mesh is shown; on the right the resulting LS-mesh after moving the control points on the head.

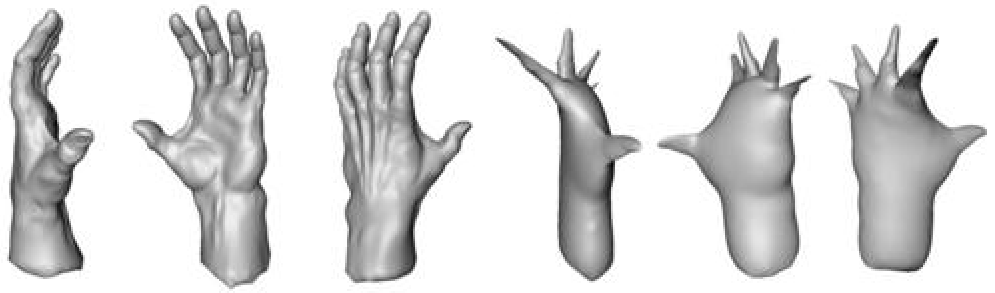
$$\tilde{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} \left(\left\| L\mathbf{x} - \delta^{(x)} \right\|^2 + \sum_{j \in C} \omega^2 \left| x_j - c_j \right|^2 \right)$$

x is nx1 size column vector storing x coords of n vertices. $\delta^{(x)}$ is nx1 col vector for x comp. of rest-pose delta coords. Similarly, solve for the y and z coords separately.

Differential Coordinates (Deformation)

- ✓ Recovery/reconstruction can also be casted as mesh deformation through control points.
  - ✓ Application: embed landmarks w/ an inherently slow process (all-pairs geodsics needed), then use *Least-squares Meshes* to embed the remaining mesh verts; resulting triangles'll be shaped similarly to the original mesh thanks to L matrix (regularization term) and embedded pnts'll act as control pnts (matching term).





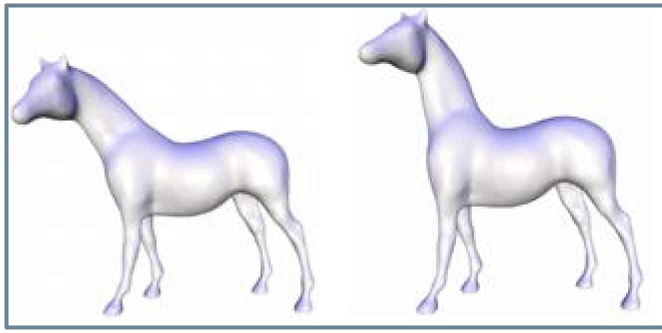


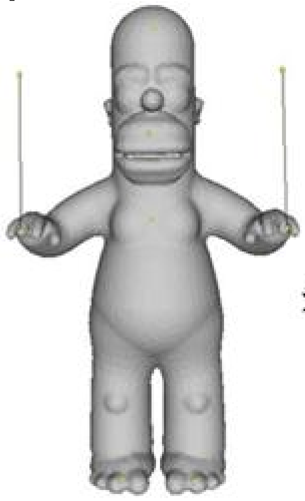
Figure 10. An example of editing the LS-mesh. On the left the original LS-mesh is shown; on the right the resulting LS-mesh after moving the control points on the head.

$$\tilde{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} \left(\underbrace{\left\| L\mathbf{x} - \delta^{(x)} \right\|^2}_{\text{E\_regu}} + \underbrace{\sum_{j \in C} \omega^2 \left| x_j - c_j \right|^2}_{\text{E\_match}} \right)$$

c_j: ctrl pnts

Differential Coordinates (Deformation)

- ✓ Mesh deformation using Differential Coordinates.
- ✓ Idea: match term competes with the regularization term.
- ✓ Match term: Move handle-vertices to the user-specified points as close as possible.
- ✓ Regularization term: Preserve the original features before the deformation, e.g., angles, areas, volumes, edge lengths, ..
- ✓ Without regularization term, match term is satisfied exactly, which disrespects the original properties:

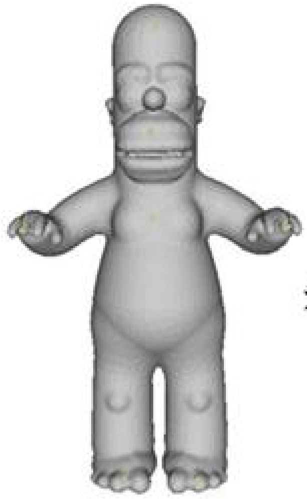


$$\tilde{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} \left(\left\| L\mathbf{x} - \delta^{(x)} \right\|^2 + \sum_{j \in C} \omega^2 \left| x_j - c_j \right|^2 \right)$$

E_regu E_match

Differential Coordinates (Deformation)

- ✓ Mesh deformation using Differential Coordinates.
- ✓ Idea: match term competes with the regularization term.
- ✓ Match term: Move handle-vertices to the user-specified points as close as possible.
- ✓ Regularization term: Preserve the original features before the deformation, e.g., angles, areas, volumes, edge lengths, ..
- ✓ Without matching term, mesh does not move/deform at all because it is already happy with the initial configuration.

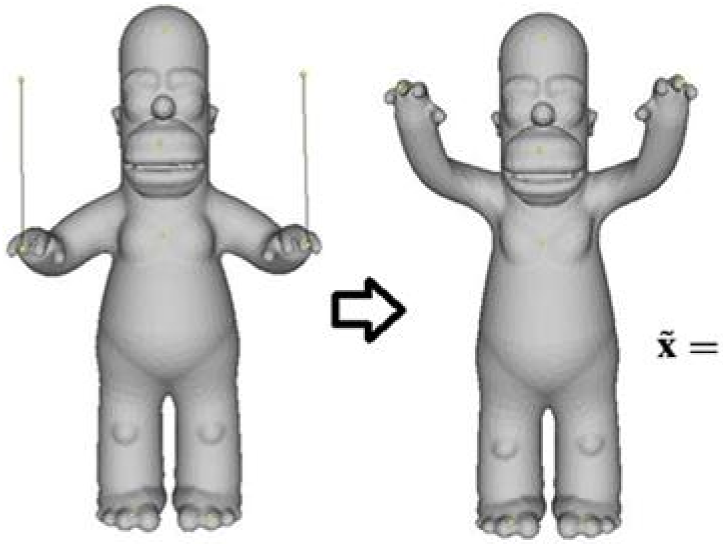


$$\tilde{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} \left(\left\| L\mathbf{x} - \delta^{(x)} \right\|^2 + \sum_{j \in C} \omega^2 \left| x_j - c_j \right|^2 \right)$$

E_regu E_match

Differential Coordinates (Deformation)

- ✓ Mesh deformation using Differential Coordinates.
- ✓ Idea: match term competes with the regularization term.
- ✓ Match term: Move handle-vertices to the user-specified points as close as possible.
- ✓ Regularization term: Preserve the original features before the deformation, e.g., angles, areas, volumes, edge lengths, ..
- ✓ With both terms, we find a compromise: neither is satisfied exactly.

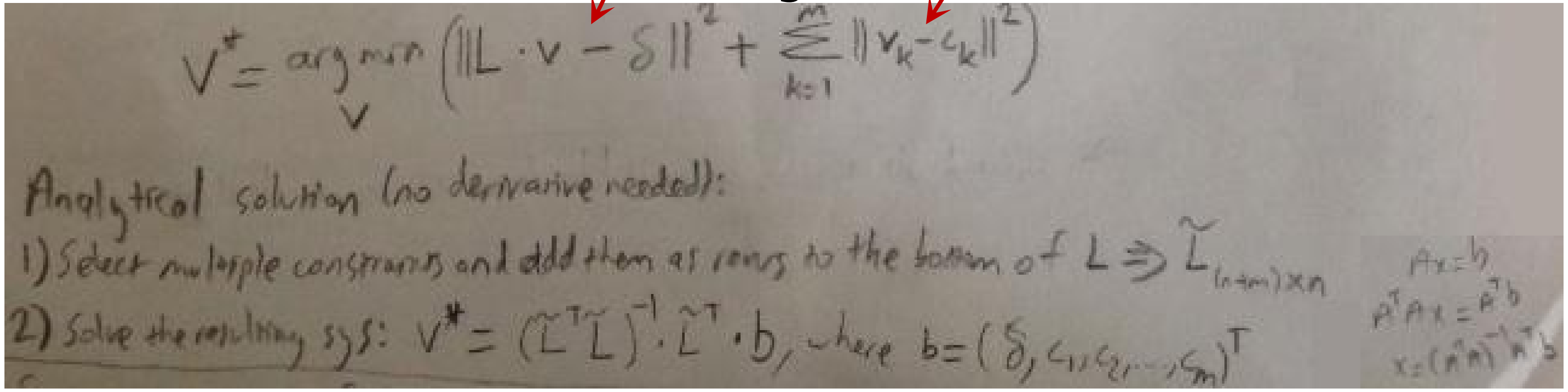


$$\tilde{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} \left(\left\| L\mathbf{x} - \delta^{(x)} \right\|^2 + \sum_{j \in C} \omega^2 \left| x_j - c_j \right|^2 \right)$$

E_regu E_match

Differential Coordinates (Deformation)

- ✓ Mesh deformation using Differential Coordinates.
- ✓ Idea: match term competes with the regularization term.
- ✓ Match term: Move handle-vertices to the user-specified points as close as possible.
- ✓ Regularization term: Preserve the original features before the deformation, e.g., angles, areas, volumes, edge lengths, ..
  - ✓ Differential coords encapsulates original local shape properties.
  - ✓ Regularization term wants to minimize the difference b/w new differential coordinates and original differential coords.



$$V^* = \underset{V}{\arg\min}\left(\|L\cdot v - \delta\|^2 + \sum_{k=1}^{m}\|v_k - c_k\|^2\right)$$

Analytical solution (no derivative needed):

1) Select multiple constraints and add them as rows to the bottom of $L \Rightarrow \tilde{L}_{(n+m)\times n}$

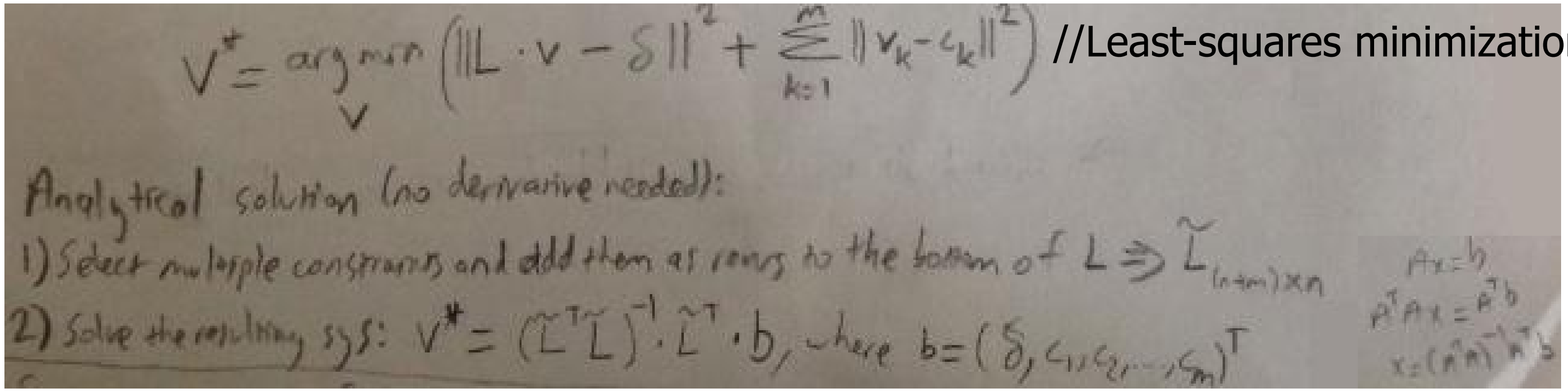
2) Solve the resulting sys: $v^* = (\tilde{L}^T\tilde{L})^{-1}\cdot\tilde{L}^T\cdot b$, where $b = (\delta, c_1, c_2, \ldots, c_m)^T$

$Ax = b$

$A^TAx = A^Tb$

$x = (A^TA)^{-1}A^Tb$

Differential Coordinates (Deformation)



$$V^* = \arg\min_{V} \left(\|L \cdot v - \delta\|^2 + \sum_{k=1}^{m} \|v_k - c_k\|^2 \right)$$ //Least-squares minimization.

Analytical solution (no derivative needed):

1) Select multiple constraints and add them as rows to the bottom of $L \Rightarrow \tilde{L}_{(n+m)\times n}$

2) Solve the resulting sys: $v^* = (\tilde{L}^T \tilde{L})^{-1} \cdot \tilde{L}^T \cdot b$, where $b = (\delta, c_1, c_2, \ldots, c_m)^T$

$Ax = b$

$A^T A x = A^T b$

$x = (A^T A)^{-1} A^T b$

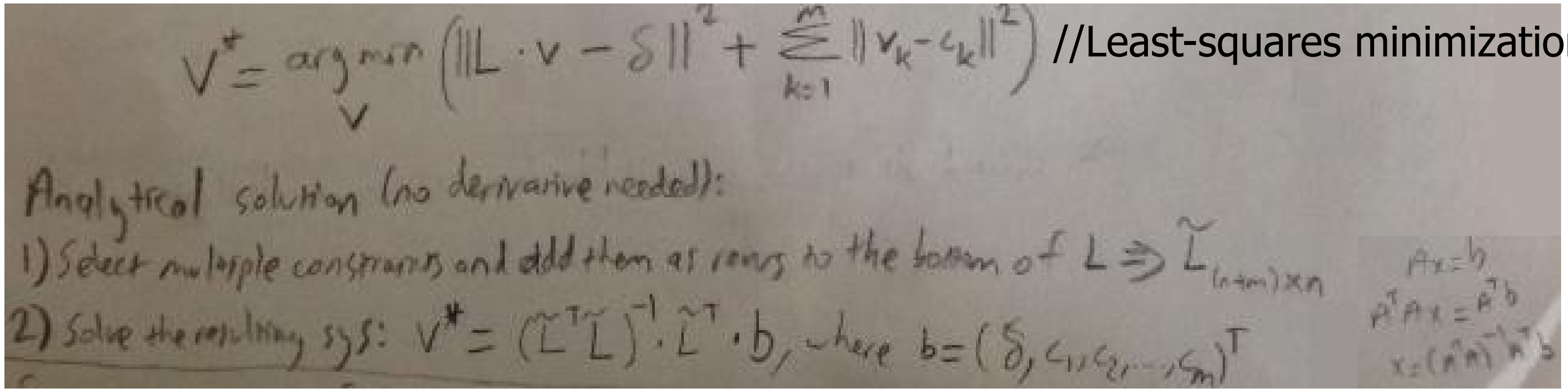
✓ Derivation of the analytical solution is based on this idea:

$$\arg\min_{v' \in V} \left\{ \sum_{\{i,j\} \in E} \left\| (v'_j - v'_i) - (v_j - v_i) \right\|^2 + w \sum_{i \in C} \|v'_i - C_i\|^2 \right\}$$

can be rewritten as $\arg\min_{\mathbf{v}'} \left\| A\mathbf{v}' - \mathbf{b} \right\|^2$. Then, solve this separately for x-, y-, z-components.

Showing it for the x-component in the next slide.

Differential Coordinates (Deformation)



$$V^* = \underset{V}{\arg\min}\left(\|L\cdot v - \delta\|^2 + \sum_{k=1}^{m}\|v_k - c_k\|^2\right)$$ //Least-squares minimization.

Analytical solution (no derivative needed):

1) Select multiple constraints and add them as rows to the bottom of $L \Rightarrow \tilde{L}_{(n+m)\times n}$

2) Solve the resulting sys: $V^* = (\tilde{L}^T\tilde{L})^{-1}\cdot\tilde{L}^T\cdot b$, where $b = (\delta, c_1, c_2, \ldots, c_m)^T$

$Ax = b$

$A^TAx = A^Tb$

$x = (A^TA)^{-1}A^Tb$

✓ Derivation idea:

$$\underset{v'\in V}{\arg\min}\left\{\sum_{\{i,j\}\in E}\left\|(v'_j - v'_i) - \overset{e}{(v_j - v_i)}\right\|^2 + w\sum_{i\in C}\|v'_i - C_i\|^2\right\}$$

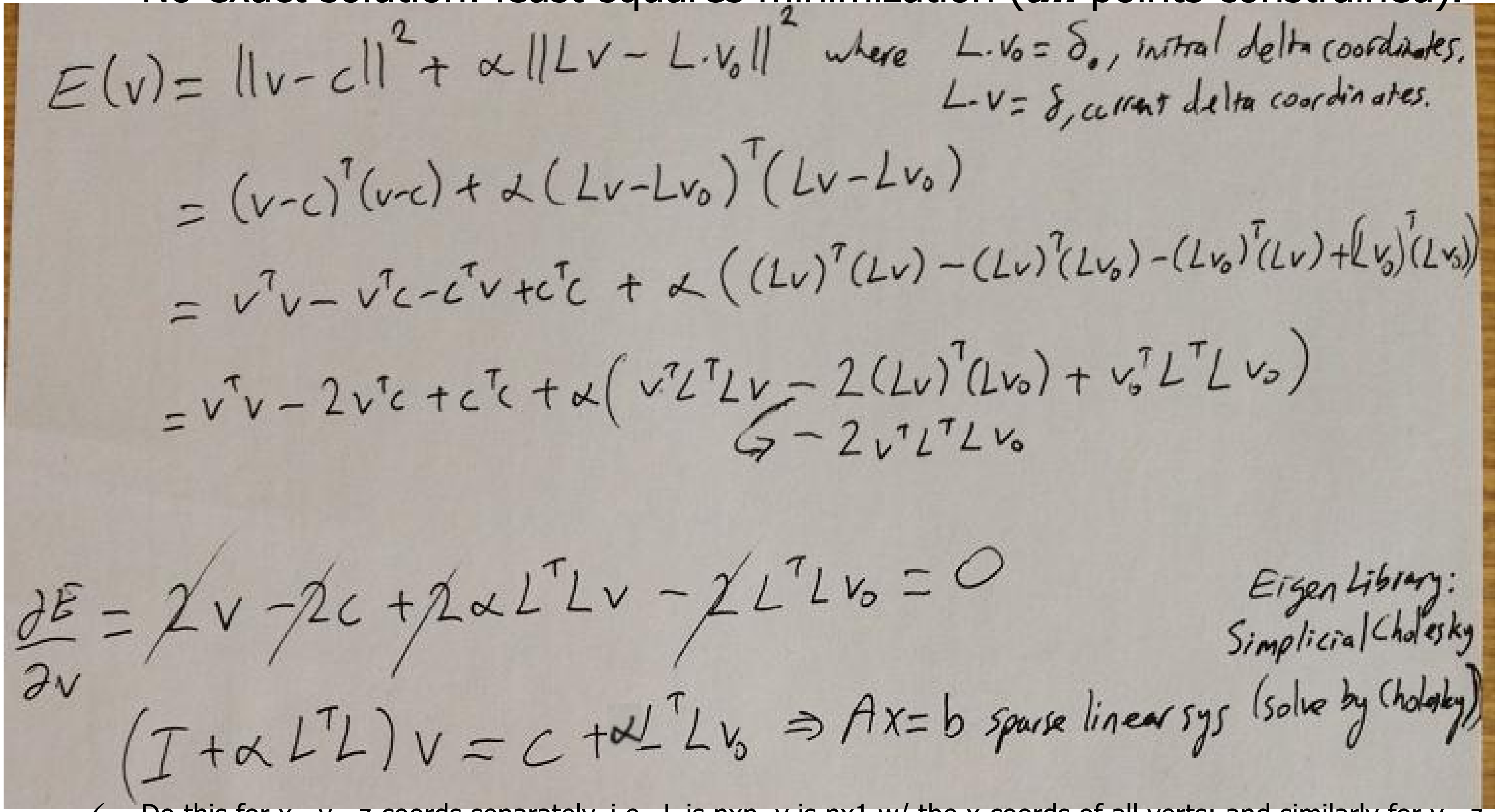
Least-squares minimization.

$$\underbrace{\left[\begin{array}{cccc} 1 & & & -1 \\ & & -1 & 1 \\ & & & \vdots \\ \hline & & & w \\ w & & & \\ & & & \vdots \end{array}\right]}_{\mathrm{A}} \underbrace{\begin{bmatrix} v'_{0x} \\ v'_{1x} \\ \vdots \end{bmatrix}}_{\mathbf{v}'_x} - \underbrace{\left[\begin{array}{c} e_{0x} \\ e_{1x} \\ \vdots \\ \hline wc_{0x} \\ wc_{1x} \\ \vdots \end{array}\right]}_{\mathbf{b}} \begin{array}{l} \left.\begin{array}{c} \\ \\ \\ \end{array}\right\} \text{edge vectors} \\ \left.\begin{array}{c} \\ \\ \\ \end{array}\right\} \text{constraints} \end{array}$$

$$\rightarrow \quad A^TA\mathbf{v}' = A^T\mathbf{b}$$

Differential Coordinates (Deformation)

✓ No exact solution: least-squares minimization (***all*** points constrained).



$$E(v) = \|v-c\|^2 + \alpha\|Lv - L\cdot v_0\|^2$$ where $L\cdot v_0 = \delta_0$, initial delta coordinates. $L\cdot v = \delta$, current delta coordinates.

$$= (v-c)^T(v-c) + \alpha(Lv-Lv_0)^T(Lv-Lv_0)$$

$$= v^Tv - v^Tc - c^Tv + c^Tc + \alpha((Lv)^T(Lv) - (Lv)^T(Lv_0) - (Lv_0)^T(Lv) + (Lv_0)^T(Lv_0))$$

$$= v^Tv - 2v^Tc + c^Tc + \alpha(v^TL^TLv \underbrace{- 2(Lv)^T(Lv_0)}_{-2v^TL^TLv_0} + v_0^TL^TLv_0)$$

$$\frac{\partial E}{\partial v} = \cancel{2}v - \cancel{2}c + \cancel{2}\alpha L^TLv - \cancel{2}L^TLv_0 = 0$$

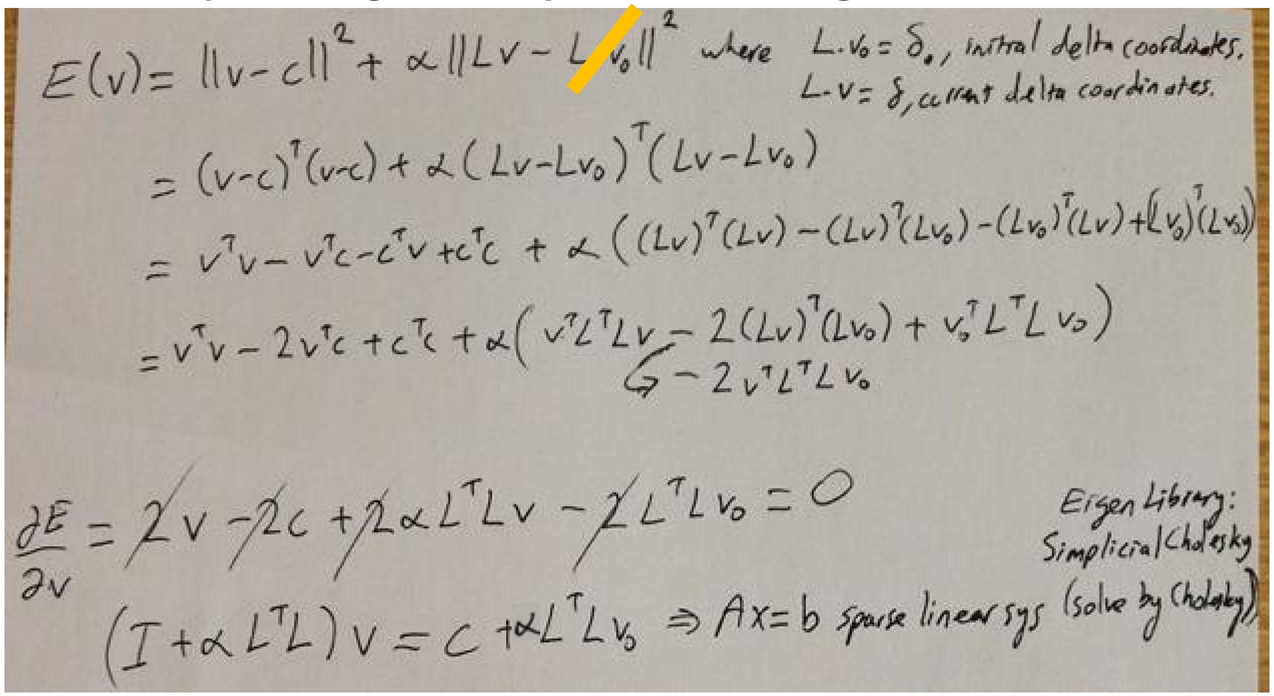
Eigen Library: SimplicialCholesky

$$(I + \alpha L^TL)v = c + \alpha L^TLv_0 \Rightarrow Ax=b$$ sparse linear sys (solve by Cholesky)

✓ Do this for x-, y-, z-coords separately, i.e., L is nxn, v is nx1 w/ the x-coords of all verts; and similarly for y-, z-.

Differential Coordinates (Def./Compression)

- ✓ No exact solution: least-squares minimization (***all*** points constrained).
- ✓ Putting **0** instead of initial delta coords ($L.v_0 = \delta_0$) is also OK, since then you will be forcing each vertex to the weighted centroid of its neighbors w/o depending on any initial triangulation/delta.



$$E(v) = \|v-c\|^2 + \alpha\|Lv - \cancel{Lv_0}\|^2$$ where $L.v_0 = \delta_0$, initial delta coordinates. $L.v = \delta$, current delta coordinates.

$$= (v-c)^T(v-c) + \alpha(Lv-Lv_0)^T(Lv-Lv_0)$$

$$= v^Tv - v^Tc - c^Tv + c^Tc + \alpha\left((Lv)^T(Lv) - (Lv)^T(Lv_0) - (Lv_0)^T(Lv) + (Lv_0)^T(Lv_0)\right)$$

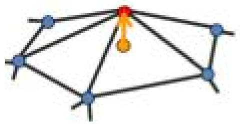
$$= v^Tv - 2v^Tc + c^Tc + \alpha\left(v^TL^TLv - 2(Lv)^T(Lv_0) + v_0^TL^TLv_0\right)$$

$$-2v^TL^TLv_0$$

$$\frac{\partial E}{\partial v} = \cancel{2}v - \cancel{2}c + \cancel{2}\alpha L^TLv - \cancel{2}L^TLv_0 = 0$$

$$(I + \alpha L^TL)v = c + \alpha L^TLv_0 \Rightarrow Ax = b$$ sparse linear sys

Eigen Library: SimplicialCholesky (solve by Cholesky)



$$\vec{\delta_i} = \vec{v_i} - \frac{1}{d_i}\sum_{j\in N(i)} \vec{v_j} = 0.$$

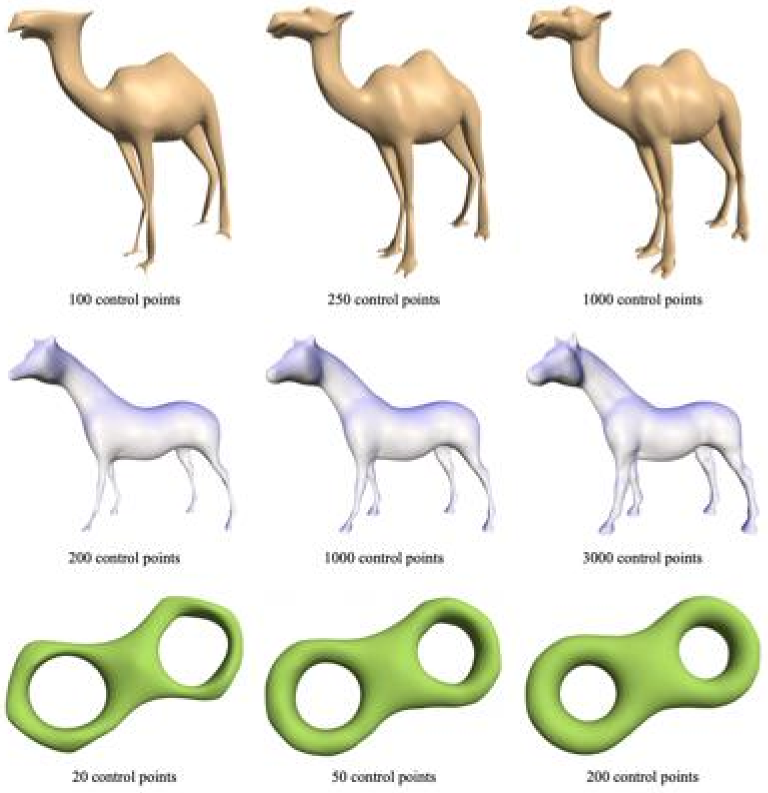
$$L_{ij} = \begin{cases} 1 & i = j \\ -\frac{1}{d_i} & (i,j) \in E \\ 0 & \text{otherwise.} \end{cases}$$

$$L\mathbf{x} = 0, \quad L\mathbf{y} = 0, \quad L\mathbf{z} = 0,$$

E_regu becomes alpha*||Lv - **0**||^2

Differential Coordinates (Def./Compression)

- ✓ No exact solution: least-squares minimization (***all*** points constrained).
- ✓ Putting **0** is not as detail-preserving as putting $L \cdot v_0 = \delta_0$ → not preferred.
- ✓ Putting **0** is, however, very good for mesh compression: The only thing stored is L and xyz positions of the few control vertices/points.
- ✓ Similarly, very good for progressive mesh transmission.



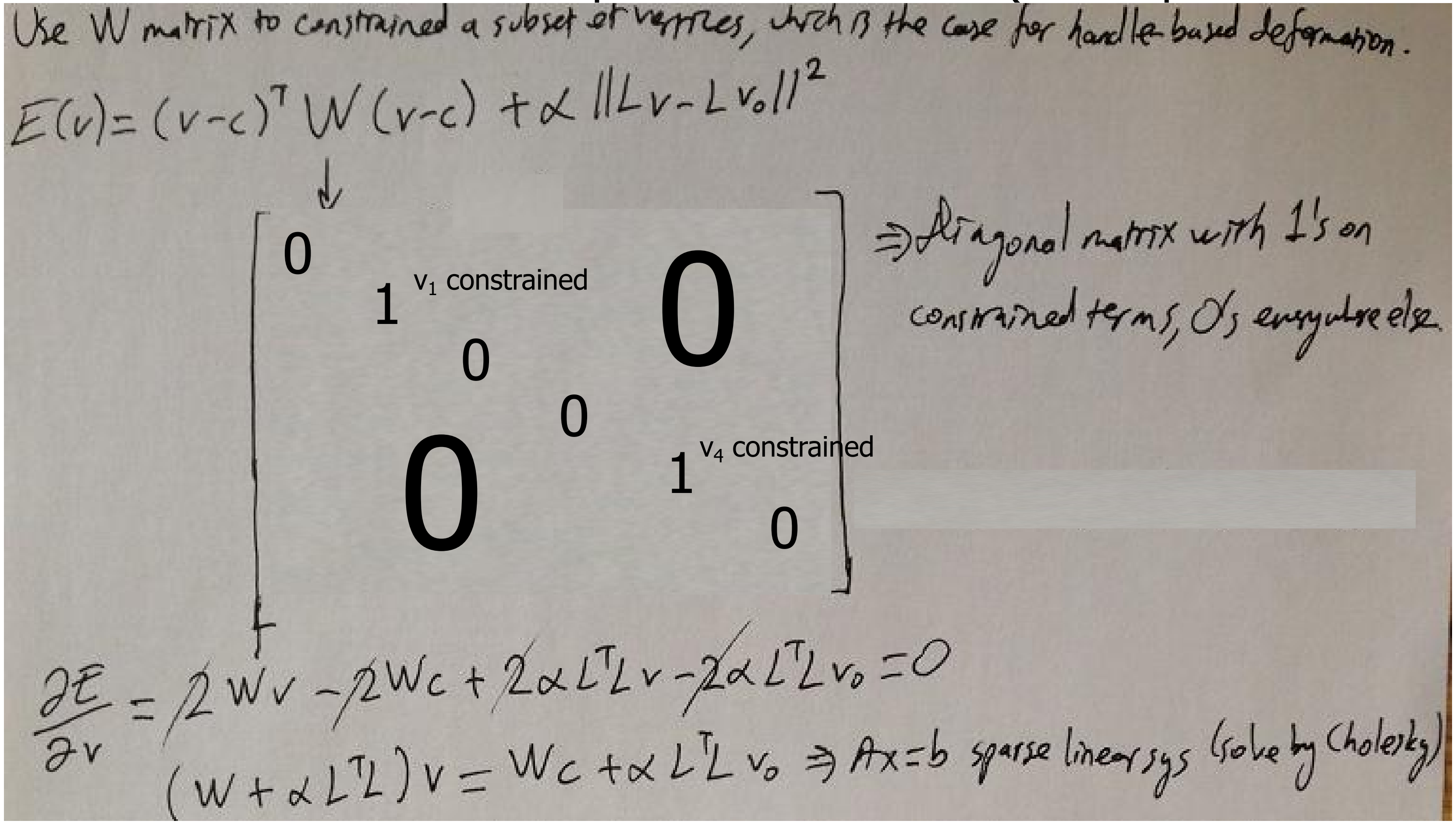
vertices: 39074

vertices: 19851

# vertices: 2718

Differential Coordinates (Deformation)

- ✓ No exact solution: least-squares minimization (***some*** pnts constrained).



Use W matrix to constrained a subset of vertices, which is the case for handle-based deformation.

$$E(v) = (v-c)^T W (v-c) + \alpha \|Lv - Lv_0\|^2$$

$$\downarrow$$

$$\begin{array}{c|l} \begin{bmatrix} 0 & & & & & \\ & 1 & & & \mathbf{0} & \\ & & 0 & & & \\ & & & 0 & & \\ & \mathbf{0} & & & 1 & \\ & & & & & 0 \end{bmatrix} & \begin{array}{l} \\ v_1 \text{ constrained} \\ \\ \\ v_4 \text{ constrained} \\ \\ \end{array} \end{array}$$

$\Rightarrow$ Diagonal matrix with 1's on constrained terms, 0's everywhere else.

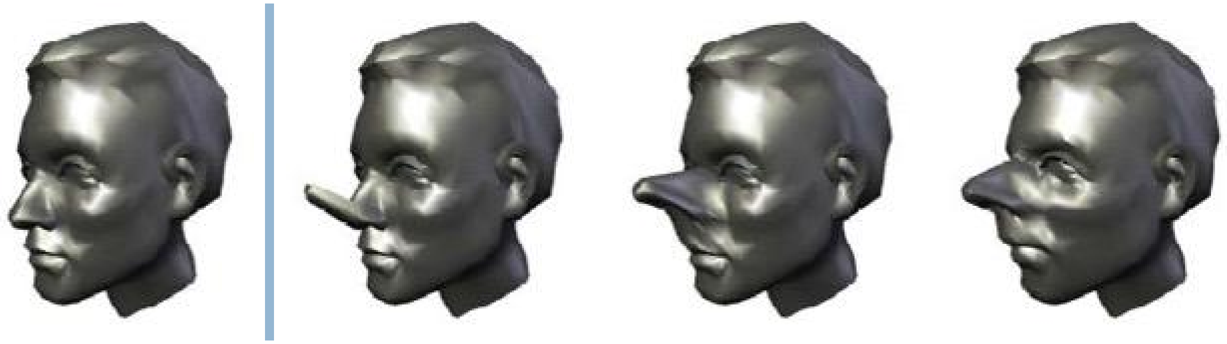
$$\frac{\partial E}{\partial v} = \cancel{2}Wv - \cancel{2}Wc + \cancel{2}\alpha L^T L v - \cancel{2}\alpha L^T L v_0 = 0$$

$$(W + \alpha L^T L)v = Wc + \alpha L^T L v_0 \Rightarrow Ax = b \text{ sparse linear sys (solve by Cholesky)}$$

- ✓ See Slide 95 for a version that does x-, y-, z-coordinates at once (not separately), i.e., L is 3n x 3n there.
- ✓ Do this for x-, y-, z-coords separately, i.e., L is nxn, v is nx1 w/ the x-coords of all verts; and similarly for y-, z-.

Differential Coordinates (Deformation)

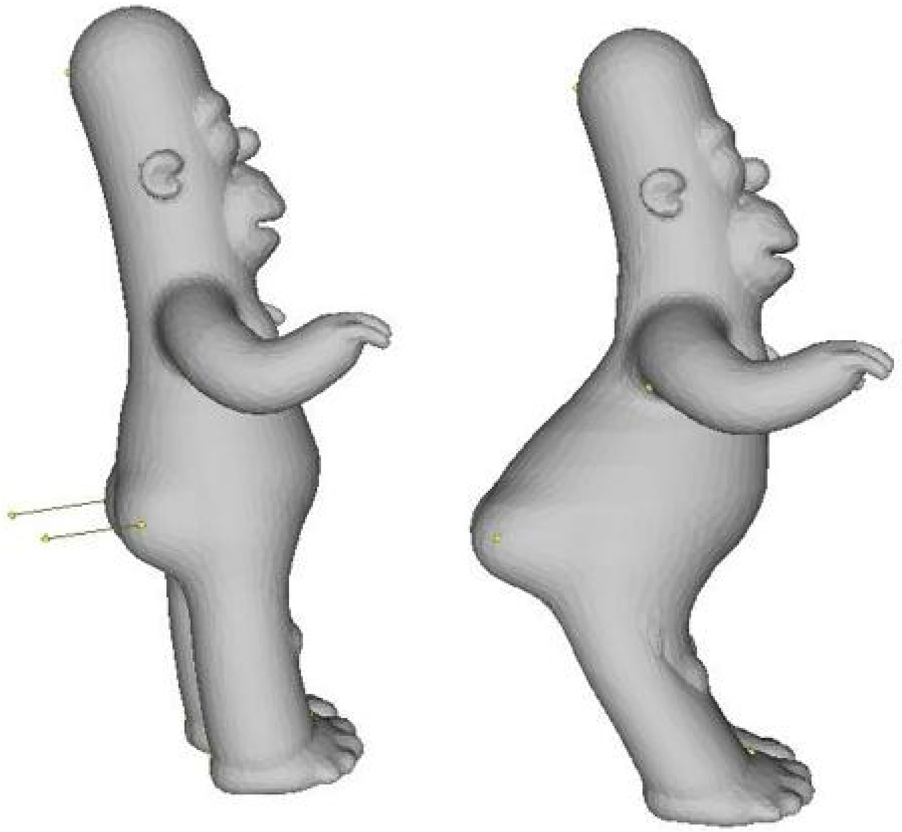
- ✓ Mesh deformation using Differential Coordinates.



- ✓ Regularization weight decreases as we go to left.

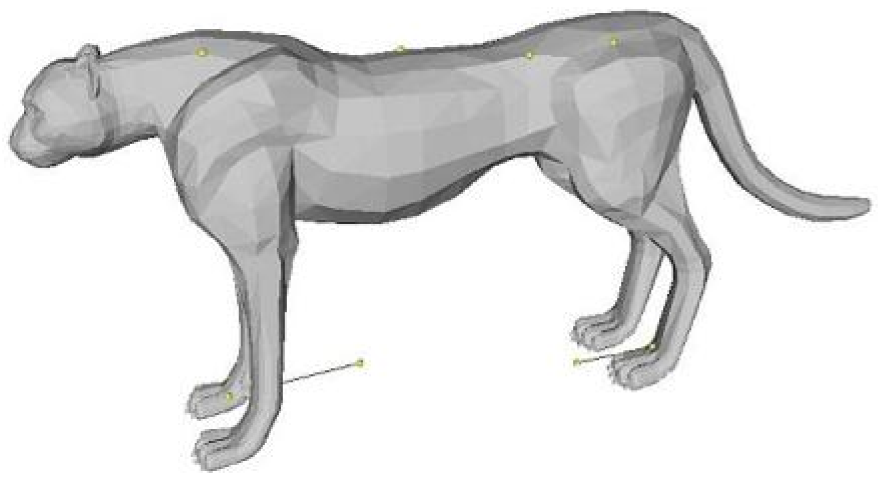
Differential Coordinates (Deformation)

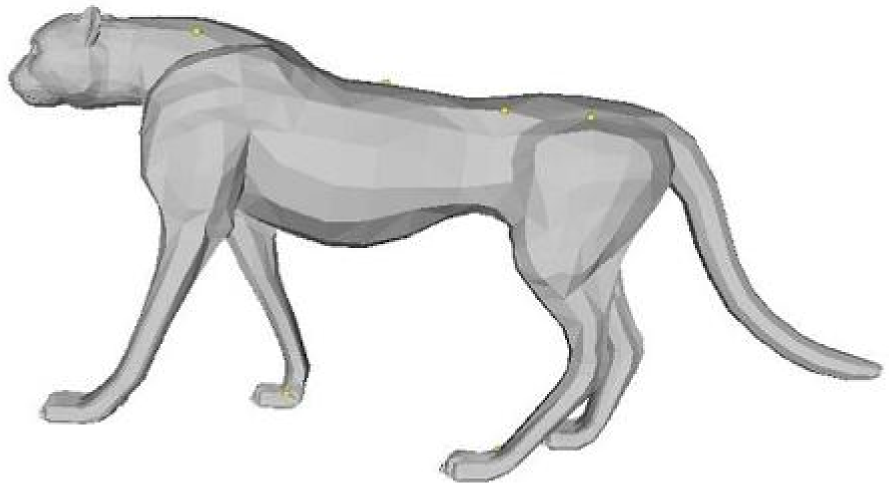
- ✓ Mesh deformation using Differential Coordinates.



Differential Coordinates (Deformation)

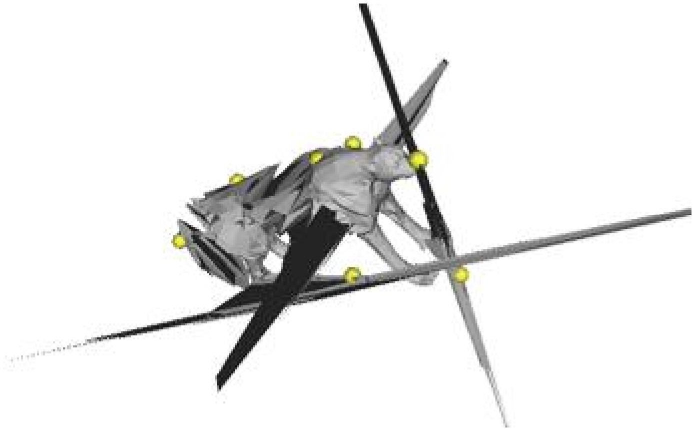
✓ Mesh deformation using Differential Coordinates.

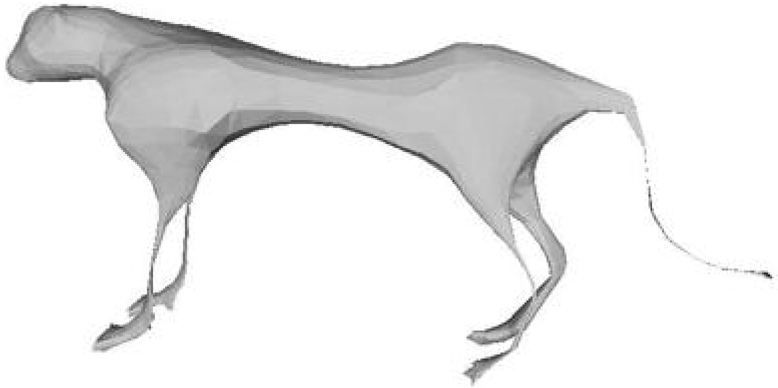


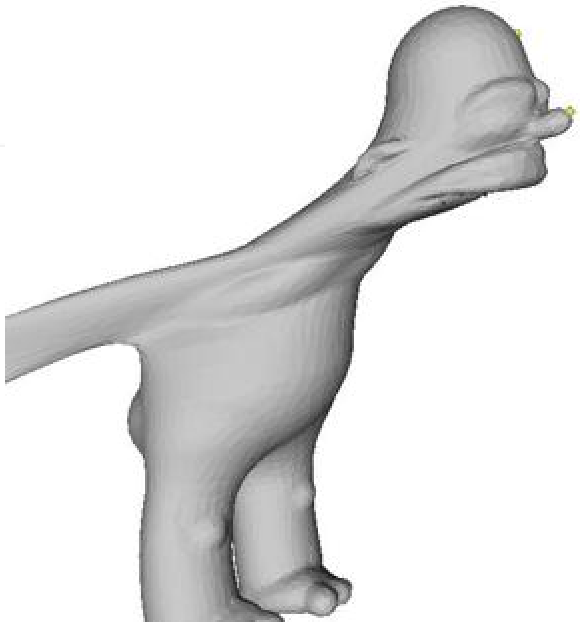


Differential Coordinates (Deformation)

- ✓ Mesh deformation using Differential Coordinates.
- ✓ You will (most likely) make weird errors along the way; don’t give up.

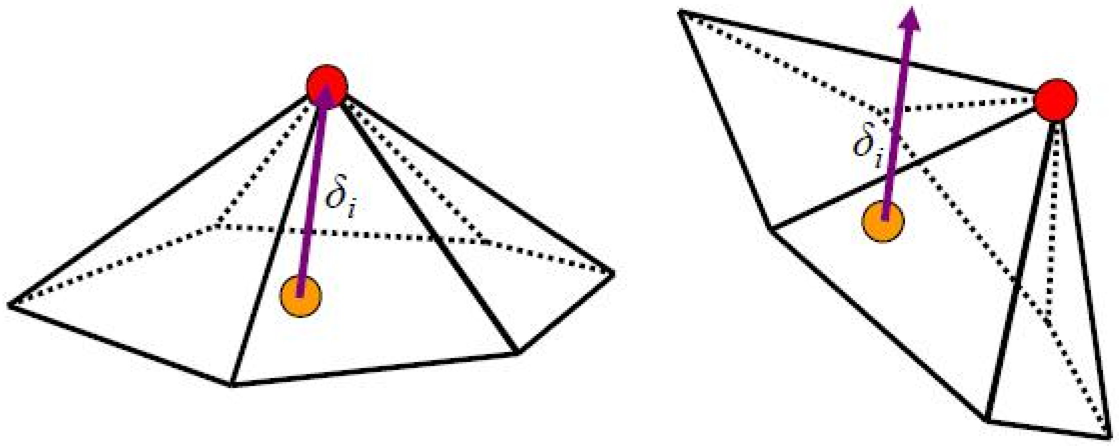






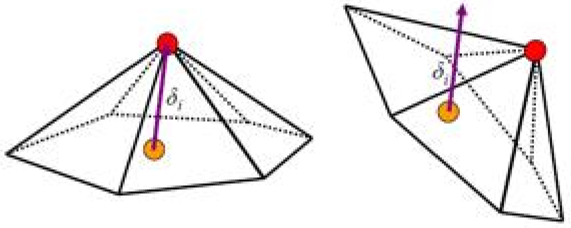
Differential Coordinates (Deformation)

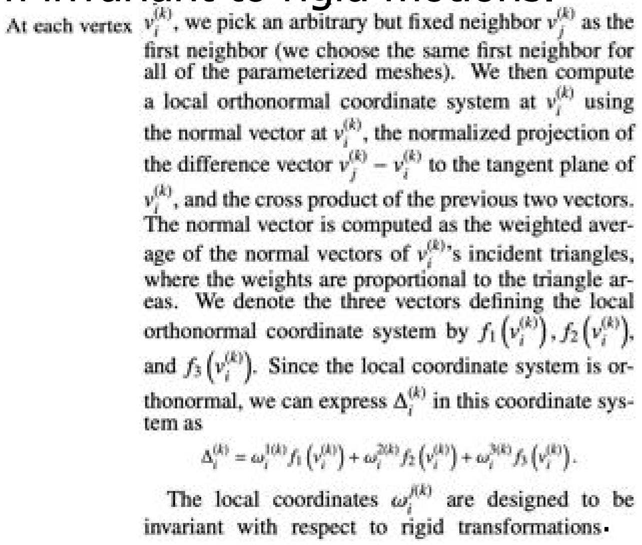
- ✓ Mesh deformation using Differential Coordinates.
- ✓ Good solution but does not support large deformations as it is not rotation-invariant ☹
  - ✓ Penalizes (=disallows) large rotations, hence large deformations.



Differential Coordinates (Deformation)

- ✓ Mesh deformation using Differential Coordinates.
- ✓ Good solution but does not support large deformations as it is not rotation-invariant ☹
  - ✓ Penalizes (=disallows) large rotations, hence large deformations.
  - ✓ Instead of preserving the initial diff. coords (i.e., minimize current - initial), set the current diff. coords to 0 (i.e., minimize current - 0)? //see Slide 61.
  - ✓ Another solution idea: express each diff. coord w.r.t. the local coord frame below, which makes them invariant to rigid motions.

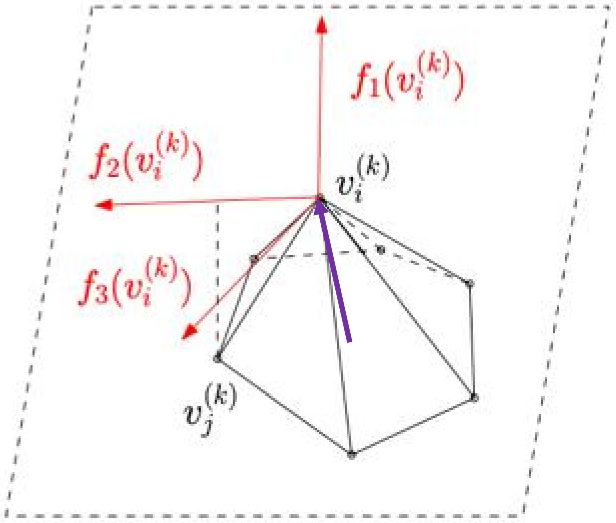




At each vertex $v_i^{(k)}$, we pick an arbitrary but fixed neighbor $v_j^{(k)}$ as the first neighbor (we choose the same first neighbor for all of the parameterized meshes). We then compute a local orthonormal coordinate system at $v_i^{(k)}$ using the normal vector at $v_i^{(k)}$, the normalized projection of the difference vector $v_j^{(k)} - v_i^{(k)}$ to the tangent plane of $v_i^{(k)}$, and the cross product of the previous two vectors. The normal vector is computed as the weighted average of the normal vectors of $v_i^{(k)}$'s incident triangles, where the weights are proportional to the triangle areas. We denote the three vectors defining the local orthonormal coordinate system by $f_1(v_i^{(k)})$, $f_2(v_i^{(k)})$, and $f_3(v_i^{(k)})$. Since the local coordinate system is orthonormal, we can express $\Delta_i^{(k)}$ in this coordinate system as

$$\Delta_i^{(k)} = \omega_i^{1(k)} f_1(v_i^{(k)}) + \omega_i^{2(k)} f_2(v_i^{(k)}) + \omega_i^{3(k)} f_3(v_i^{(k)}).$$

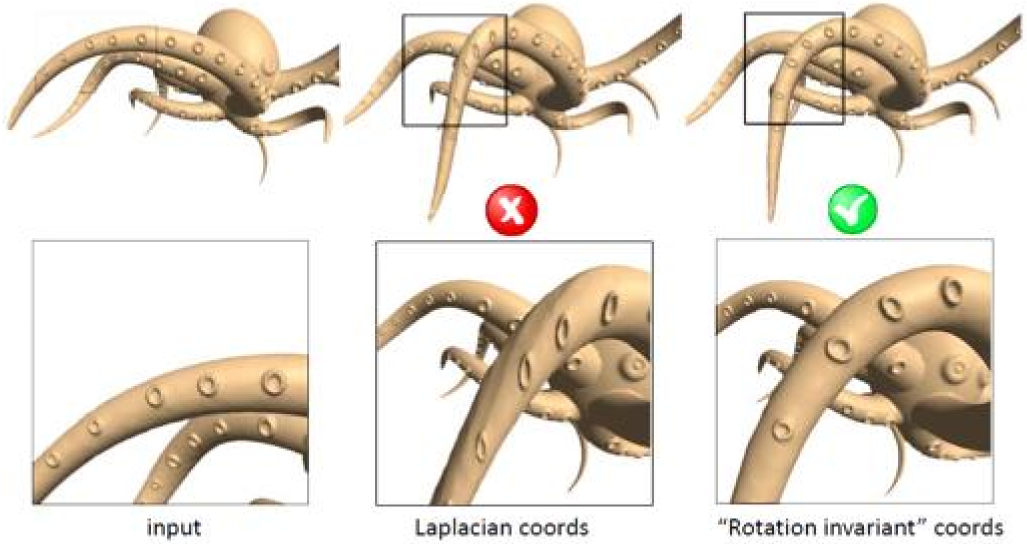
The local coordinates $\omega_i^{j(k)}$ are designed to be invariant with respect to rigid transformations.



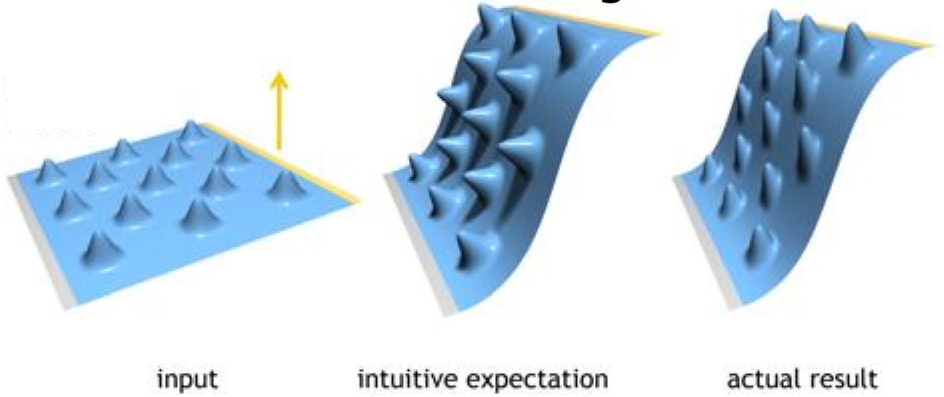
Posture-Invariant Statistical Shape Analysis Using Laplace Operator, 2012.

Differential Coordinates (Deformation)

- ✓ Mesh deformation using Differential Coordinates.



- ✓ Shearing artifact due to the following reason:

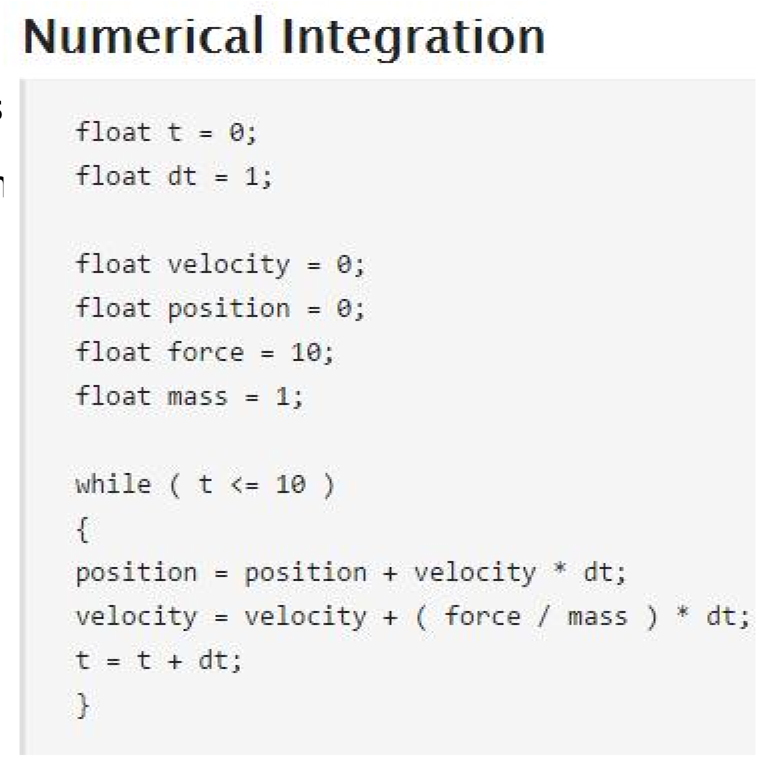


Physically-based Deformation

- ✓ Idea: given that the object is here and traveling this fast, in a short amount of time from now should be over there.
- ✓ Perform this prediction using a technique called Integration.
- ✓ Boils down to basic math (Newton’s laws).
  - ✓ $F = ma$
  - ✓ $a = F/m$ //the more an object weights, the less its acceleration from the same force
  - ✓ $dv / dt = a$ //rate of change in velocity = change in velocity per dt (= per 1 sec); $dv = a\ dt$
  - ✓ $v = v + dv$ //new velocity
  - ✓ $dx / dt = v$ //rate of change in position; $dx = v\ dt$
  - ✓ $x = x + dx$ //new position

Physically-based Deformation

- ✓ Idea: given that the object is here, and is traveling this fast, in a short amount of time from now should be over there.
- ✓ Perform this prediction using a technique called Integration.
- ✓ Boils down to high-school math (Newton's laws).
  - ✓ F = ma
  - ✓ a = F/m //the more an object weights, the less
  - ✓ dv / dt = a //rate of change in velocity = chan
  - ✓ v = v + dv //new velocity
  - ✓ dx / dt = v //rate of change in position
  - ✓ x = x + dx //new position



Numerical Integration

```
float t = 0;
float dt = 1;

float velocity = 0;
float position = 0;
float force = 10;
float mass = 1;

while ( t <= 10 )
{
position = position + velocity * dt;
velocity = velocity + ( force / mass ) * dt;
t = t + dt;
}
```

Physically-based Deformation

- ✓ Numerical integration: numerically integrate to find the solution, instead of solving differential equations.

- ✓ Start at a certain initial position and velocity.
- ✓ Take a small step forward in time to find the velocity and position at the next time value.
- ✓ Do this repeatedly to go forward in time in small increments, each time taking the results of the previous integration as the starting point for the next.

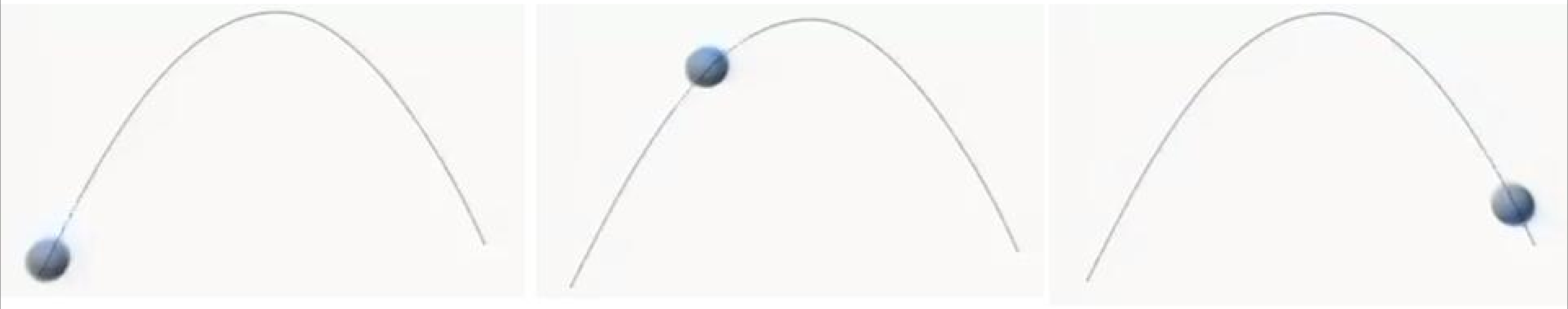
- ✓ aka Euler Integration.
- ✓ More sophisticated (and accurate) integrators (x = vt + at^2)
  - ✓ Semi-implicit and implicit Euler.
  - ✓ Runge Kutta order 4.

Physically-based Deformation

- ✓ Euler integration (Euler's method) with damping.
- ✓ Damping is an influence on an oscillatory system that has good effect of reducing, restricting, or preventing its oscillations.
  - ✓ With damping, e.g., a car stays steady/smooth on a rough road.

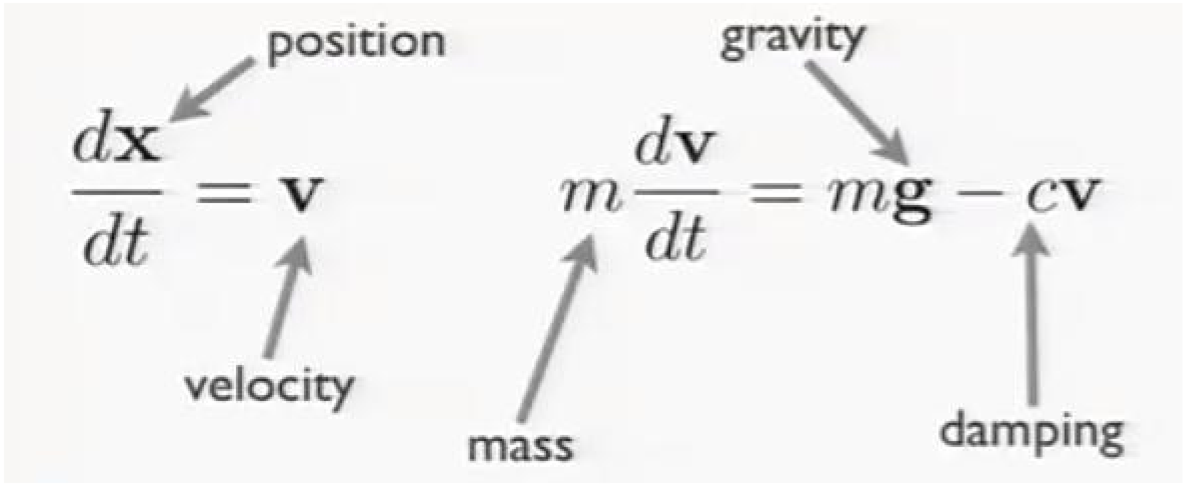
Physically-based Deformation

✓ Run a one-particle simulation using Newton’s laws & Euler’s method.



Physically-based Deformation

- ✓ Run a one-particle simulation using Newton's laws & Euler's method.



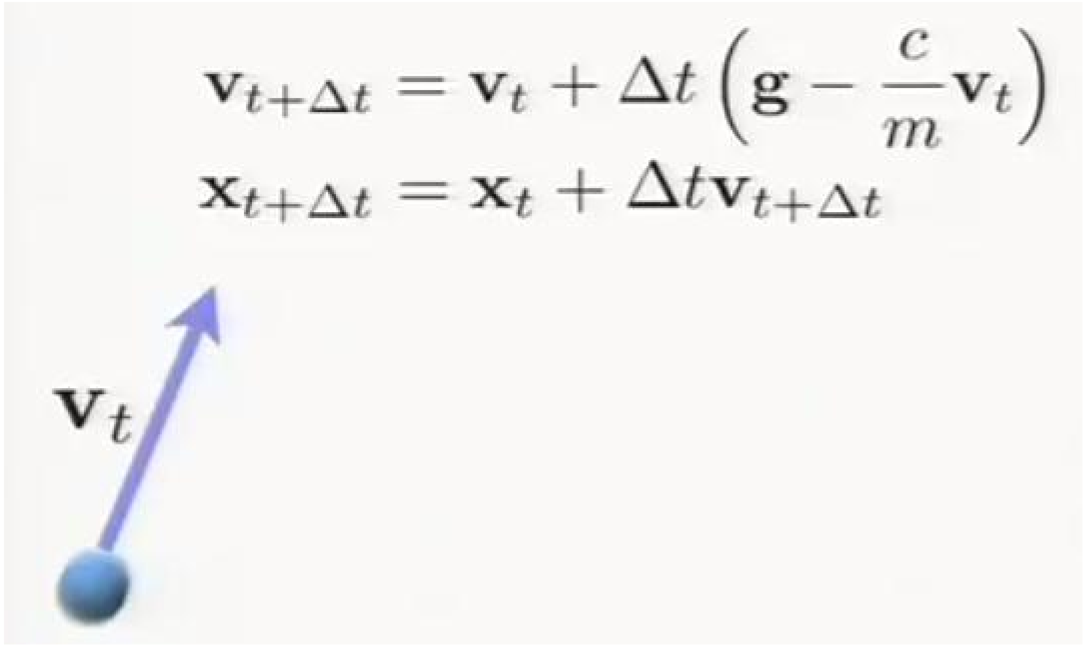
$$\frac{d\mathbf{x}}{dt} = \mathbf{v} \qquad m\frac{d\mathbf{v}}{dt} = m\mathbf{g} - c\mathbf{v}$$

- ✓ Derivative of position w.r.t. time is velocity ➔ gives the new position.
  - ✓ $\mathbf{x}_{t+\Delta t} = \mathbf{x}_t + \Delta t \mathbf{v}_{t+\Delta t}$
- ✓ Derivative of velocity w.r.t. time (acceleration) times mass are the forces that are applied to the system ➔ gives the new velocity.
  - ✓ $\mathbf{v}_{t+\Delta t} = \mathbf{v}_t + \Delta t \left(\mathbf{g} - \frac{c}{m}\mathbf{v}_t\right)$ Gravity force and a damping force which acts oppose velocity.

Physically-based Deformation

- ✓ Run a one-particle simulation using Newton's laws & Euler's method.

- ✓ Start with an initial position and velocity.



$$\mathbf{v}_{t+\Delta t} = \mathbf{v}_t + \Delta t \left(\mathbf{g} - \frac{c}{m} \mathbf{v}_t \right)$$

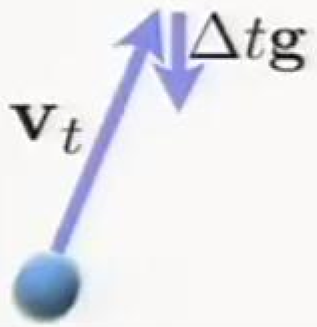
$$\mathbf{x}_{t+\Delta t} = \mathbf{x}_t + \Delta t \mathbf{v}_{t+\Delta t}$$

Physically-based Deformation

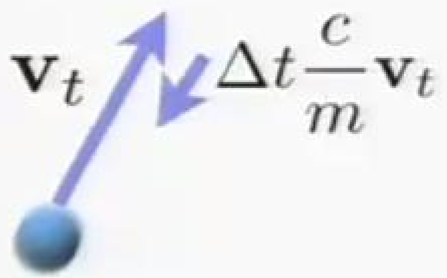
✓ Run a one-particle simulation using Newton's laws & Euler's method.

✓ Apply gravity (left) and damping (right) forces.

$$\mathbf{v}_{t+\Delta t} = \mathbf{v}_t + \Delta t\left(\mathbf{g} - \frac{c}{m}\mathbf{v}_t\right)$$
$$\mathbf{x}_{t+\Delta t} = \mathbf{x}_t + \Delta t\mathbf{v}_{t+\Delta t}$$



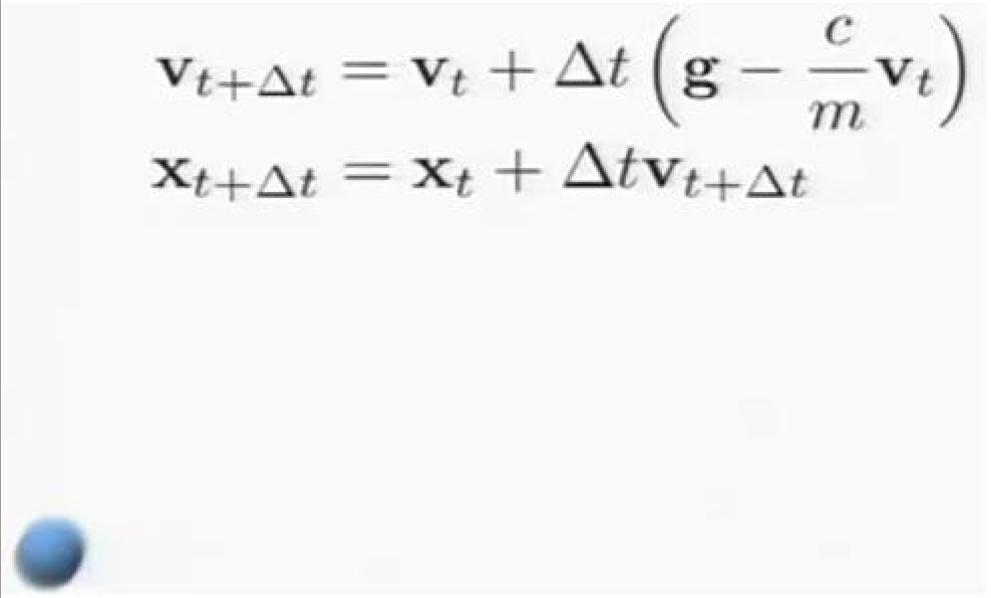
$$\mathbf{v}_{t+\Delta t} = \mathbf{v}_t + \Delta t\left(\mathbf{g} - \frac{c}{m}\mathbf{v}_t\right)$$
$$\mathbf{x}_{t+\Delta t} = \mathbf{x}_t + \Delta t\mathbf{v}_{t+\Delta t}$$



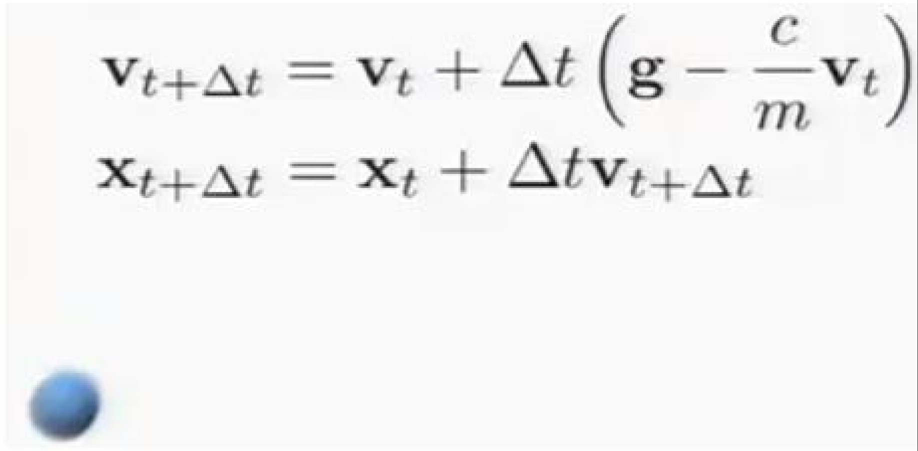
✓ Note that, damping is applied to the gravity-applied new v_t vector.

Physically-based Deformation

- ✓ Run a one-particle simulation using Newton’s laws & Euler’s method.
- ✓ Apply gravity (left) and damping (right) forces ➔ gives new velocity, which is used to time step the position.



$$\mathbf{v}_{t+\Delta t} = \mathbf{v}_t + \Delta t \left(\mathbf{g} - \frac{c}{m}\mathbf{v}_t \right)$$
$$\mathbf{x}_{t+\Delta t} = \mathbf{x}_t + \Delta t \mathbf{v}_{t+\Delta t}$$



$$\mathbf{v}_{t+\Delta t} = \mathbf{v}_t + \Delta t \left(\mathbf{g} - \frac{c}{m}\mathbf{v}_t \right)$$
$$\mathbf{x}_{t+\Delta t} = \mathbf{x}_t + \Delta t \mathbf{v}_{t+\Delta t}$$

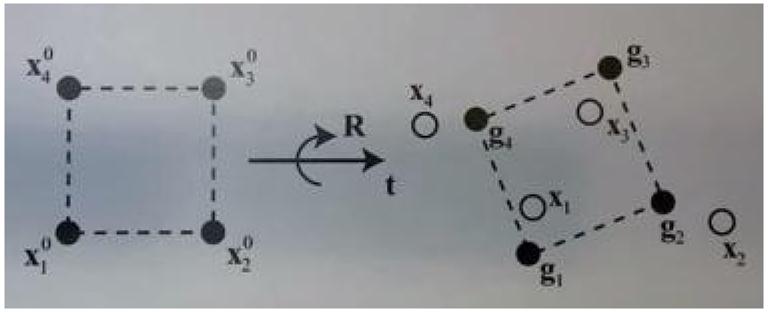
OLD NEW

Physically-based Deformation

- ✓ A related paper: Backward steps in rigid body simulation.
- ✓ A related paper: Meshless deformation based on shape matching.

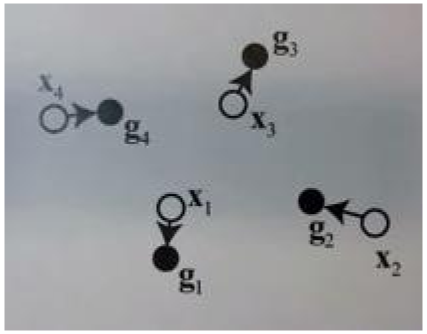
Physically-based Deformation

- ✓ A related paper: Meshless deformation based on shape matching.
- ✓ Idea:
  - ✓ Simulation (gravity force, etc.) or user interaction brings the initial good shape (left) to some arbitrary position (empty circles).
  - ✓ Find the rigid transformation that brings initial pose to this arbitrary pose as good as possible in the least square-sense (see Registration slides).



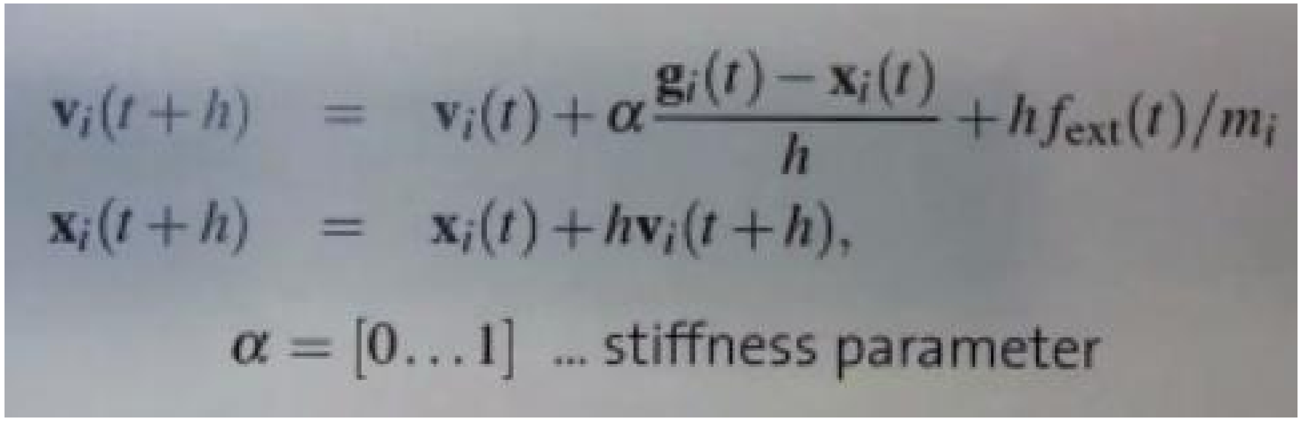
Physically-based Deformation

- ✓ A related paper: Meshless deformation based on shape matching.
- ✓ Idea:
  - ✓ Simulation (gravity force, etc.) or user interaction brings the initial good shape (left) to some arbitrary position (empty circles).
  - ✓ Find the rigid transformation that brings initial pose to this arbitrary pose as good as possible in the least square-sense (see Registration slides).
  - ✓ Attract points to these goal positions g.
    - ✓ Like a regularization term (emulate internl forces preserving initial good shape).



Physically-based Deformation

- ✓ A related paper: Meshless deformation based on shape matching.
- ✓ Idea:
  - ✓ Simulation (gravity force, etc.) or user interaction brings the initial good shape (left) to some arbitrary position (empty circles).
  - ✓ Find the rigid transformation that brings initial pose to this arbitrary pose as good as possible in the least square-sense (see Registration slides).
  - ✓ Attract points to these goal positions g.
  - ✓ Numerical integration, or time-stepping:



$$\mathbf{v}_i(t+h) = \mathbf{v}_i(t) + \alpha \frac{\mathbf{g}_i(t) - \mathbf{x}_i(t)}{h} + h f_{\text{ext}}(t)/m_i$$

$$\mathbf{x}_i(t+h) = \mathbf{x}_i(t) + h\mathbf{v}_i(t+h),$$

$\alpha = [0\ldots1]$ ... stiffness parameter

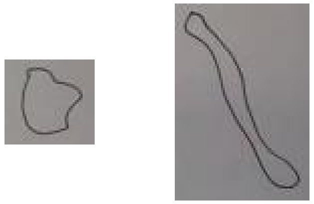
Deformation Energies

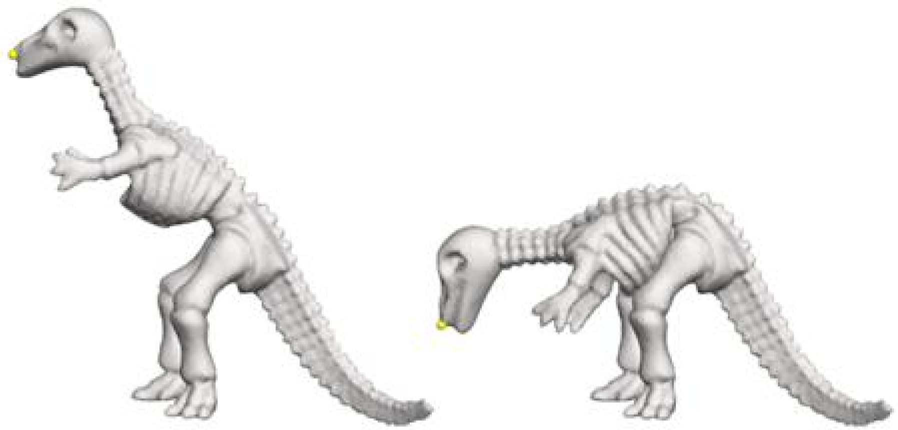
- ✓ Keep the mesh in good shape after all the arbitrary interactions.
- ✓ Like the second term in velocity in the previous slide but now let’s look at the problem in terms of deformation energies.
- ✓ Already done this while minimizing Laplacian energy in Slide 60 (differential coordinates). Let’s revisit in more detail.

$$E(v) = \|v - c\|^2 + \alpha \|Lv - L \cdot v_0\|^2$$

Deformation Energies

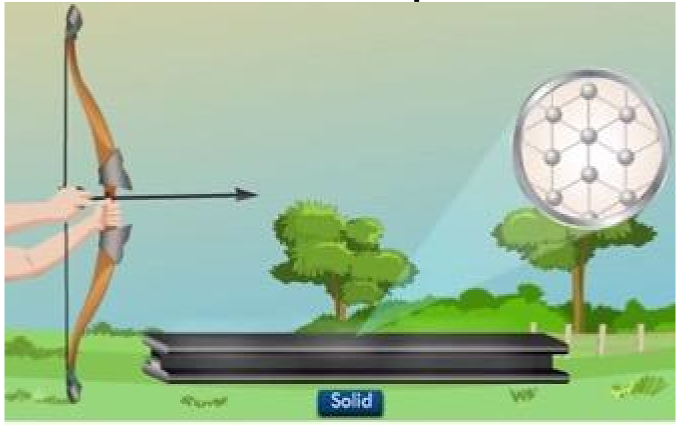
- ✓ What is a deformation energy?
  - ✓ A scalar that quantifies the amount of deformation from the initial pose.
  - ✓ = that tells how much deformed the new thing is.
    - ✓ 0 if no deformation at all (rotation-translation not count as def).

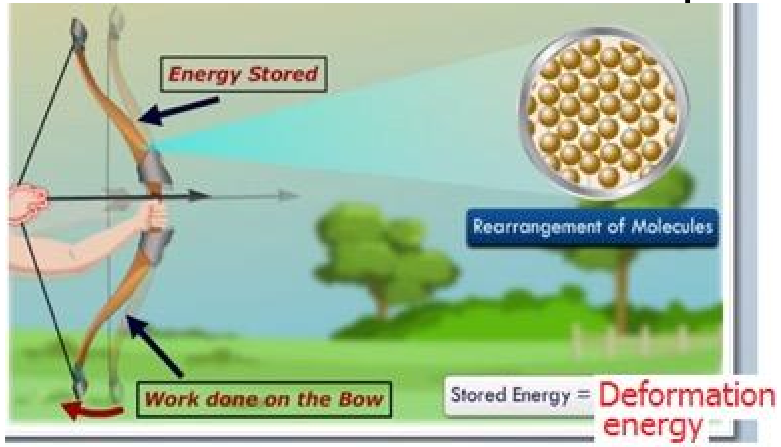


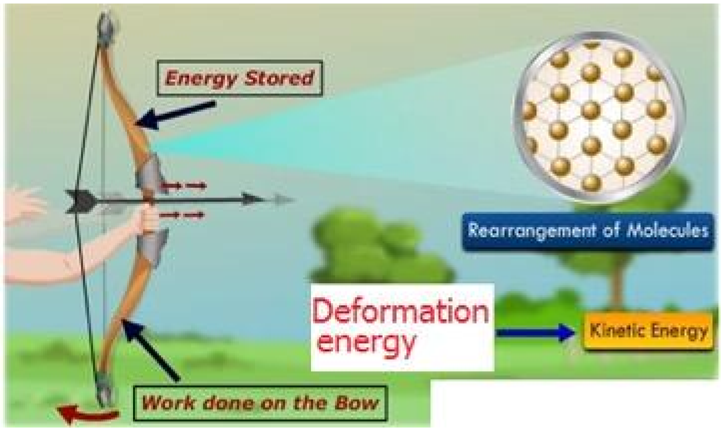


Deformation Energies

- ✓ What is a deformation energy?
  - ✓ A scalar that quantifies the amount of deformation from initial pose.




Energy Stored
Rearrangement of Molecules
Work done on the Bow
Stored Energy = Deformation energy



Deformation Energies

- ✓ Minimize deformation energy to keep the new mesh as close as possible to the initial mesh.
  - ✓ Without any other term, not interesting: just freeze the mesh.
  - ✓ With matching term (external force, user interaction, ..), keeps the new mesh in good shape (assuming initial mesh is already good).

Deformation Energies

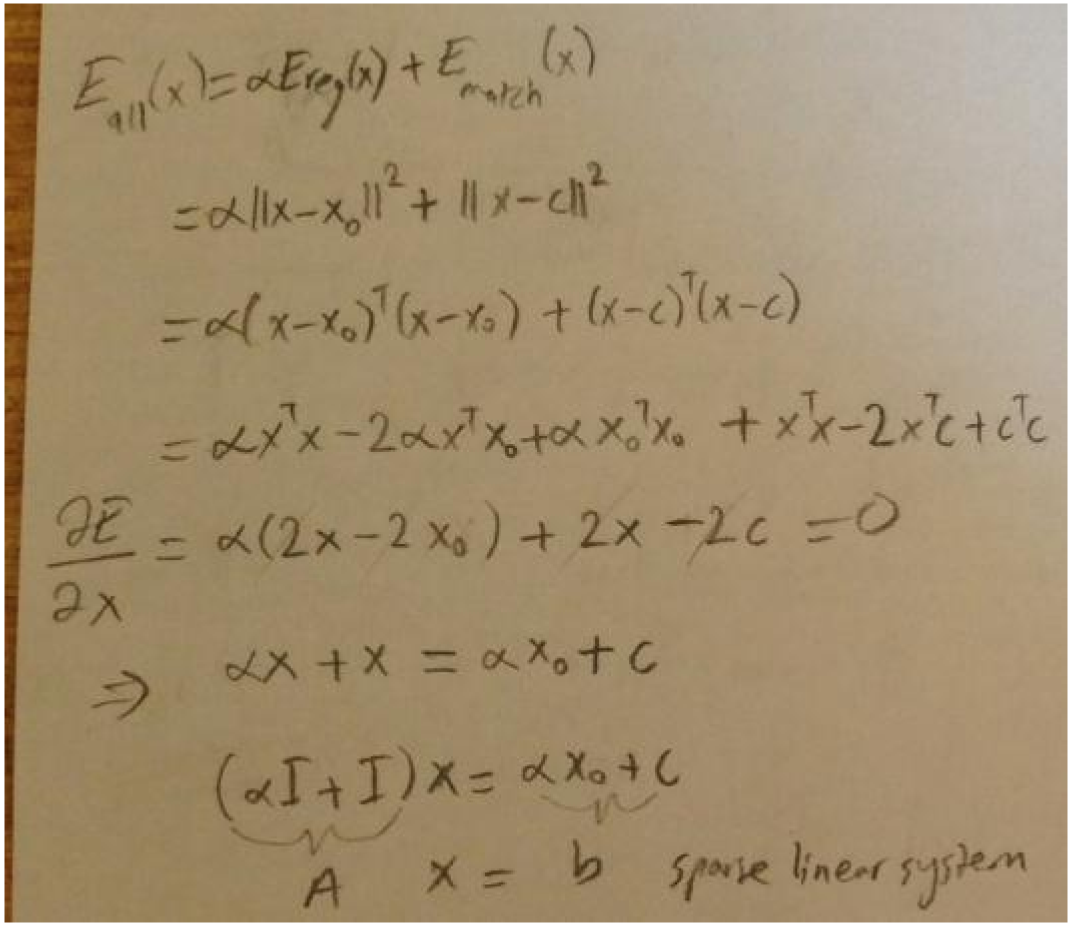
- ✓ Dead simple deformation energy.
- ✓ Measures difference from the initial Absolute Coordinates.
- ✓ Measures difference from the previous Absolute Coordinates (a variant called Tikhonov regularization).

- ✓ E_reg = $||x – x0||^2$ //deviation of x from initial Abs. Coordinates x0
- ✓ E_all(x) = E_reg(x) + E_match(x) //more interesting w/ E_m=$||x\text{-}c||^2$

- ✓ From now on, x is a 3n x 1 column vector where n is the number of mesh vertices, and 3 elements are stored for each vertex, namely the x-, y-, and z-coordinates. Similarly, x0 is the rest-pose of the mesh.

Deformation Energies

✓ Measures difference from the initial Absolute Coordinates.



$$E_{all}(x) = \alpha E_{reg}(x) + E_{match}(x)$$

$$= \alpha \|x - x_0\|^2 + \|x - c\|^2$$

$$= \alpha (x - x_0)^T (x - x_0) + (x - c)^T (x - c)$$

$$= \alpha x^T x - 2\alpha x^T x_0 + \alpha x_0^T x_0 + x^T x - 2x^T c + c^T c$$

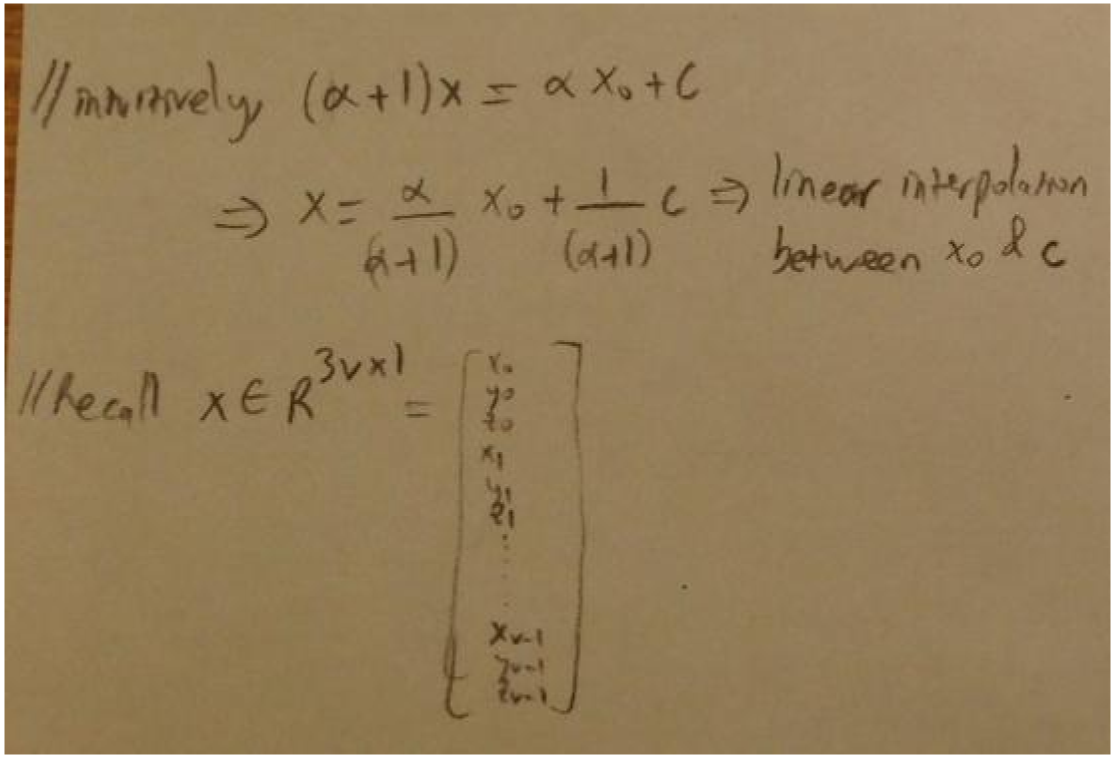
$$\frac{\partial E}{\partial x} = \alpha (2x - 2x_0) + 2x - 2c = 0$$

$$\Rightarrow \quad \alpha x + x = \alpha x_0 + c$$

$$\underbrace{(\alpha I + I)}_{A} \; \underset{x}{x} = \underbrace{\alpha x_0 + c}_{b} \quad \text{sparse linear system}$$

Deformation Energies

✓ Measures difference from the initial Absolute Coordinates.



//intuitively $(\alpha+1)x = \alpha x_0 + c$

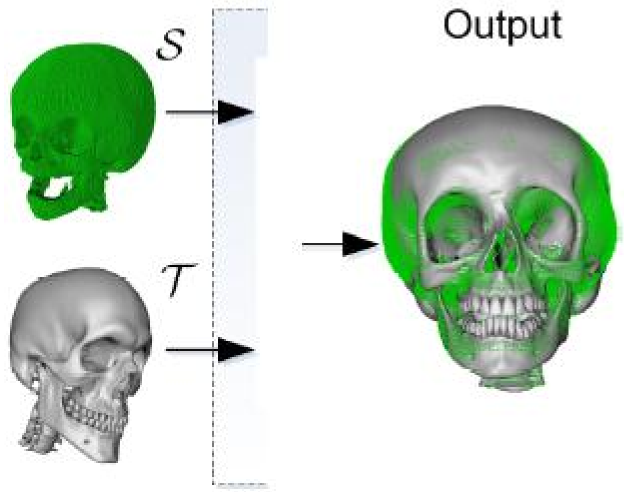
$\Rightarrow x = \frac{\alpha}{(\alpha+1)} x_0 + \frac{1}{(\alpha+1)} c \Rightarrow$ linear interpolation between x_0 & c

//Recall $x \in R^{3v \times 1} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ x_1 \\ y_1 \\ z_1 \\ \vdots \\ x_{v-1} \\ y_{v-1} \\ z_{v-1} \end{bmatrix}$

Deformation Energies

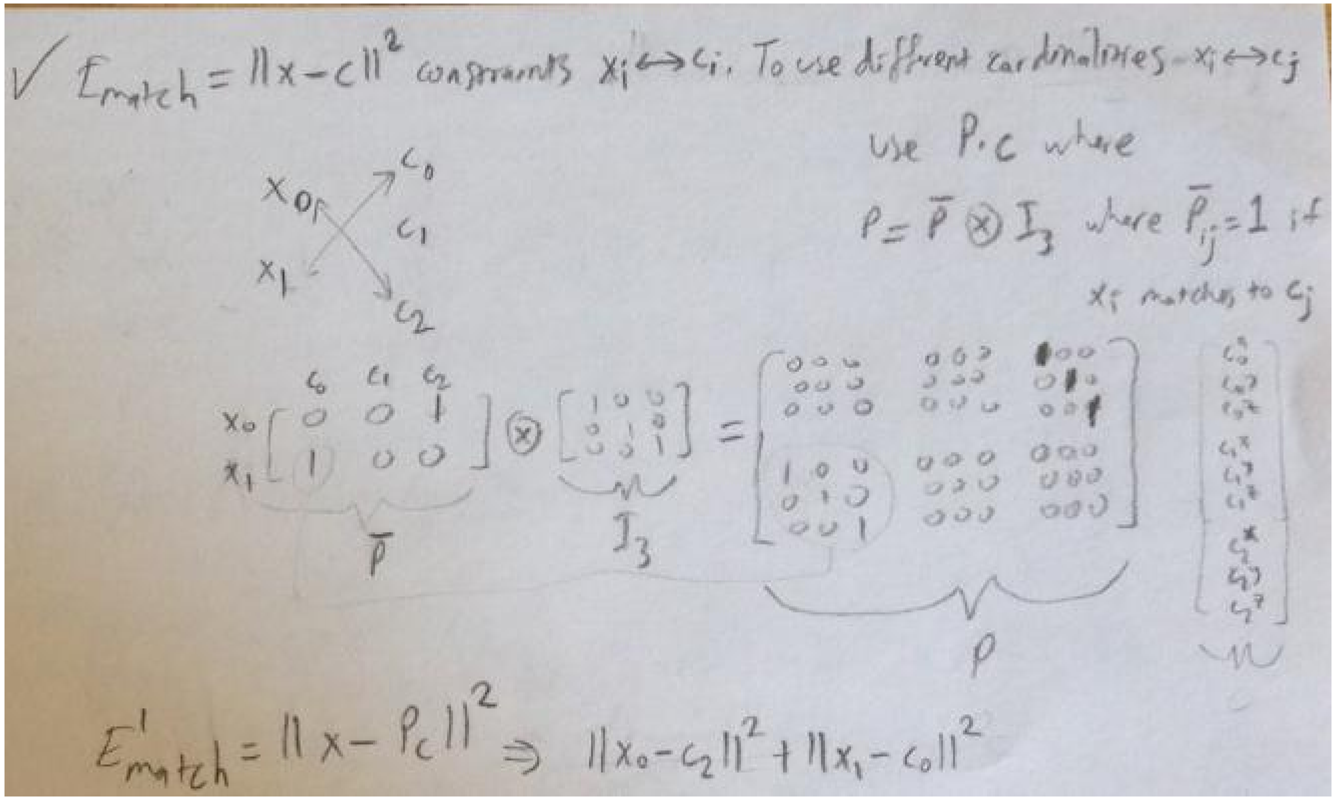
✓ We have a simple xi → ci support so far.

✓ Enable xi → cj correspondences, which may come handy when registering one shape (gray) into another one (green), with different # of vertices.



Deformation Energies

- ✓ Enable xi → cj correspondences.



✓ $E_{match} = ||x - c||^2$ constraints $x_i \leftrightarrow c_i$. To use different cardinalities $x_i \leftrightarrow c_j$ use $P \cdot c$ where

$P = \bar{P} \otimes I_3$ where $\bar{P}_{ij} = 1$ if x_i matches to c_j

$$\begin{tikzcd} x_0 \arrow[drr] & & c_0 \\ & & c_1 \\ x_1 \arrow[uurr] & & c_2 \end{tikzcd}$$

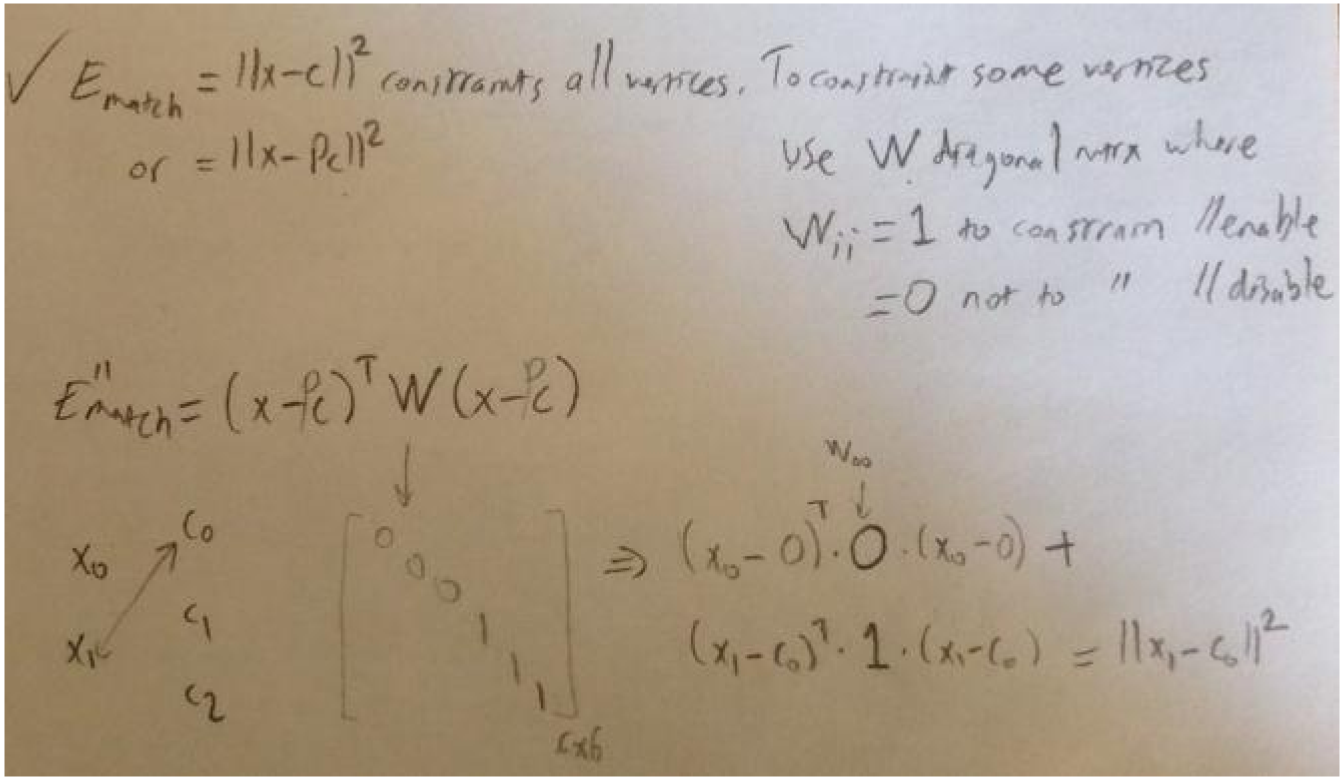
$$\underbrace{\begin{array}{c|ccc} & c_0 & c_1 & c_2 \\ \hline x_0 & 0 & 0 & 1 \\ x_1 & 1 & 0 & 0 \end{array}}_{\bar{P}} \otimes \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{I_3} = \underbrace{\begin{bmatrix} 0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&1 \\ 1&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0 \end{bmatrix}}_{P} \underbrace{\begin{bmatrix} c_0^x \\ c_0^y \\ c_0^z \\ c_1^x \\ c_1^y \\ c_1^z \\ c_2^x \\ c_2^y \\ c_2^z \end{bmatrix}}_{c}$$

$E'_{match} = ||x - P c||^2 \Rightarrow ||x_0 - c_2||^2 + ||x_1 - c_0||^2$

- ✓ The purpose of the Kronecker product is to be able to select 3 elements, namely x-, y-, z-coordinates, of the corresponding point from c.

Deformation Energies

- ✓ We constraint all vertices xi so far (via xi → ci or xi → cj).
- ✓ We may want to constraint some vertices, e.g., only handle vertices for interactive deformation.



✓ $E_{match} = \|x-c\|^2$ constraints all vertices. To constraint some vertices

or $= \|x - P_c\|^2$

use W diagonal matrix where

$W_{ii} = 1$ to constrain //enable

$= 0$ not to " //disable

$$E''_{match} = (x - P_c)^T W (x - P_c)$$

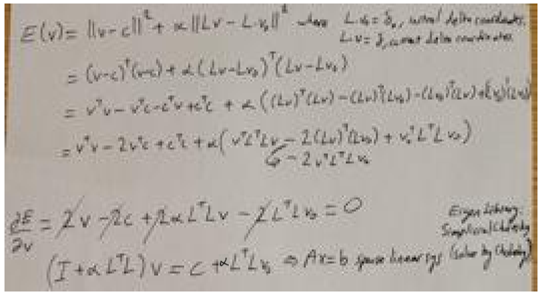
$$\begin{tikzcd} x_0 & c_0 \\ x_1 \arrow[ur] & c_1 \\ & c_2 \end{tikzcd}$$

$$\begin{bmatrix} 0 & & & & & \\ & 0 & & & & \\ & & 0 & & & \\ & & & 1 & & \\ & & & & 1 & \\ & & & & & 1 \end{bmatrix}_{6\times 6} \Rightarrow (x_0 - 0)^T \cdot \overset{W_{00}}{0} \cdot (x_0 - 0) +$$

$$(x_1 - c_0)^T \cdot 1 \cdot (x_1 - c_0) = \|x_1 - c_0\|^2$$

Deformation Energies

- ✓ A better deformation energy: Measure difference from the initial Differential Coordinates. (note that Slide 57 has an analytical soln for this minimztn)
- ✓ All vertices are constrained.
- ✓ Note, L here is the Kronecker extension (w/ I_3) of nxn L we introduced before (Slide 45), hence it is 3n x 3n.
- ✓ Slide 60 uses nxn L to solve for x-, y-, z-coords separately. It also lacks P matrix.



$$E(v) = \|v-c\|^2 + \alpha\|Lv - Lv_0\|^2$$ where $Lv_0 = \delta_0$, initial delta coordinates, $Lv = \delta$, current delta coordinates

$$= (v-c)^T(v-c) + \alpha(Lv-Lv_0)^T(Lv-Lv_0)$$

$$= v^Tv - v^Tc - c^Tv + c^Tc + \alpha((Lv)^T(Lv) - (Lv)^T(Lv_0) - (Lv_0)^T(Lv) + (Lv_0)^T(Lv_0))$$

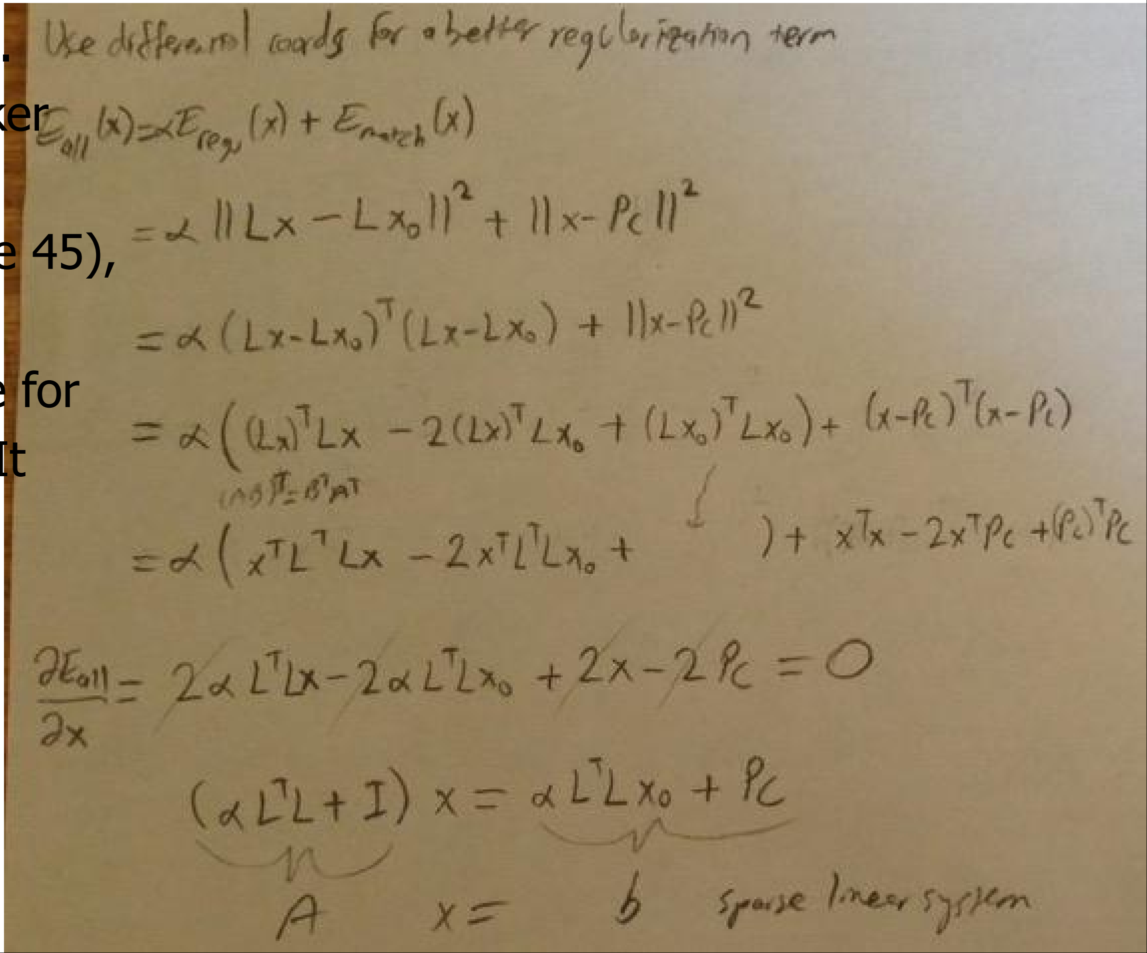
$$= v^Tv - 2v^Tc + c^Tc + \alpha(v^TL^TLv - 2(Lv)^T(Lv_0) + v_0^TL^TLv_0)$$

$$\hookrightarrow -2v^TL^TLv_0$$

$$\frac{\partial E}{\partial v} = \cancel{2}v - \cancel{2}c + \cancel{2}\alpha L^TLv - \cancel{2}L^TLv_0 = 0$$

$$(I + \alpha L^TL)v = c + \alpha L^TLv_0 \Rightarrow Ax=b$$ sparse linear sys

Eigen Library: SimplicialCholesky (Solve by Cholesky)



Use differential coords for a better regularization term

$$E_{all}(x) = \alpha E_{reg}(x) + E_{match}(x)$$

$$= \alpha\|Lx - Lx_0\|^2 + \|x - P_c\|^2$$

$$= \alpha(Lx - Lx_0)^T(Lx - Lx_0) + \|x - P_c\|^2$$

$$= \alpha\left((Lx)^TLx - 2(Lx)^TLx_0 + (Lx_0)^TLx_0\right) + (x-P_c)^T(x-P_c)$$

$(AB)^T = B^TA^T$

$$= \alpha\left(x^TL^TLx - 2x^TL^TLx_0 + \quad\right) + x^Tx - 2x^TP_c + (P_c)^TP_c$$

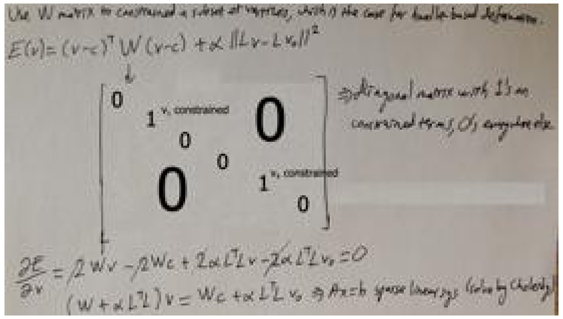
$$\frac{\partial E_{all}}{\partial x} = 2\alpha L^TLx - 2\alpha L^TLx_0 + 2x - 2P_c = 0$$

$$\underbrace{(\alpha L^TL + I)}_{A}\, x = \underbrace{\alpha L^TLx_0 + P_c}_{b}$$

$A \quad x = \quad b$ sparse linear system

Deformation Energi

- ✓ Some vertices are constrained.
- ✓ Note, L here is the Kronecker extension (w/ I_3) of nxn L we introduced before (Slide 45 hence it is 3n x 3n.
- ✓ Slide 63 uses nxn L to solve for x-, y-, z-coords separately.



Use W matrix to constrained a subset of vertices, which is the case for handle-based deformation.

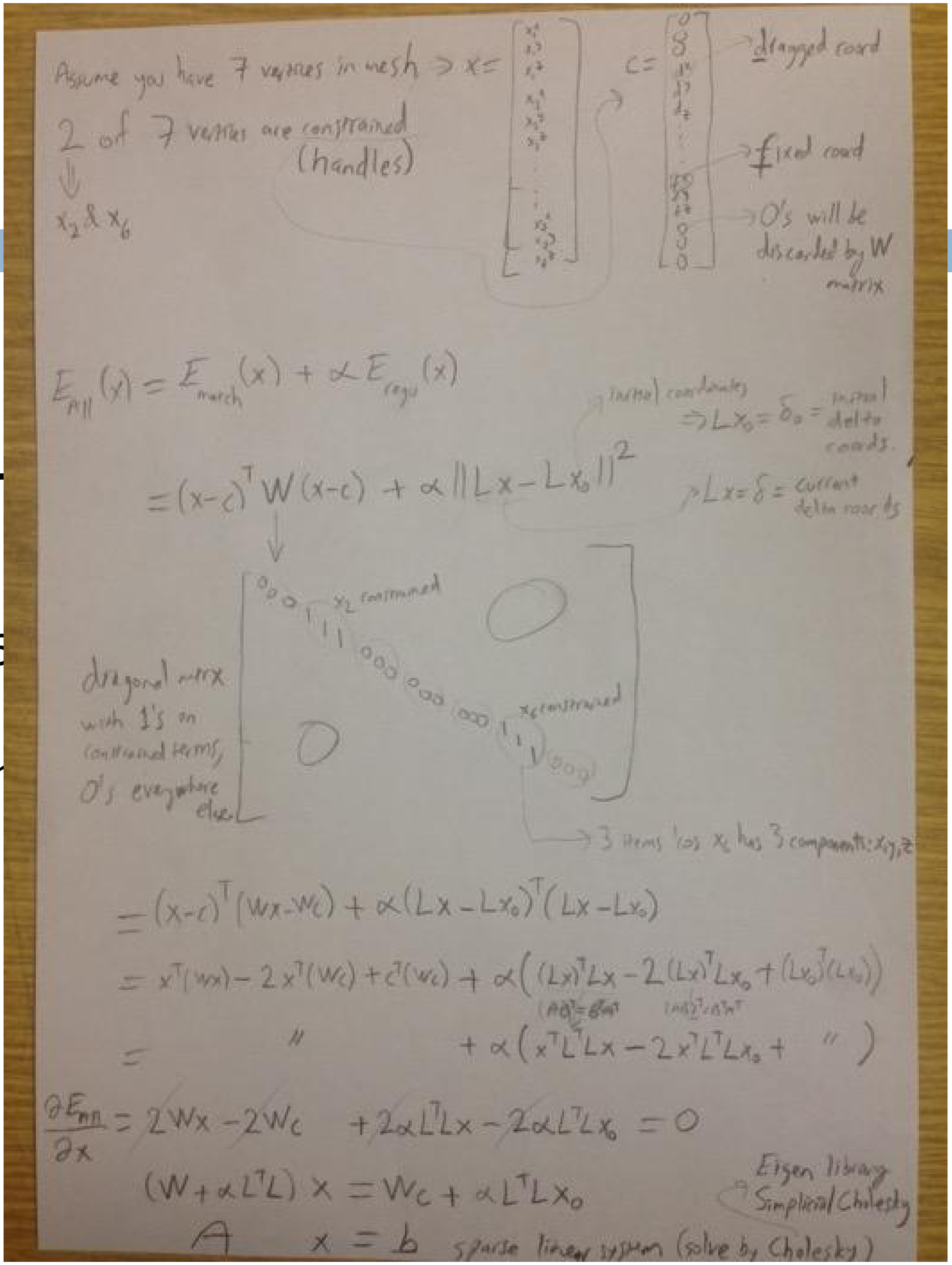
$E(v) = (v-c)^T W (v-c) + \alpha \|Lv - Lv_0\|^2$

$$\begin{bmatrix} 0 & & & & & \\ & 1^{v_i \text{ constrained}} & & & 0 & \\ & & 0 & & & \\ & 0 & & 0 & & \\ & & & & 1^{v_j \text{ constrained}} & \\ & & & & & 0 \end{bmatrix}$$

⇒ diagonal matrix with 1's on constrained terms, 0's everywhere else.

$\frac{\partial E}{\partial v} = \cancel{2} Wv - \cancel{2} Wc + \cancel{2}\alpha L^T L v - \cancel{2}\alpha L^T L v_0 = 0$

$(W + \alpha L^T L) v = Wc + \alpha L^T L v_0$ ⇒ $Ax = b$ sparse linear sys. (solve by Cholesky)



Assume you have 7 vertices in mesh ⇒ $x = \begin{bmatrix} x_1^x \\ x_1^y \\ x_1^z \\ x_2^x \\ x_2^y \\ x_2^z \\ \vdots \\ x_7^x \\ x_7^y \\ x_7^z \end{bmatrix}$ $c = \begin{bmatrix} 0 \\ 0 \\ 0 \\ d^x \\ d^y \\ d^z \\ \vdots \\ f^x \\ f^y \\ f^z \\ 0 \\ 0 \\ 0 \end{bmatrix}$ → dragged coord → fixed coord → 0's will be discarded by W matrix

2 of 7 vertices are constrained (handles)

⇓

x_2 & x_6

$E_{All}(x) = E_{match}(x) + \alpha E_{regu}(x)$

$= (x-c)^T W (x-c) + \alpha \|Lx - Lx_0\|^2$

initial coordinates ⇒ $Lx_0 = \delta_0$ = initial delta coords.

$Lx = \delta$ = current delta coords

$$\begin{bmatrix} 0\,0\,0 & & & & \\ & 1\,1\,1^{\,x_2 \text{ constrained}} & & 0 & \\ & & 0\,0\,0 \;\; 0\,0\,0 \;\; 0\,0\,0 & & \\ & 0 & & 1\,1\,1^{\,x_6 \text{ constrained}} & \\ & & & & 0\,0\,0 \end{bmatrix}$$

diagonal matrix with 1's on constrained terms, 0's everywhere else.

→ 3 items 'cos x_6 has 3 components: x, y, z

$= (x-c)^T (Wx - Wc) + \alpha (Lx - Lx_0)^T (Lx - Lx_0)$

$= x^T(Wx) - 2x^T(Wc) + c^T(Wc) + \alpha\left((Lx)^T Lx - 2(Lx)^T Lx_0 + (Lx_0)^T(Lx_0)\right)$

$(AB)^T = B^T A^T$ $(AB)^T = B^T A^T$

$=$ 〃 $+ \alpha\left(x^T L^T L x - 2x^T L^T L x_0 + \text{〃}\right)$

$\frac{\partial E_{All}}{\partial x} = \cancel{2} Wx - \cancel{2} Wc + \cancel{2}\alpha L^T L x - \cancel{2}\alpha L^T L x_0 = 0$

$(W + \alpha L^T L)\, x = Wc + \alpha L^T L x_0$

$A \quad x = b$ sparse linear system (solve by Cholesky)

Eigen library SimplicialCholesky

Deformation Energies

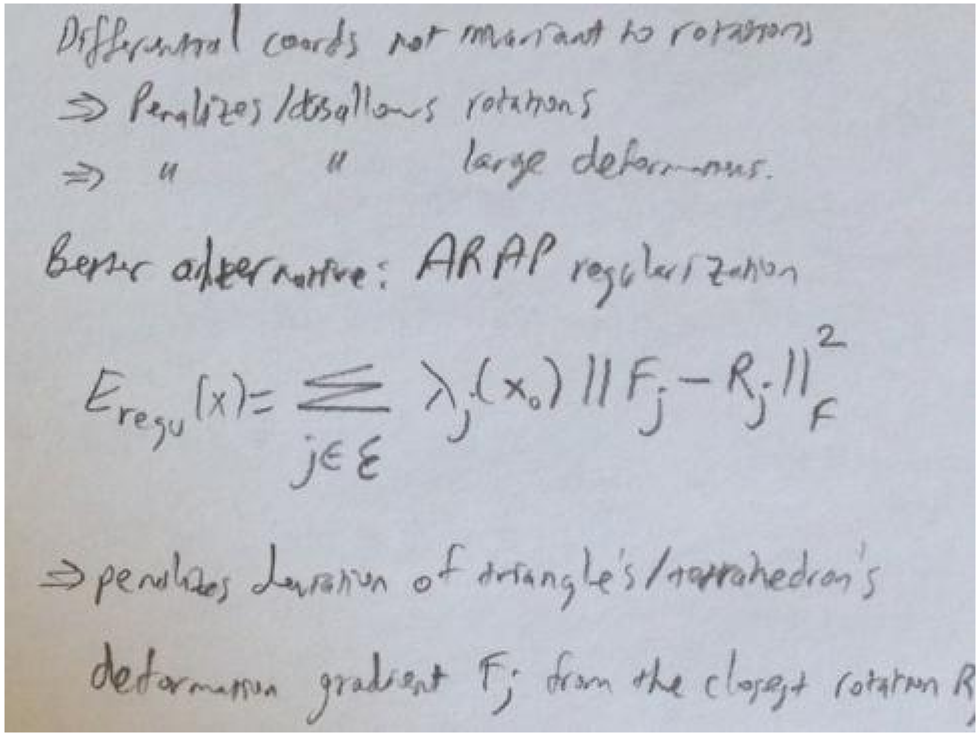
- ✓ Yet another deformation energy that incurs: displacement (d) of 2 connected points (i and j) should be small, especially when the length of the connecting edge (x) is small.

$$E_{\text{displ}} = \sum_{(\mathbf{x}_i,\mathbf{x_j})\in \text{edges}} \|\mathbf{d_i} - \mathbf{d_j}\|^2 / \|\mathbf{x_i} - \mathbf{x_j}\|^2$$

- ✓ A non-linear energy term such as this one can be minimized using the Gauss-Newton method.

ARAP Energy

- ✓ An even better deformation energy: Measure difference of each element's deformation gradient from the closest rigid rotation: ARAP.
- ✓ Invariant to rotations, hence supports larger deformations than Differential coordinates, which is variant to (= penalizes) rotations.



Differential coords not invariant to rotations

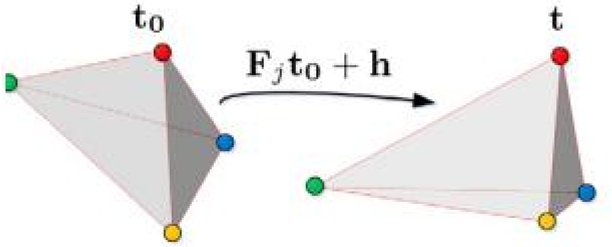
⇒ Penalizes / disallows rotations

⇒ " " large deformations.

Better alternative: ARAP regularization

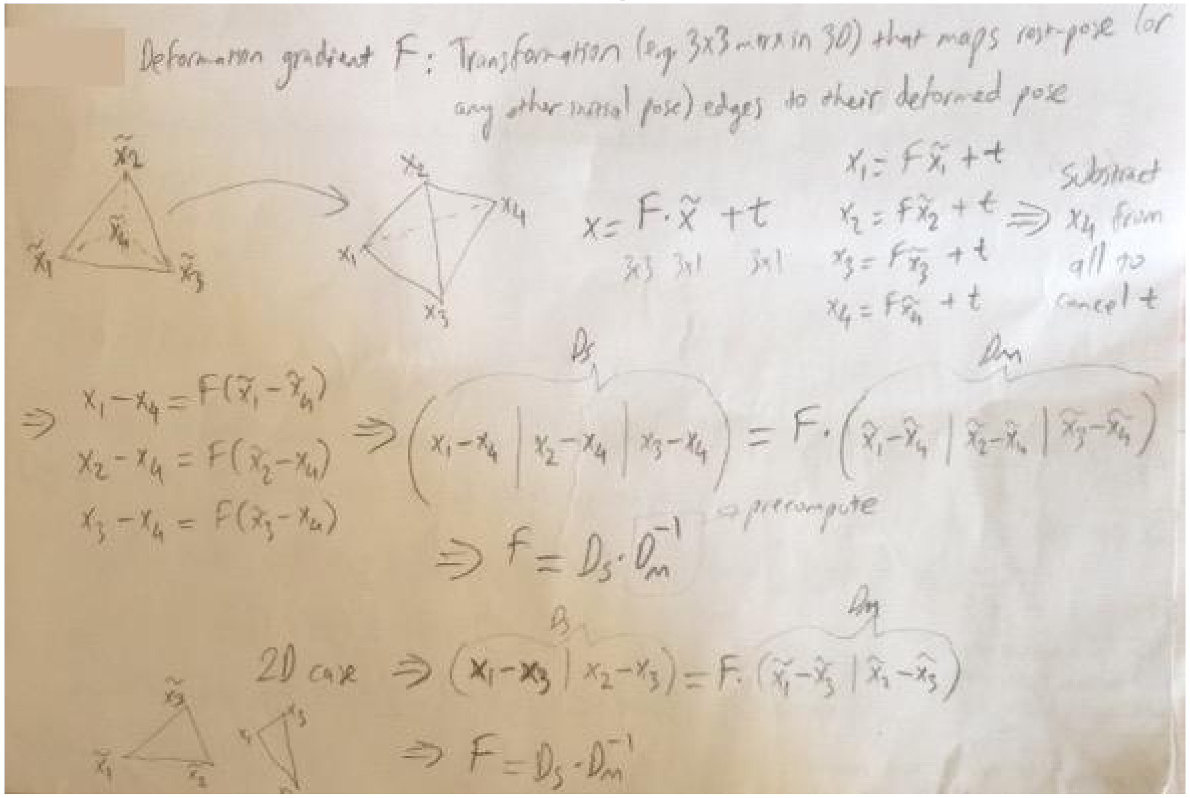
$$E_{regu}(x) = \sum_{j \in \mathcal{E}} \lambda_j(x_0) \| F_j - R_j \|_F^2$$

⇒ penalizes deviation of triangle's / tetrahedron's deformation gradient F_j from the closest rotation R



ARAP Energy

- ✓ Deformation gradient of tetrahedron (or triangle) j.
- ✓ May be used in minimization of E_regu in previous slide (we'll not do it that way).


$\tilde{x}_2$
$\tilde{x}_4$
$\tilde{x}_1$
$\tilde{x}_3$
x_2
x_4
x_1
x_3
$\tilde{x}_3$
$\tilde{x}_1$
$\tilde{x}_2$
x_1
x_3

Deformation gradient F: Transformation (e.g. 3x3 matrix in 3D) that maps rest-pose (or any other initial pose) edges to their deformed pose

$$\underset{3\times1}{x} = \underset{3\times3}{F} \cdot \underset{3\times1}{\tilde{x}} + \underset{3\times1}{t}$$

$$\begin{aligned} x_1 &= F\tilde{x}_1 + t \\ x_2 &= F\tilde{x}_2 + t \\ x_3 &= F\tilde{x}_3 + t \\ x_4 &= F\tilde{x}_4 + t \end{aligned} \Rightarrow$$

subtract x_4 from all to cancel t

$$\Rightarrow \begin{aligned} x_1 - x_4 &= F(\tilde{x}_1 - \tilde{x}_4) \\ x_2 - x_4 &= F(\tilde{x}_2 - x_4) \\ x_3 - x_4 &= F(\tilde{x}_3 - x_4) \end{aligned} \Rightarrow \underbrace{\left(x_1 - x_4 \,\middle|\, x_2 - x_4 \,\middle|\, x_3 - x_4\right)}_{D_s} = F \cdot \underbrace{\left(\tilde{x}_1 - \tilde{x}_4 \,\middle|\, \tilde{x}_2 - \tilde{x}_4 \,\middle|\, \tilde{x}_3 - \tilde{x}_4\right)}_{D_m}$$

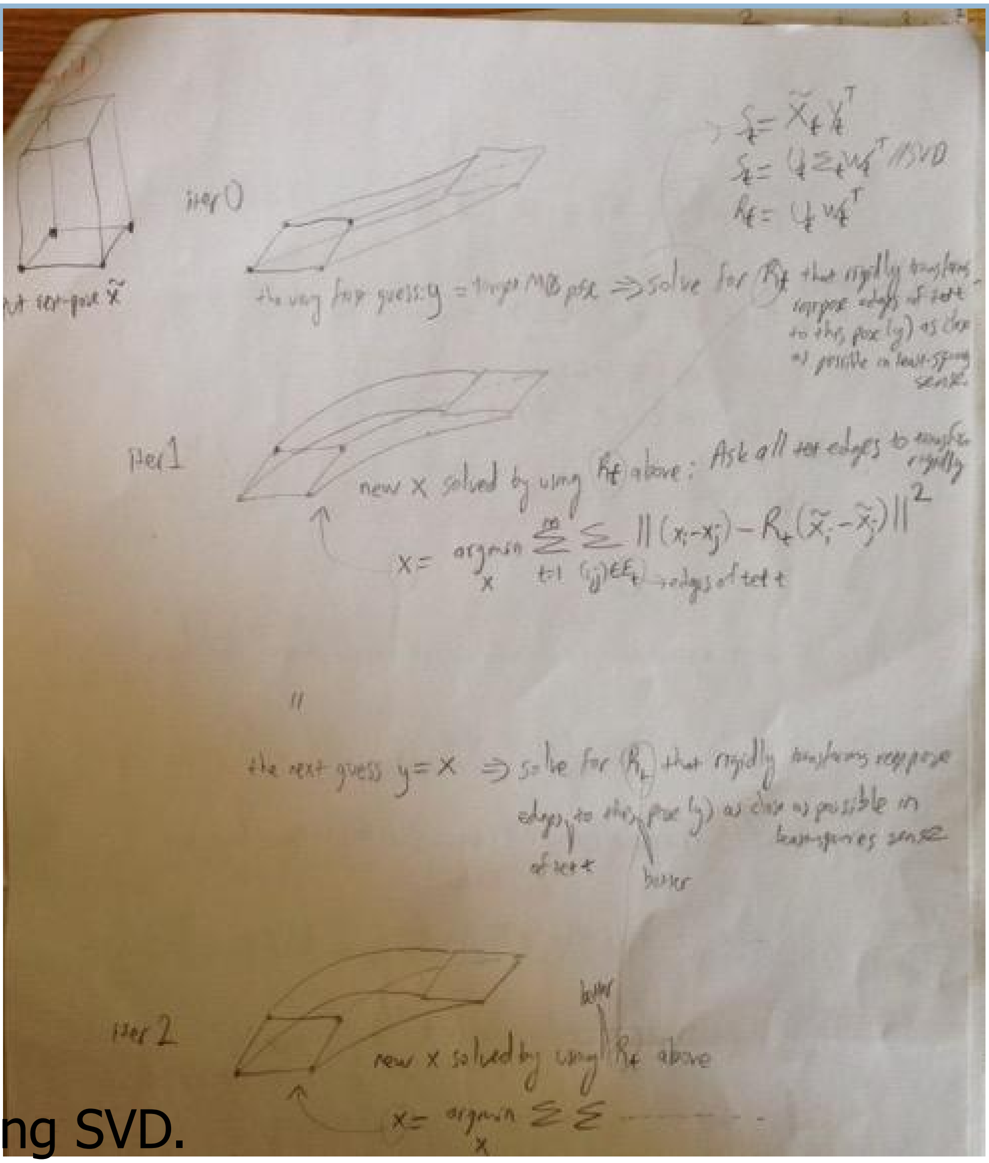
$$\Rightarrow F = D_s \cdot \boxed{D_m^{-1}} \rightarrow \text{precompute}$$

2D case $\Rightarrow \underbrace{\left(x_1 - x_3 \,\middle|\, x_2 - x_3\right)}_{D_s} = F \cdot \underbrace{\left(\tilde{x}_1 - \tilde{x}_3 \,\middle|\, \tilde{x}_2 - \tilde{x}_3\right)}_{D_m}$

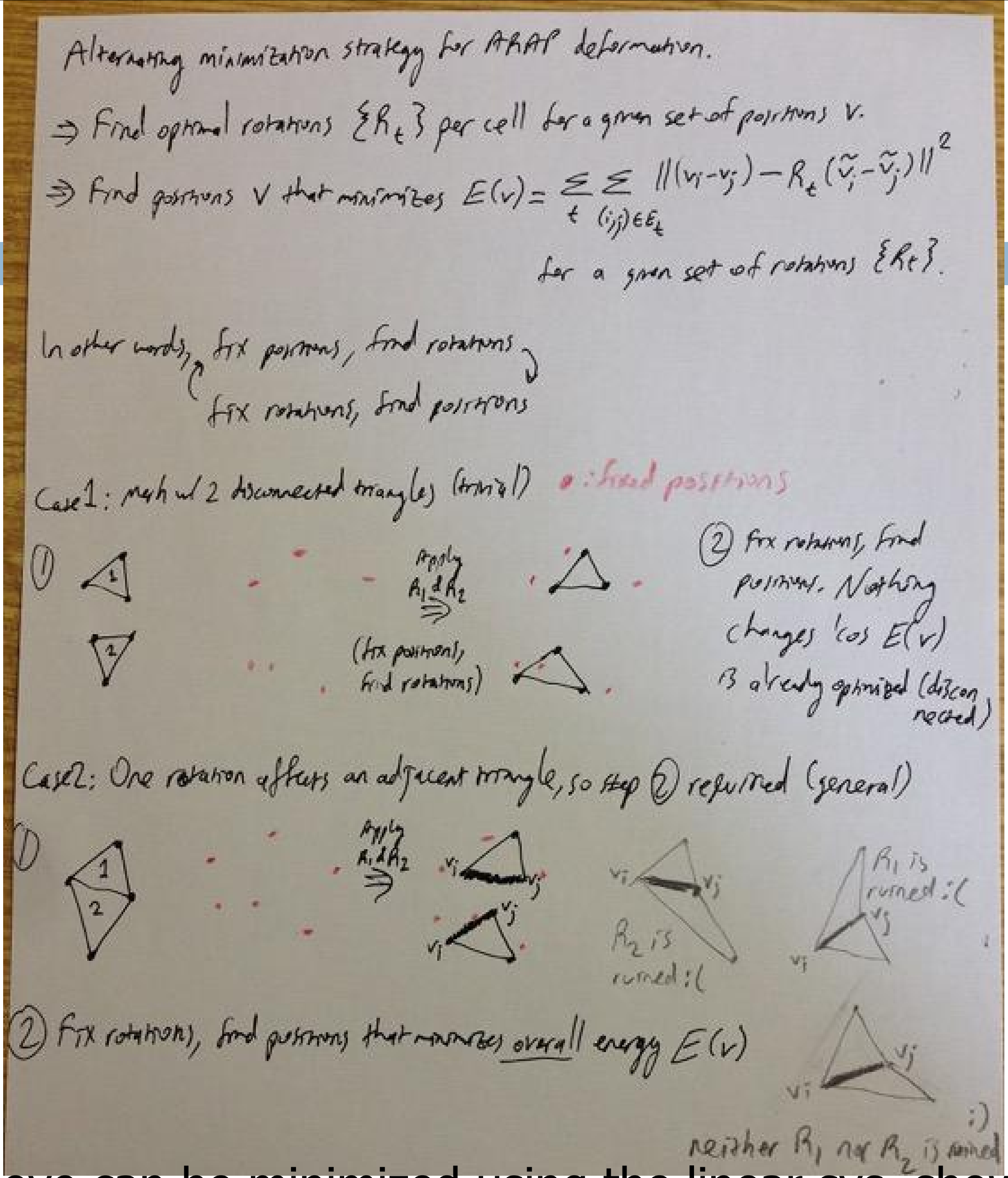
$$\Rightarrow F = D_s \cdot D_m^{-1}$$

ARAP Energy

- ✓ ARAP alternating optimization to minimize E_regu in Slide 97.
- ✓ Better step-by-step illustration starts at Slide 103.
- ✓ R_t can be found via paper: Least-Squares Rigid Motion Using SVD.


rest pose $\tilde{x}$
iter 0
the very first guess: y = target MoCap pose ⇒ solve for R_t that rigidly transforms rest pose edges of tet t to this pose (y) as close as possible in least-squares sense.
$S_t = \tilde{X}_t Y_t^T$
$S_t = U_t \Sigma_t V_t^T$ // SVD
$R_t = U_t V_t^T$
iter 1
new x solved by using R_t above: Ask all rest edges to transform rigidly
$x = \arg\min_x \sum_{t=1}^{m} \sum_{(i,j) \in E_t} \| (x_i - x_j) - R_t(\tilde{x}_i - \tilde{x}_j) \|^2$
edges of tet t
the next guess y = x ⇒ solve for R_t that rigidly transforms rest pose edges of tet t to this pose (y) as close as possible in least-squares sense
better
iter 2
new x solved by using R_t above
better
$x = \arg\min_x \sum \sum$ ......

ARAP



Alternating minimization strategy for ARAP deformation.

⇒ Find optimal rotations $\{R_t\}$ per cell for a given set of positions v.

⇒ Find positions v that minimizes $E(v) = \sum_t \sum_{(i,j)\in E_t} \|(v_i - v_j) - R_t(\tilde{v}_i - \tilde{v}_j)\|^2$ for a given set of rotations $\{R_t\}$.

In other words, fix positions, find rotations, fix rotations, find positions

Case1: mesh w/ 2 disconnected triangles (trivial) •: fixed positions

① Apply R_1 & R_2 ⇒ (fix positions, find rotations)

② Fix rotations, find positions. Nothing changes 'cos E(v) is already optimized (disconnected)

Case2: One rotation affects an adjacent triangle, so step ② required (general)

① Apply R_1 & R_2 ⇒

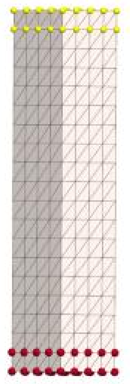
② Fix rotations, find positions that minimizes overall energy E(v)

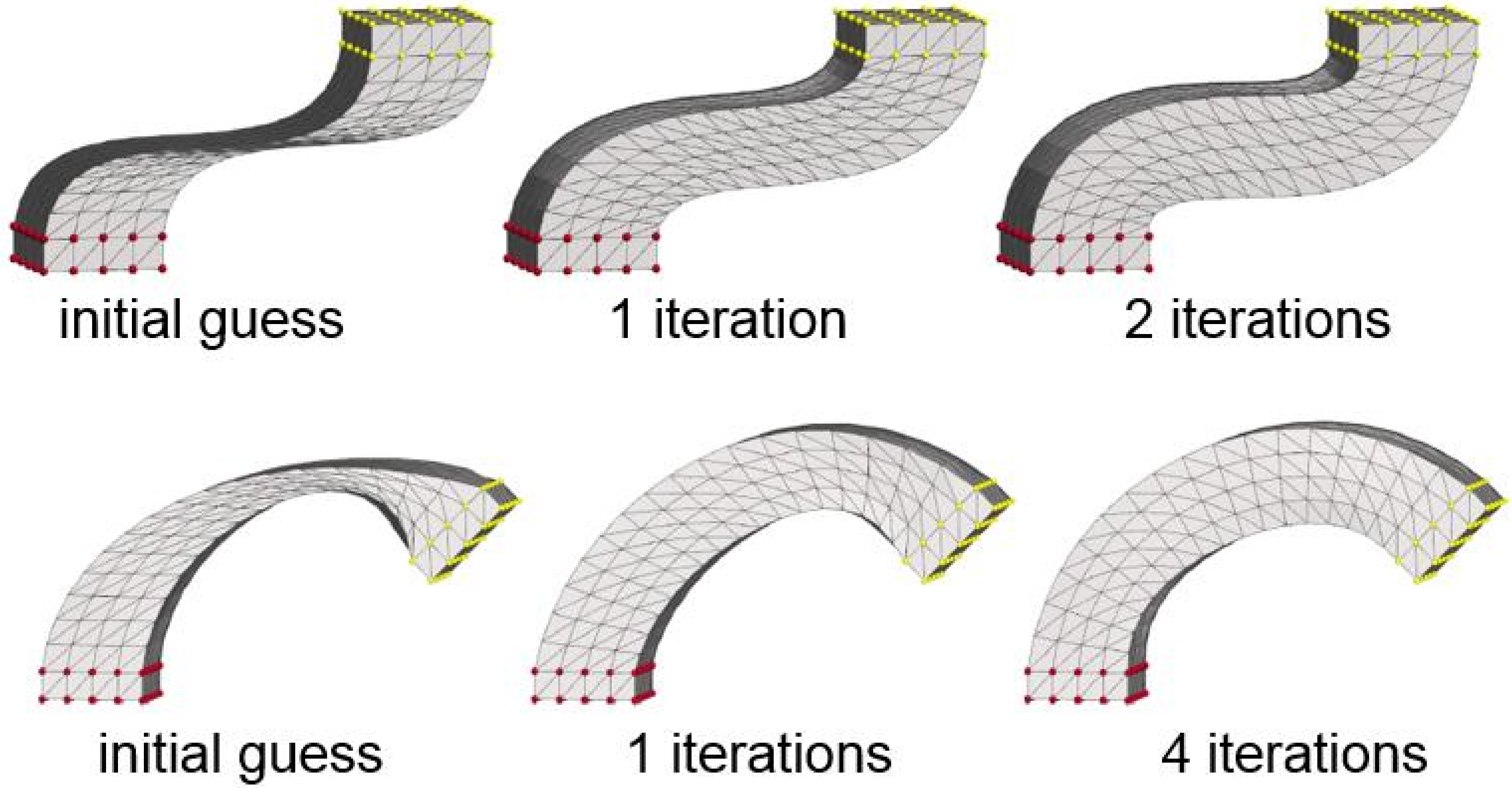
:) neither R_1 nor R_2 is ruined

✓ E(v) above can be minimized using the linear sys. shown in Eq. 9 of the ARAP paper: As-Rigid-As-Possible Surface Modeling, 2007.

ARAP Deformation

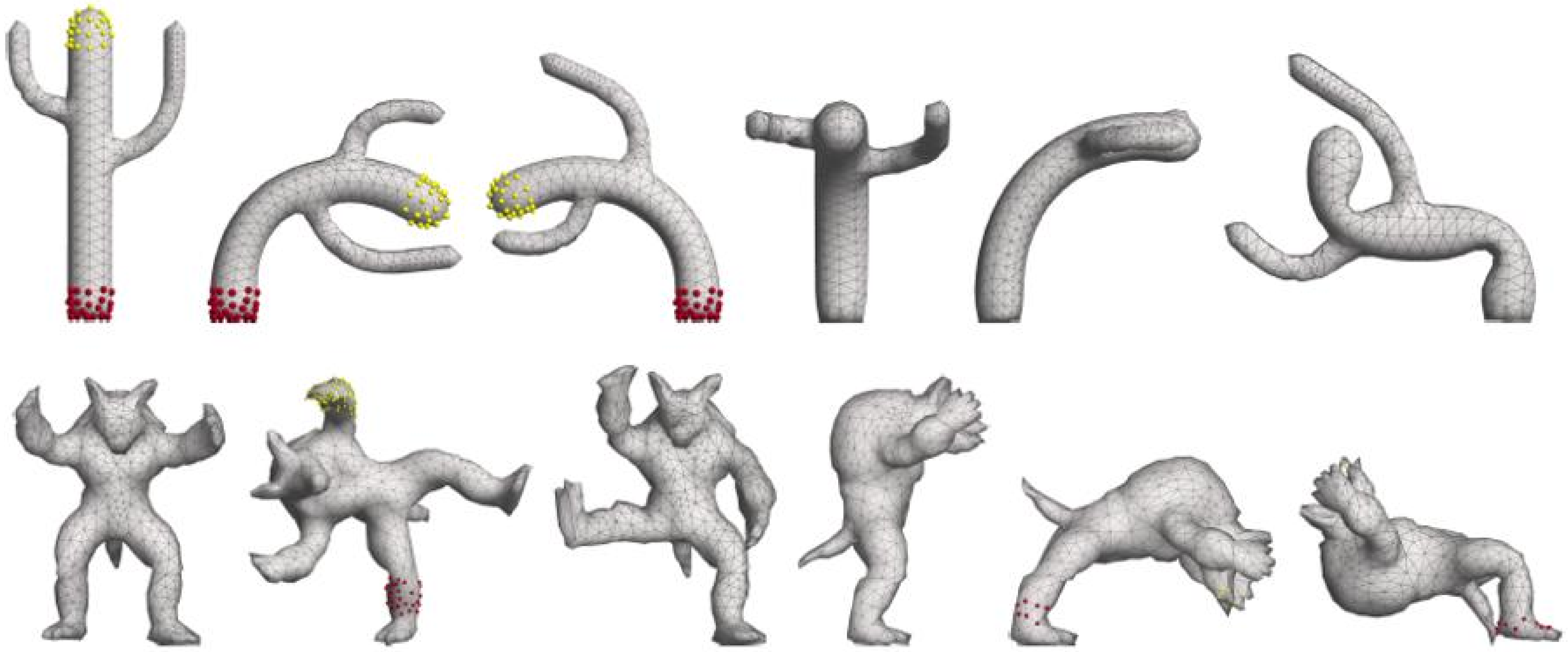
- ✓ As-rigid-as-possible (ARAP) deformation model.
  - ✓ Invariant to rotations → large deformation support.
  - ✓ State-of-the-art.





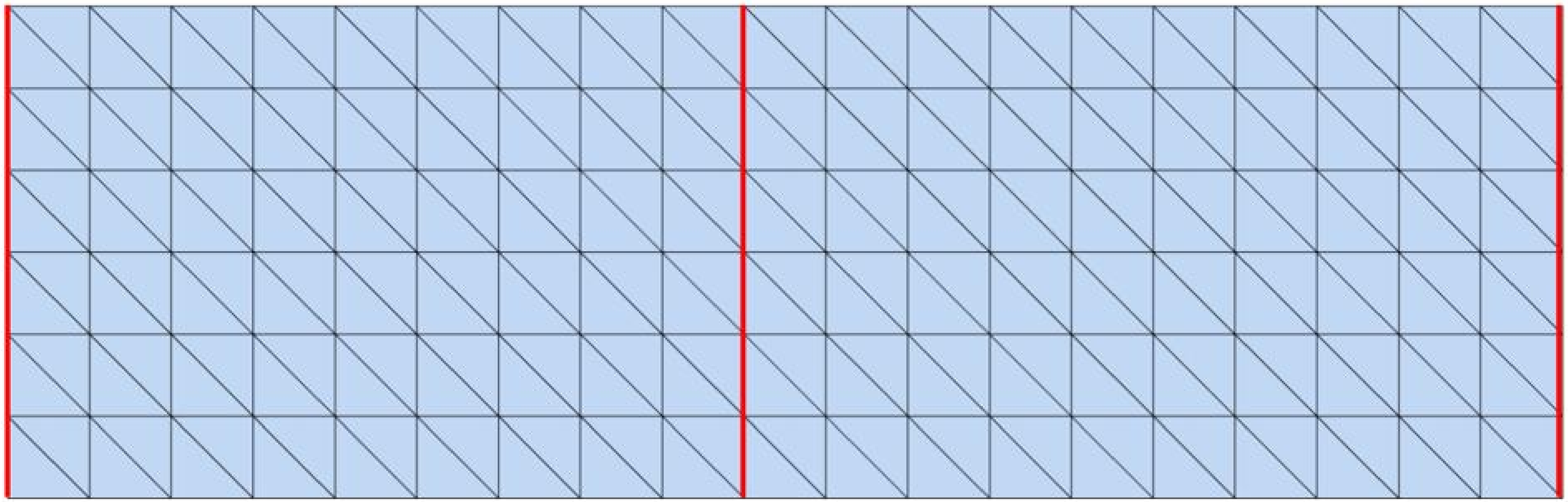
ARAP Deformation

- ✓ As-rigid-as-possible (ARAP) deformation model.
  - ✓ Invariant to rotations → large deformation support.
  - ✓ State-of-the-art.



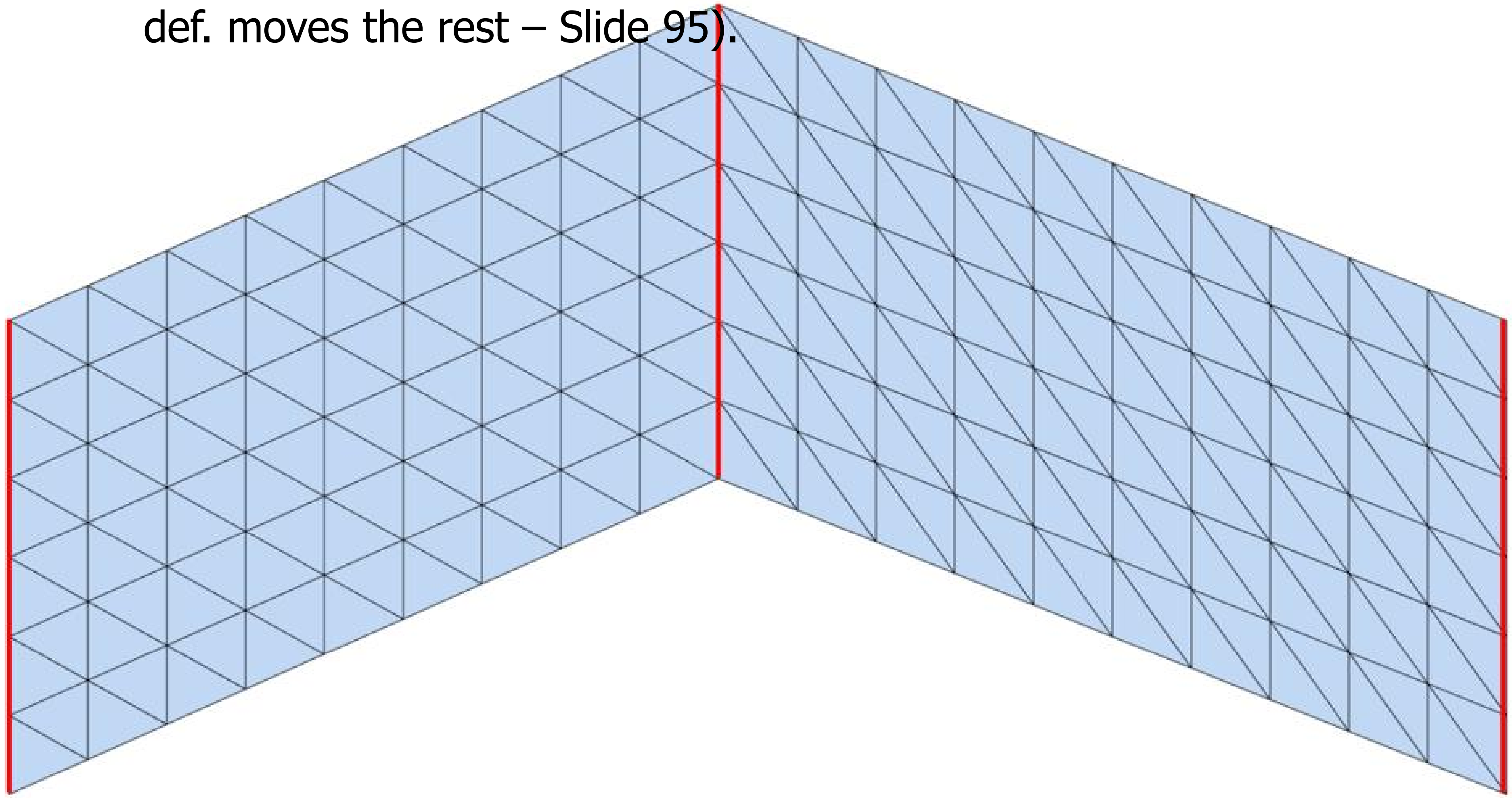
ARAP Deformation

- ✓ Found this illustration in Aigerman's IGS'16 summer school presentation.
- ✓ Input mesh.
- ✓ Rest-pose.



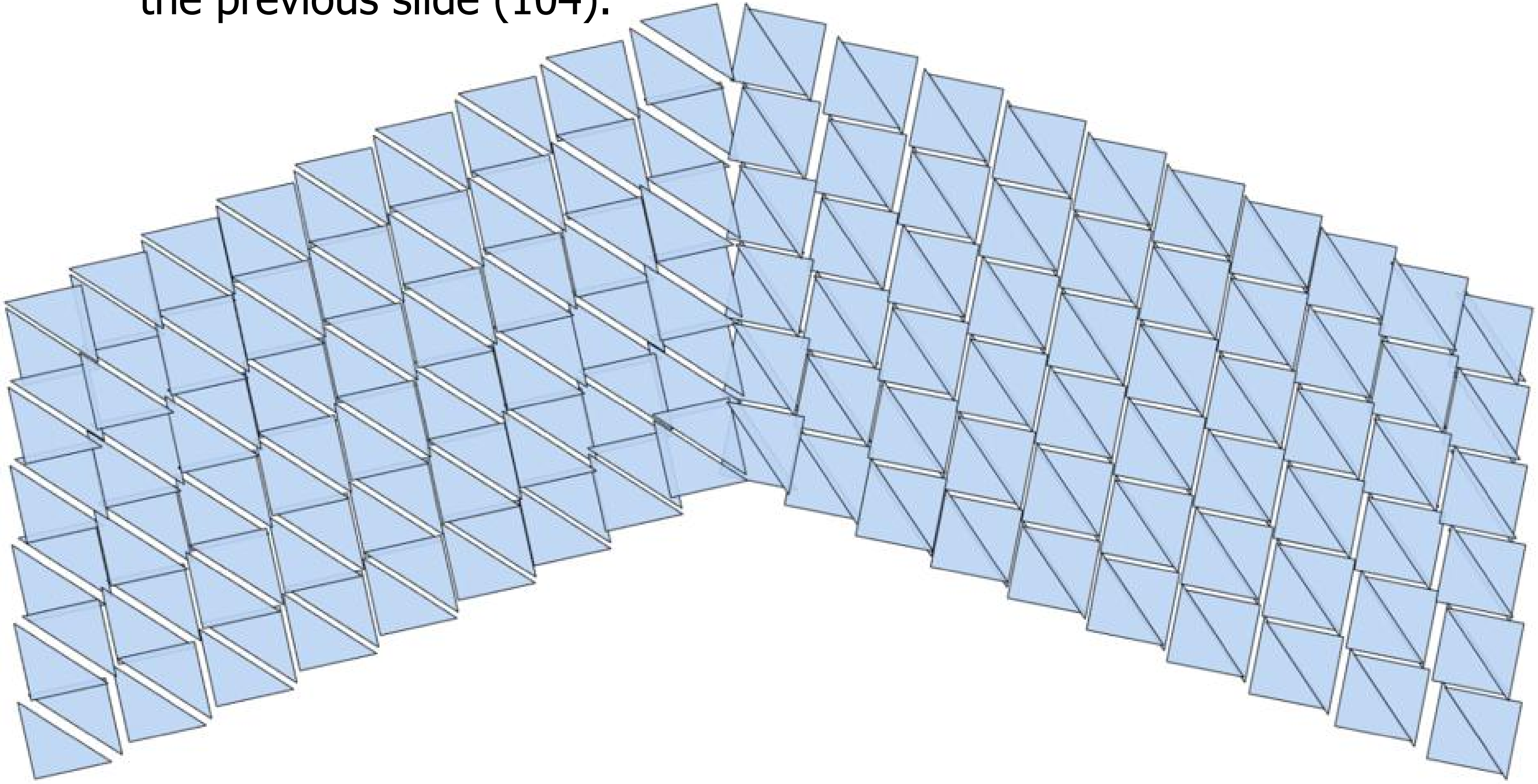
ARAP Deformation

- ✓ Initial guess (left-right fixed, middle dragged by user, simple Laplacian def. moves the rest – Slide 95).



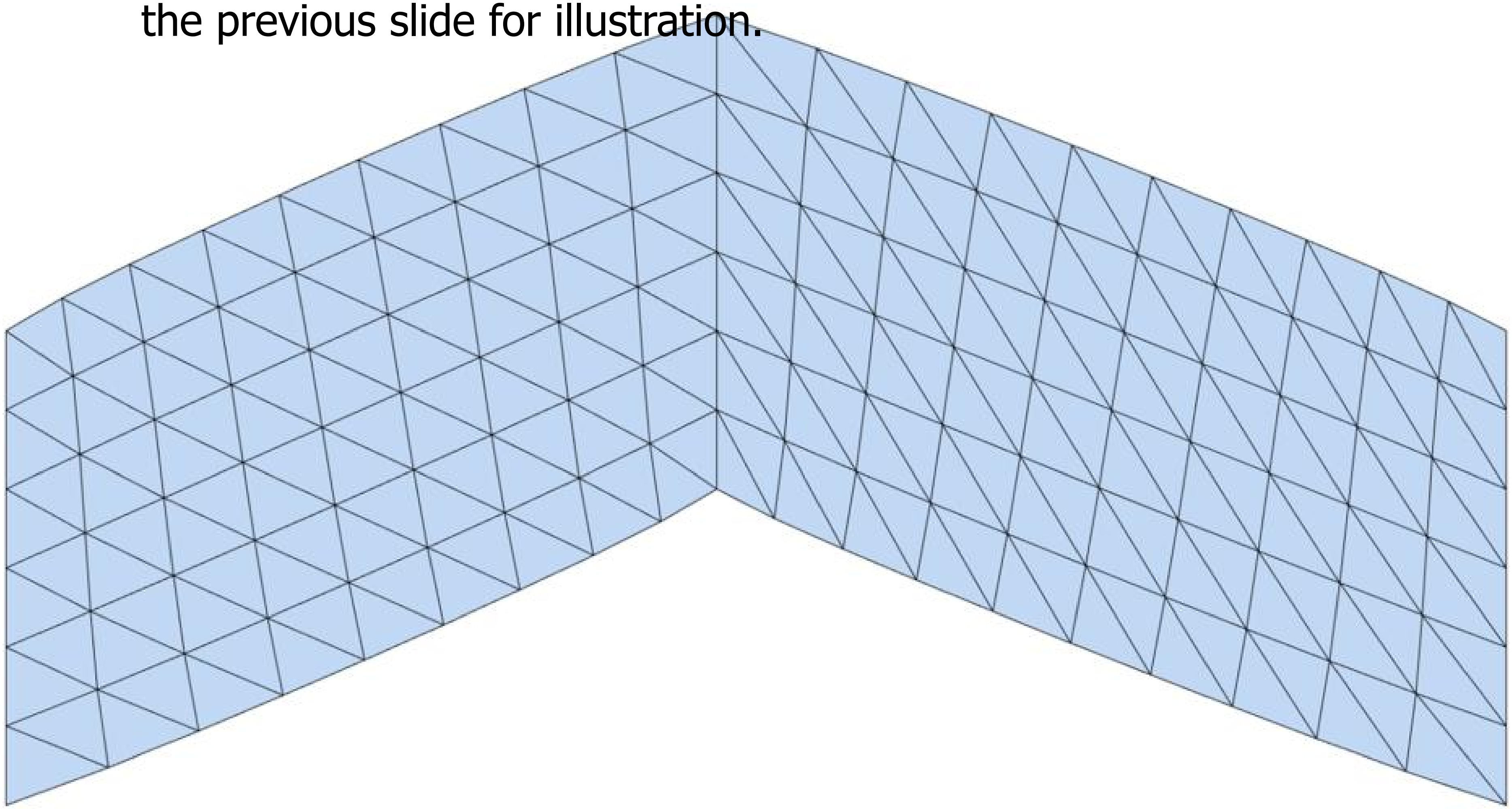
ARAP Deformation

- ✓ Apply optimal rotations from rest-pose triangles to their counterparts in the previous slide (104).



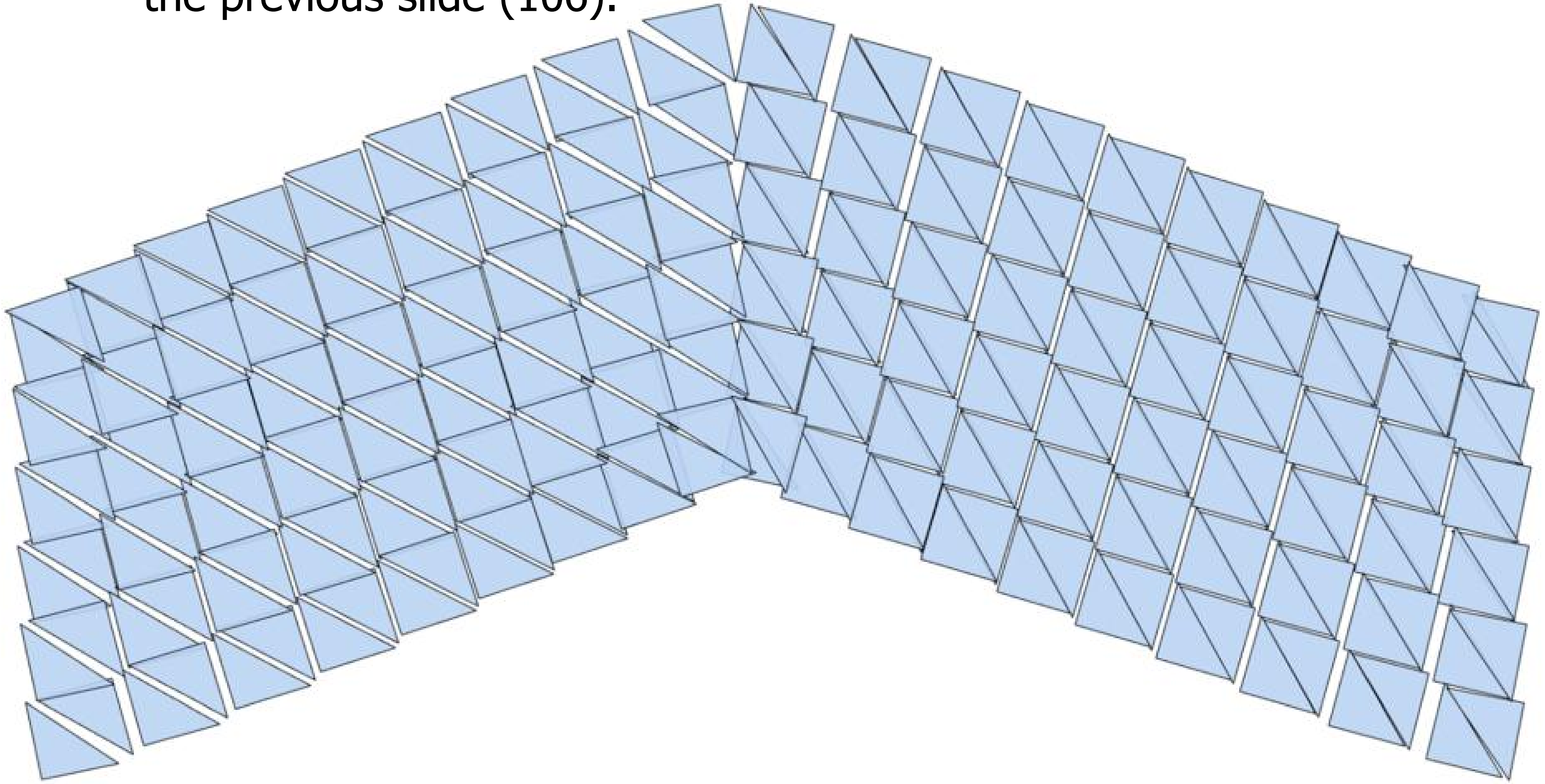
ARAP Deformation

✓ Minimize E(v) in Slide 100. Note that vertices/edge are duplicated in the previous slide for illustration.



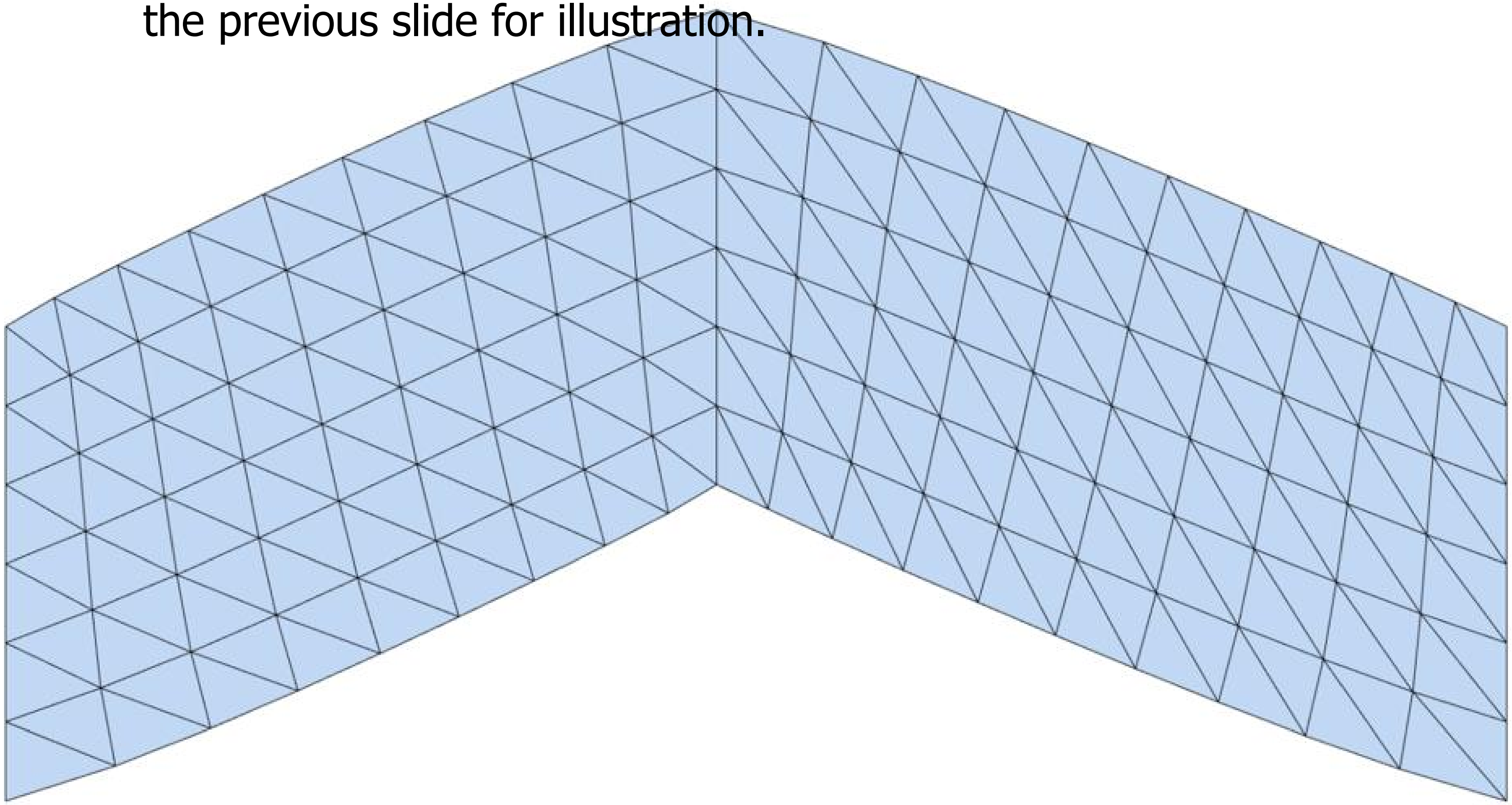
ARAP Deformation

✓ Apply optimal rotations from rest-pose triangles to their counterparts in the previous slide (106).



ARAP Deformation

- ✓ Minimize E(v) in slide 100. Note that vertices/edge are duplicated in the previous slide for illustration.



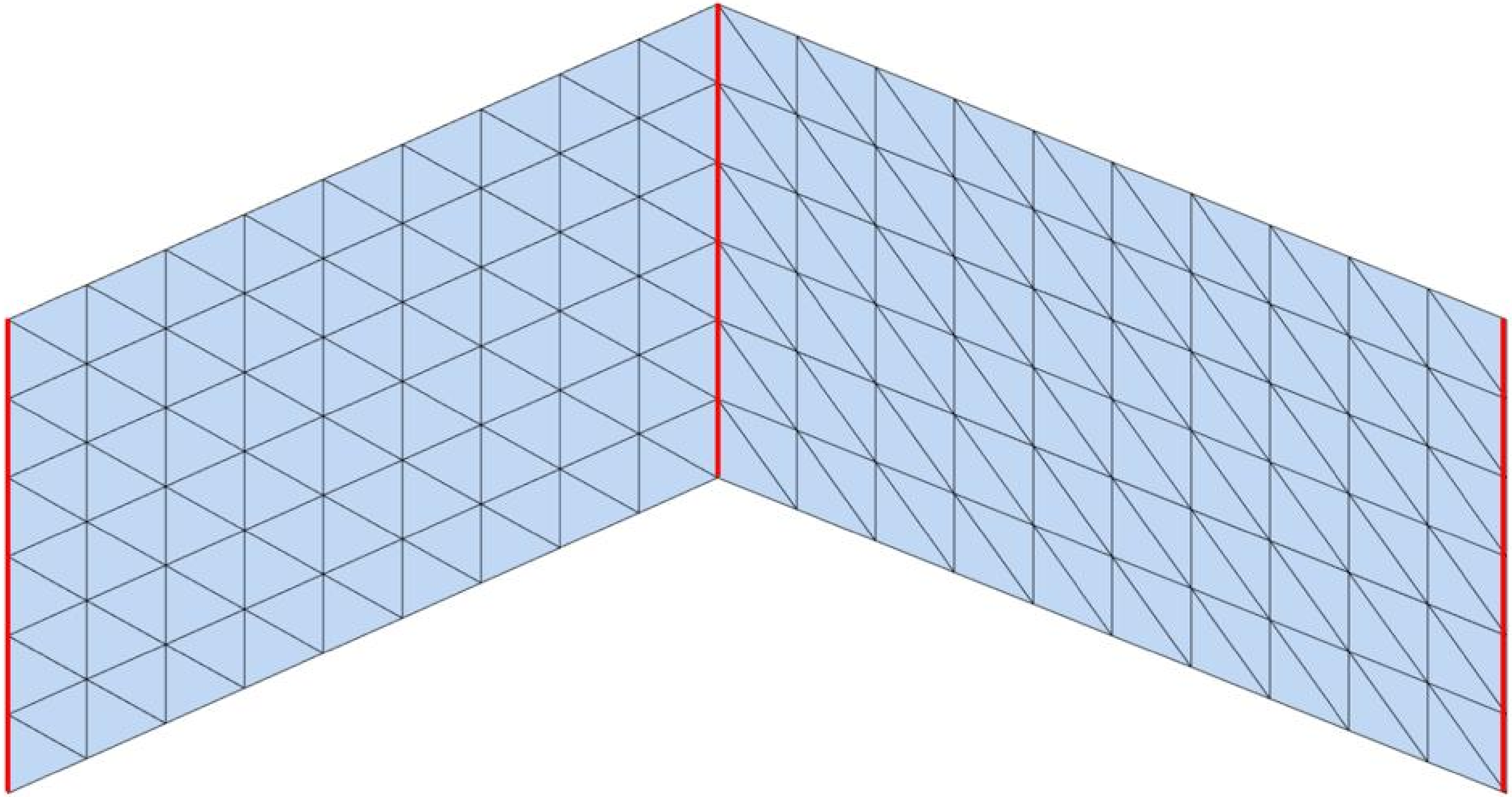
ARAP Deformation

- ✓ And so on. 5-6 iterations sufficient for most cases. Stop automatically when displacement b/w two consecutive iterations is sufficiently small.
- ✓ Note that SVD decomposition of 3x3 matrix for rotation is super-fast.
  - ✓ I recommend: http://pages.cs.wisc.edu/~sifakis/project_pages/svd.html
- ✓ Note that Laplace system matrix (minimization of E(v)) is constant throughout the iterations and has to be factored only once (super-fast).

- ✓ Let's see the iteration results again w/o the rotation slides.

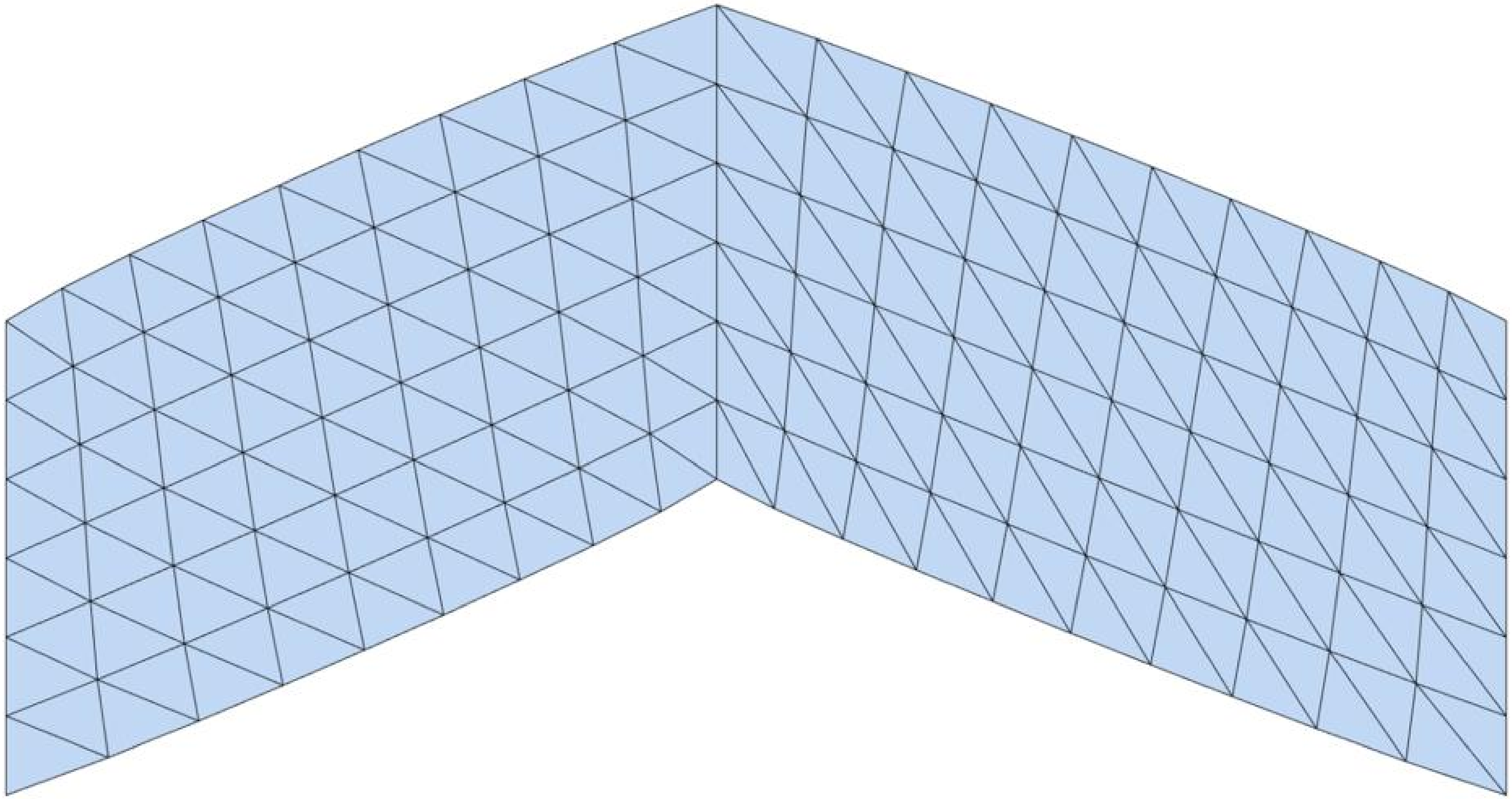
ARAP Deformation

✓ Initial guess (no iteration result yet).



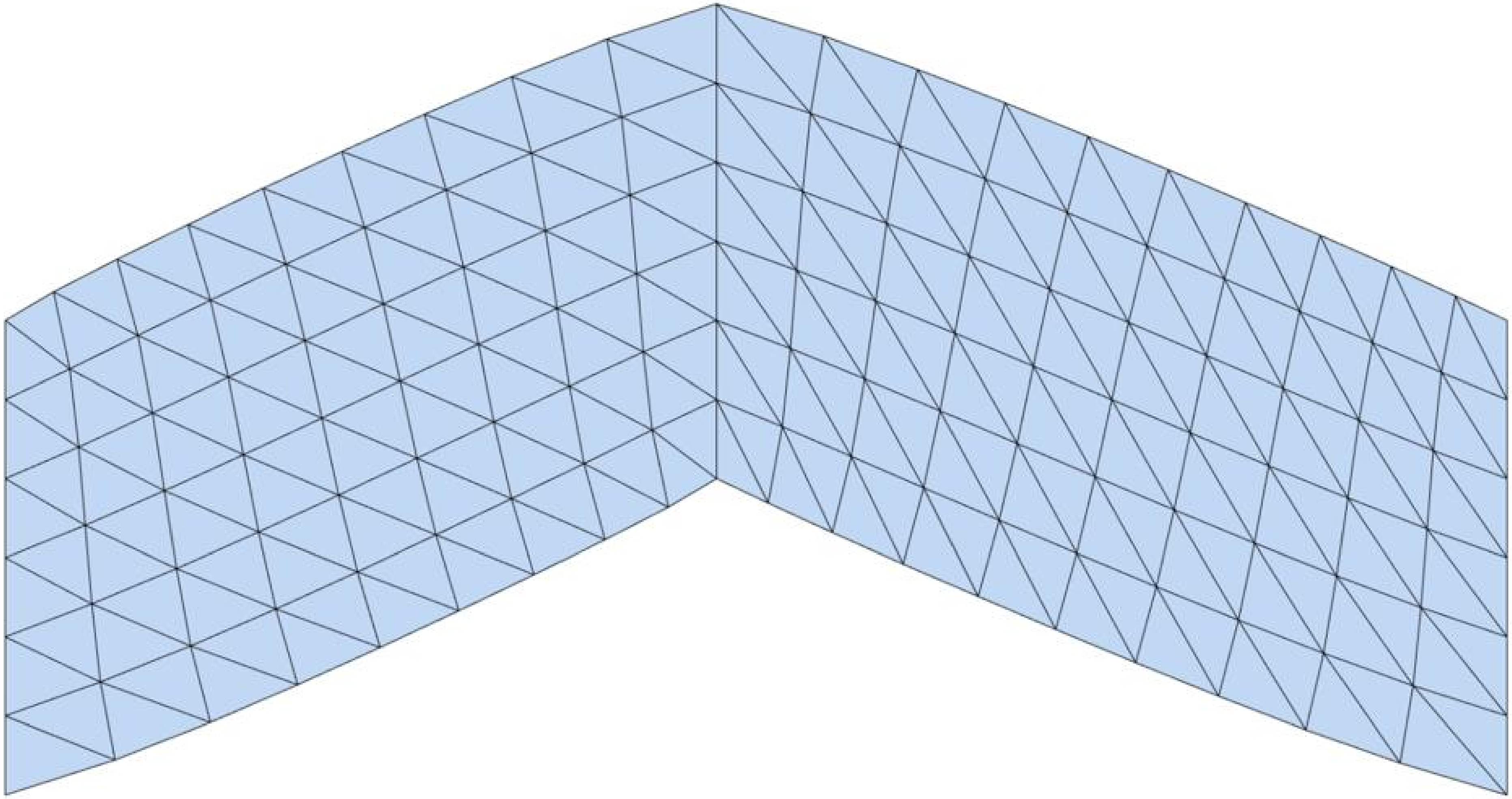
ARAP Deformation

✓ End of iteration # 1.



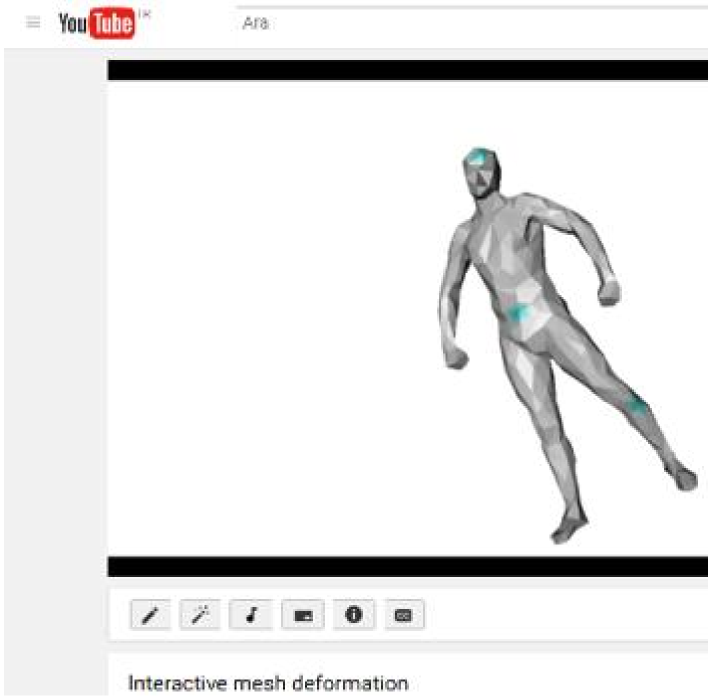
ARAP Deformation

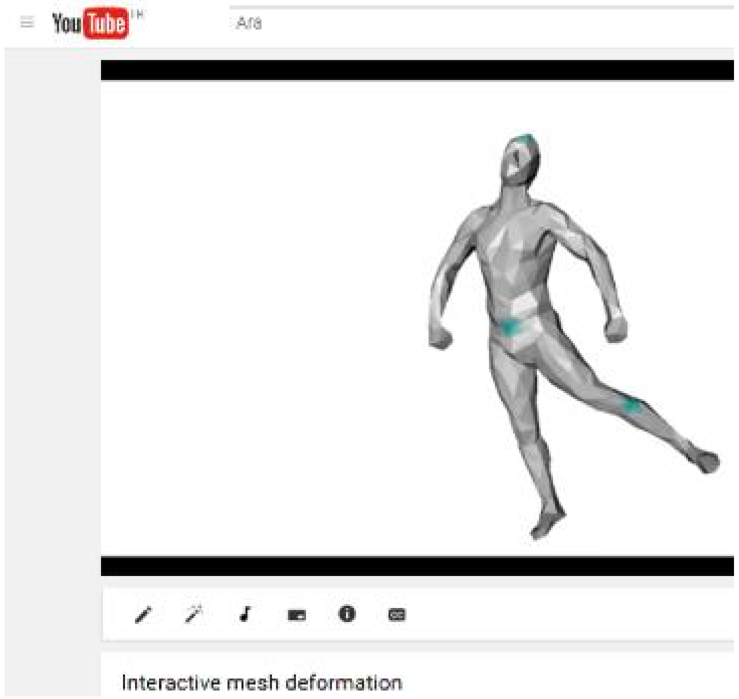
- ✓ End of iteration # 2.



ARAP Deformation

- ✓ As-rigid-as-possible (ARAP) deformation model.
  - ✓ Invariant to rotations → large deformation support.
  - ✓ State-of-the-art.
  - ✓ Video capture from my real-time interaction session: https://youtu.be/yAp1xw5JLiw





Potential Project Topics

- ✓ Implement a paper about interpolation on polygons.
  - ✓ http://cgvr.cs.uni-bremen.de/teaching/cg_literatur/barycentric_floater.pdf
  - ✓ https://www.mn.uio.no/math/english/people/aca/michaelf/papers/wach_mv.pdf
  - ✓ http://folk.uio.no/martinre/Publications/mv3d.pdf
  - ✓ https://www.youtube.com/watch?v=PDJxjXCjiis
  - ✓ http://hecodes.com/2016/07/mesh-manipulation-using-mean-values-coordinates-in-three-js/
- ✓ Implement a paper about interpolation on surfaces.
  - ✓ Weighted Averages on Surfaces, 2013.
  - ✓ Barycentric Coordinates on Surfaces, 2010.
- ✓ Implement deformation via FFD, ARAP, Least-squares Meshes paper, or the local coord. Laplacian in Slide 69.
- ✓ Implement physically-based deformation, e.g., the paper titled Meshless deformation based on shape matching.