

Contents

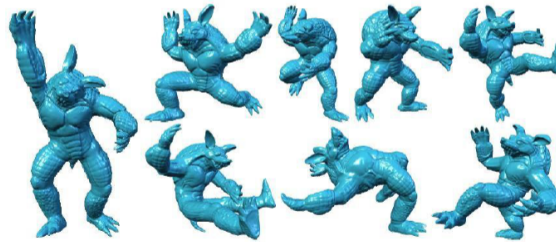
1	Deformation and Non-Rigid Surface Tracking	2
1.1	Deformation	2
1.1.1	Surface deformation	3
1.1.1.1	Laplacian Surface Editing	3
1.1.1.2	As-Rigid-as-Possible Deformation	3
1.1.1.3	Embedded Deformation	4
1.1.1.4	ARAP vs ED	5
1.1.2	Space deformation	5
1.1.2.1	Cages	5
1.1.2.2	Grids	5
1.1.2.3	Skeleton	6
1.1.2.4	Deformation Graphs	6
1.1.2.5	Summary on Deformation Proxies	7
1.2	Non-Rigid Surface Tracking	7

1 Deformation and Non-Rigid Surface Tracking

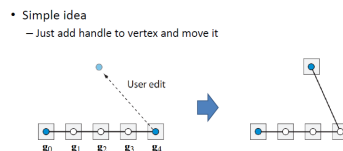
Until now we have only talked about rigid-surface tracking and reconstruction. What if we'd like to track and reconstruct a non-rigid surface? In this case the problem is a little harder as it's very difficult to **distinguish changes caused by the surface deformation and caused by motion**. To get in depth to this problem, let's first talk about how do we deform a surface.

1.1 Deformation

Mesh deformation has a lot of useful applications such as character animation. Given a character mesh, we'd like to create different poses of this character by moving only a few key vertices.



Of course this could be done by manually adjusting vertices one by one, but this would be very time inefficient.



So what we'd like to do is to create a domino effect such that, when one vertex is moving, the movement should be propagated to it's adjacent vertices, then to the adjacent vertices of it's adjacent vertices, and so on.



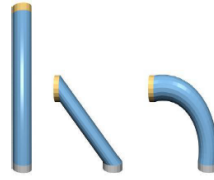
This moving behavior can not be arbitrary defined and should be intuitive and natural:

- It should achieve a smooth global deformation.
- Global changes should preserve local detail.

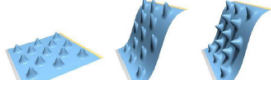
The moving behavior is defined by a **regularizer function**, and the full energy term is described as:

$$\begin{aligned}
 & \bullet E_{fit}(\mathbf{V}) = \sum_{i \in C} \|c_i - v_i\|_2^2 \\
 & \quad \swarrow \quad \quad \quad \nearrow \quad \quad \quad \nearrow \\
 & \quad \text{Loop over all picked points by user} \quad \text{Target pos. of correspondence} \quad \text{Picked correspondence} \\
 & \bullet E_{reg}(\mathbf{V}, \mathbf{X}) = \dots \quad \leftarrow \text{Somehow make neighbors move!} \\
 & \bullet \text{Minimize } \lambda_{fit} E_{fit}(\mathbf{V}) + \lambda_{reg} E_{reg}(\mathbf{V}, \mathbf{X})
 \end{aligned}$$

- How to choose a regularizer?
- Computed with respect to base mesh V
- Intuitive and natural behavior
 - Smooth global deformation
 - Global changes should preserve local detail
 - Should be zero in un-deformed state (base mesh)



- $E_{reg}(V, X) = \dots$



1.1.1 Surface deformation

This type of deformation are only suitable for meshes.

1.1.1.1 Laplacian Surface Editing

Laplacian surface editing enforce smoothness on deformations by minimizing distance between the original and transformed Laplacian coordinates. In simple words, this method obligates that neighbors of N should move in same way as N . The Laplacian coordinate of a vertex N is just the sum of the differences between N and it's adjacent vertices.

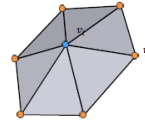
- Enforce Smoothness on deformations

$$E_{reg}(V, X) = \sum_i \sum_{j \in \mathcal{N}(i)} \left\| (v'_i - v_i) - (v'_j - v_j) \right\|_2^2$$

Sum over all vertices

Sum over 1-ring (vertex fan)

Deformation difference

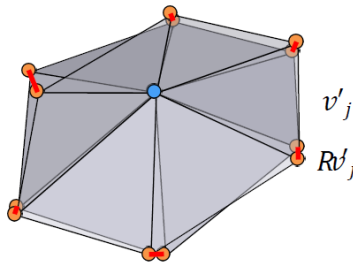


But clearly the deformation under this method is not intuitive as it assumes we are dealing with elastic surfaces.

1.1.1.2 As-Rigid-as-Possible Deformation

ARAP maximizes the local rigidity by enforcing local transformations assigned to each vertex fan are almost rigid. We assign one rotation matrix per fan and the energy is computed as the deviation from the undeformed fan to deformed vertex.

- For each vertex fan, measure deviation from rigidity

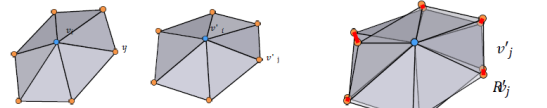


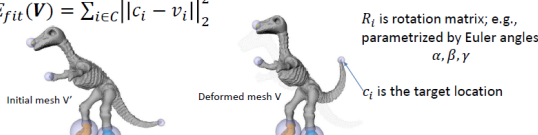
- One rotation matrix per fan; sum up deviations from rigidity

$$E_{ARAP}(R, V) = \sum_i \sum_{j \in \mathcal{N}(i)} \left\| (v_i - v_j) - R_i(v'_i - v'_j) \right\|_2^2$$

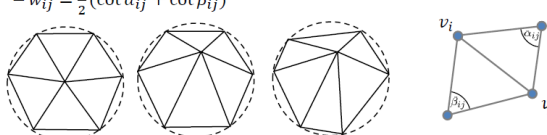
Edge (i.e., cancels translation)

Find best rotation matrix that maps undeformed fan to deformed vertex fan



- One rotation matrix per fan; sum up deviations from rigidity
 - $E_{ARAP}(\mathbf{R}, \mathbf{V}) = \sum_i \sum_{j \in \mathcal{N}(i)} \left\| (v_i - v_j) - R_i(v'_i - v'_j) \right\|_2^2$
 - $E_{fit}(\mathbf{V}) = \sum_{i \in \mathcal{C}} \|c_i - v_i\|_2^2$
- 
- R_i is rotation matrix; e.g., parametrized by Euler angles α, β, γ
 c_i is the target location

Observe that here the difference is computed on edges, **in this way we can make the the ARAP translation invariant**. We can also weight the edges so that close neighbors can count more. The weight is computed as the sum of the cotangent of the opposite angles of the edge:

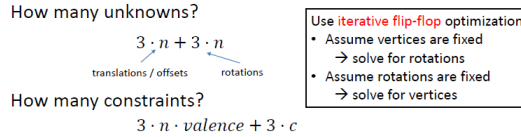
- Could weight: close neighbors should count more
 - $E_{ARAP}(\mathbf{R}, \mathbf{V}) = \sum_i \sum_{j \in \mathcal{N}(i)} w_{ij} \left\| (v_i - v_j) - R_i(v'_i - v'_j) \right\|_2^2$
 $w_{ij} = \frac{1}{2} (\cot \alpha_{ij} + \cot \beta_{ij})$
- 

Given a mesh with n vertices then we will have $3n + 3n$ unknowns and $3nvalence + 3c$ constraints:

- The first $3n$ corresponds to the position of vertices of the deformed mesh.
- The second $3n$ corresponds to the rotations matrix associated to each vertex.
- For constraints, the *valence* is basically the local connectivity of each vertex, and c is the number of correspondences.

In order to solve this problem we can either use a 2^{nd} order solver or we can use *iterative flip-flop*. The advantage of *iterative flip-flop* is that each sub-problem is linear and thus has closed-form solution.

- Given a mesh $\in \mathbb{R}^3$ with n Vertices:



The initial estimation of the rotation matrix can be estimated from the set of correspondences using SVD.

1.1.1.3 Embedded Deformation

The idea of Embedded Deformation is the same as ARAP. The only difference is that, in ARAP we have hard constraint for rotations, but in ED we have soft-constraints for rotations, that is, it can be an approximation of $SO(3)$.

- Allow deviations from rotations: given $M \in \mathbb{R}^{3 \times 3}$
- $E_{ED}(\mathbf{M}, \mathbf{V}) = \sum_i \sum_{j \in \mathcal{N}(i)} \left\| (v_i - v_j) - M_i(v'_i - v'_j) \right\|_2^2$
- $E_{rot}(\mathbf{M}) = \sum_i Rot(M_i)$ ← soft-constraint for rotation
- $Rot(M) = (c_1 \cdot c_2)^2 + (c_1 \cdot c_3)^2 + (c_2 \cdot c_3)^2 + (c_1 \cdot c_1 - 1)^2 + (c_2 \cdot c_2 - 1)^2 + (c_3 \cdot c_3 - 1)^2$
 $c_{1,2,3}$ are columns of matrix M

1.1.1.4 ARAP vs ED

- ARAP: hard-constraints for rotations
 - Forced rotation matrices \rightarrow 3 unknowns per local matrix (6 per vertex)
 - Can be efficiently optimized with flip-flop (local SVDs / Procrustes)
- ED: soft-constraints for rotations \rightarrow more DoFs
 - I.e., \rightarrow 9 unknowns per local matrix (12 per vertex)
 - Harder to optimize (actually a quartic problem)

1.1.2 Space deformation

Surface deformation is great, but it only applies on meshes and the deformation complexity is related to the number of vertices. Thus, **we can make use of proxies defined around the surface and deform the proxy instead**. This type of deformation has several advantages:

- First, it applies to all type of surface representations. Point clouds, meshes, etc.
- Second, the deformation is sparser and has fewer degrees of freedom. The deformation complexity is decoupled from the geometry complexity of the surface.
- Third, it preserves more geometric details.

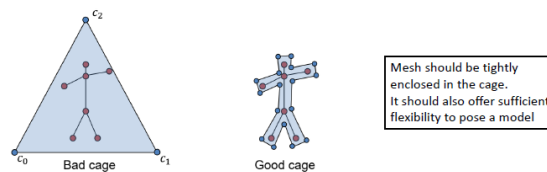
1.1.2.1 Cages

Cage is a shape proxy defined around the surface. A good cage should have the following properties:

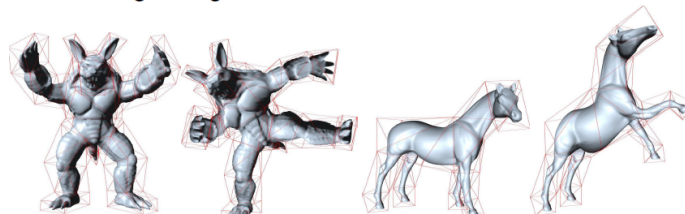
- The surface should be tightly enclosed in the cage.
- The cage should offer sufficient flexibility to pose a model.

Each vertex of the surface is defined as a linear combination w.r.t the vertices of the cage, and the coordinate of each vertex on the surface is defined by the weight of such linear combination (similar idea to barycentric coordinate). Thus, in order to deform the surface we just need to deform the cage.

- Extend the concept of barycentric coordinates to shapes (e.g., harmonic coordinates)
- Vertices are a linear combination with respect to cage



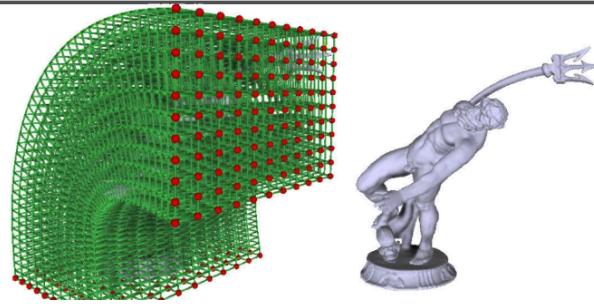
- Once cage and weights are computed, mesh can be deformed by animating the cage vertices



1.1.2.2 Grids

This method is very straightforward. For a volumetric representation, instead of deforming the surface is more intuitive to deform the grid which stores the SDF values.

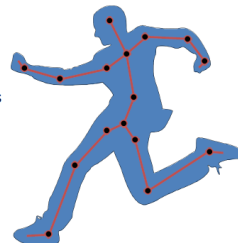
ARAP on 3D Grids



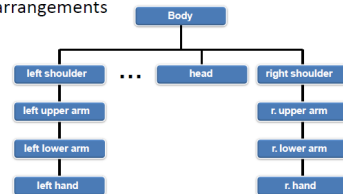
1.1.2.3 Skeleton

This method embeds skeleton in a mesh and each vertex of the mesh is a linear combination of bone matrices. In order to deform the mesh we just need to move articular points. Skeleton is typically created by hand and has hierarchical arrangements.

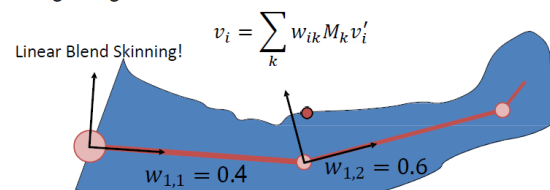
- Basic idea:
 - Embed skeleton in mesh
 - Assign mesh vertices to bones
 - Compute deformed mesh from bones



- Skeleton is typically created by hand and has hierarchical arrangements



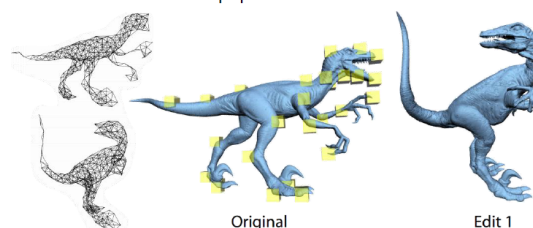
- Each vertex is linear combination of bone matrices
- Assign weights for each vertex for each matrix



We can also apply ARAP on skeletons.

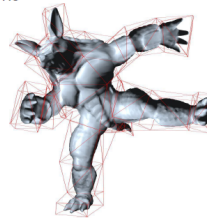
1.1.2.4 Deformation Graphs

- Introduced with the ED-paper



1.1.2.5 Summary on Deformation Proxies

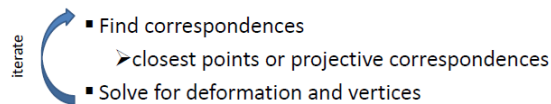
- Dimensionality reduction on deformations
- Directly on Mesh
- Cages
- Grids
- Skeletons (e.g., Linear Blend Skinning)
- Deformation Graphs
- Tetrahedral Mesh (full volume)



1.2 Non-Rigid Surface Tracking

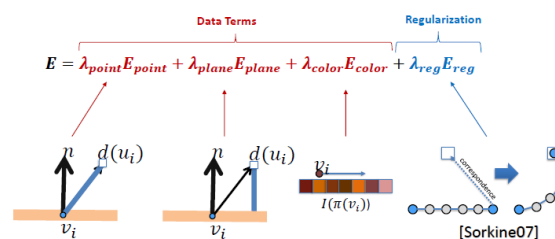
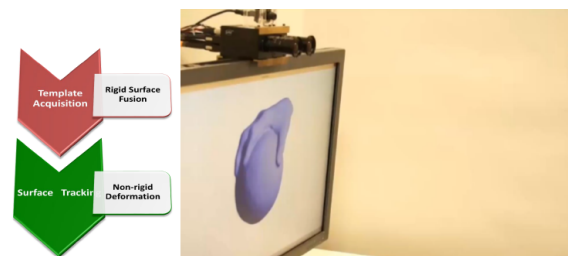
Non-Rigid Surface tracking can be done via:

- Non-rigid ICP: it's same as rigid ICP except that here we have many more degrees of freedom. Pose + vertices.
- Like rigid ICP (which has 6 DoF / frame)
 - Except we have many more degrees of freedom!



- Using Sparse Features to find the correspondences.

In order to track a non-rigid surface we need a predefined mesh, otherwise it's not easy to distinguish between surface deformation and camera moving. In the paper "Deformables" they first reconstruct the undeformed surface using rigid surface fusion and then use the reconstructed surface to track non-rigid deformations.



The color term is very important in order to make the problem over-constrained.

Observe that this is not non-rigid reconstruction as the base model is extracted beforehand. In the next lecture we will see how to do **tracking and reconstruction at the same time**.