

Dan's Notes on :

Robert Sumner and Jovan Popovic (MIT). "Deformation Transfer for Triangle Meshes." SIGGRAPH 2004.

## Overview

- The goal is to take deformations that have been applied to one triangle mesh and map them onto another triangle mesh. Note that "deformations" here might be squishy deformations in the sense that we usually talk about deformations, or it might be rigid movements of articulated model components.
- Why would we want to do this?
  - Their examples center around two useful applications:
    - Mapping deformations resulting from articulated movements on one mesh (which maybe I "know" exactly because I ran it through an FEM simulator) to another (similar) mesh that I don't want to simulate exactly. Their headline example: I simulated the deformation of a horse, now I want a camel to bend his legs in a similar way. And who wants to precisely simulate a camel?
    - Mapping facial expressions from one person to another or to a virtual model. This of course requires that I have a triangulated representation of the "known" face, which maybe I get from range data.
  - Another example would be our long-term goal of taking Phil's range data and applying it to our soft-body simulations. So maybe something *like* a liver deforms in a known way, and I want to map that deformation onto a liver deformed with similar forces.
- We'll use the term "source mesh" to refer to the mesh whose deformations we know exactly (from FEM simulation, range data, etc), and the term "target mesh" to refer to the mesh whose base

## Correspondence

- First they have "the user" (probably an artist) pick a few pairs of points on the two models that should undergo very similar deformations. Basically these are feature maps, mapping – for example – the horse's nose to the camel's nose.
- Now they have two meshes with a few known correspondence points, and they want to find out which triangles on the "target" (unknown) mesh should "deform like" specific triangles on the source mesh.
  - Note that I'm breaking with their terminology here... if I ultimately know the deformation of the horse and want the deformation of the camel, I always call the horse "source" and the camel "target". They reverse that terminology for the correspondence section of the paper.
- They solve this problem by finding an affine transformation for *every* target mesh triangle that makes the mesh "look like" the source mesh. Then they'll say that target triangles should "deform like" whatever source triangles they end up near.

- Important to note that they are solving for a transform for each *triangle*, not just each vertex
- They're going to formulate an optimization problem with the following minimization terms:
  - *Smoothness*: Adjacent target mesh triangles should be transformed in *about* the same way to get to the source mesh
  - *Deformation identity*: Triangles should be transformed as little as possible, so the transformed target mesh still looks generally like what it started out as (i.e. the camel still looks more or less like a camel)
  - *Closest valid point*: Vertices on the target mesh should be transformed toward the "closest valid point" on the source mesh. Basically this means "closest point"... i.e. this term pushes the meshes together.
- And of course they add the hard constraint that the user-specified "match points" should line up exactly on the source and target meshes after transformation.
- Each of the minimization terms is assigned a weight, describing their relative influences over the final solution.
- They choose an empirically determined series of weights, and they solve this constrained minimization problem (using whatever solver they find the web, or Matlab fsolve, or whatever) a few times to get a final result.
  - Importantly, they start with no closest valid point term, because the initial closest point to any target mesh point isn't meaningful.
- After they have a set of transformations to be applied to each triangle on the target mesh to get it to look like the source mesh, they can figure out which target triangle should "deform like" each source triangle.
  - In fact, the output of this step is a list of "matching" triangles... for each transformed target triangle, they say that it "matches" the closest source mesh triangle. And for each source mesh triangle, they say that it "matches" the closest transformed target mesh triangle (pruning duplicates for efficiency).
  - Note that some triangles appear more than once, that will be fine...
  - Also note that the "closest" triangle is really the "closest reasonable" triangle, so triangles with hugely different normals can't be paired up

## Deformation Transfer

- Okay, now I have a list of pairings between triangles in a source mesh and a target mesh in their rest positions. And I've simulated or acquired some deformation on the source mesh. Now the real work is figuring out where to put the target mesh triangles so it looks like it's undergone the same deformation...
- We want to formulate this as a least-squares minimization problem... we want to somehow encode the deformation applied to the source mesh and say "find the target vertex configuration that makes the matched triangles line up best"
- But they don't want to place any constraints on *absolute* vertex or triangle *translation*, since the meshes may be of totally different scales.
- So they just want to compute a transform-free affine transform that was applied to each source *triangle*, and apply the same affine transforms to each corresponding target triangle. It's interesting to think about the fact that this can describe the overall deformation without any translational component.

- Describing such a transform requires a 3x3 matrix, so they'll need 9 “before” values and 9 “after” values to solve for the transformation, all of which should be independent of translation.
- So their formulation of the minimization problem will include – for each triangle – a fourth “imaginary” vertex that sits “above” one vertex of the triangle (where “above” means “along the normal to the triangle face at that vertex”) :

$$\mathbf{v}_4 = \mathbf{v}_1 + (\mathbf{v}_2 - \mathbf{v}_1) \times (\mathbf{v}_3 - \mathbf{v}_1) / \sqrt{|(\mathbf{v}_2 - \mathbf{v}_1) \times (\mathbf{v}_3 - \mathbf{v}_1)|} \quad (1)$$

- They compute the vectors from vertex  $v_1$  to each of the other *three* vertices (including the imaginary, computed one), which gives us 3 3-vectors in the “before” case, and 3 3-vectors in the “after” case. With a simple 3x3 inversion, they have a 3x3 matrix that describes what happened to the triangle, independent of translation :

$$\begin{aligned} \mathbf{V} &= [\mathbf{v}_2 - \mathbf{v}_1 \quad \mathbf{v}_3 - \mathbf{v}_1 \quad \mathbf{v}_4 - \mathbf{v}_1] \\ \tilde{\mathbf{V}} &= [\tilde{\mathbf{v}}_2 - \tilde{\mathbf{v}}_1 \quad \tilde{\mathbf{v}}_3 - \tilde{\mathbf{v}}_1 \quad \tilde{\mathbf{v}}_4 - \tilde{\mathbf{v}}_1] \end{aligned} \quad (3)$$

$$\mathbf{Q} = \tilde{\mathbf{V}}\mathbf{V}^{-1}. \quad (4)$$

- $\mathbf{Q}$  is the transformation for a given triangle, which for some reason they decide to call  $\mathbf{S}$  immediately after they call it  $\mathbf{Q}$ .
- So I want to download my favorite optimization problem solver from the web and tell it :
  - Move the target vertices wherever you like, but try to minimize the absolute difference between the norms (matrix magnitudes) of the  $\mathbf{S}$ 's (deformations) for all matching source and target triangles:

$$\min_{\tilde{\mathbf{v}}_1 \dots \tilde{\mathbf{v}}_n} \sum_{j=1}^{|M|} \|\mathbf{S}_{s_j} - \mathbf{T}_{t_j}\|_F^2.$$

Here  $\mathbf{v}$  are the vertex positions,  $\mathbf{S}$  are the source transformations,  $\mathbf{T}$  are the target transformations,  $M$  is the set of matched (corresponding) triangles, and  $F$  means “Frobenius norm”, which just means taking the sum of the squares of a matrix.

- It is not clear to me that minimizing the absolute differences between transformation matrices is hugely meaningful, but I don't have a better way to do it.
- In order to make this problem look familiar and in order to be able to solve it in a standard way, they build a matrix formulation :

$$\min_{\tilde{\mathbf{v}}_1 \dots \tilde{\mathbf{v}}_n} \|\mathbf{c} - \mathbf{A}\tilde{\mathbf{x}}\|_2^2$$

Here  $\mathbf{c}$  is a vector containing all the values from the source triangle transformations (thus it's length  $9 * \text{the number of triangle pairings}$ ),  $\tilde{\mathbf{x}}$  is what we're solving for: the deformed positions of the target mesh vertices, and  $\mathbf{A}$  is a big matrix that relates  $\tilde{\mathbf{x}}$  to  $\mathbf{c}$ . The columns of  $\mathbf{A}$  correspond to target vertices, and the rows represent triangle pairings.  $\mathbf{A}$  is zero everywhere that a given triangle pairing doesn't affect (constrain) a given vertex position.

- This is a classic least-squares optimization problem, and the solution is given as :

$$\mathbf{A}^T \mathbf{A} \tilde{\mathbf{x}} = \mathbf{A}^T \mathbf{c} \quad (10)$$

- This equation can be solved *once* for a given  $\mathbf{A}$  (i.e. we can find the inverse of  $\mathbf{A}^T \mathbf{A}$  just once), and that solution can be re-used as  $\mathbf{c}$  changes. In other words, I can solve once for the correspondence between the horse and the camel, and I can plug in new horse postures to get new camel postures and I'll only need to perform matrix multiplies.

### Limitations and Possible Extensions

All in all this seemed like an excellent paper with convincing results. They didn't try to hide the limitations on their method or the amount of manual labor required, so I would expect to see more work from them on automation of the process. Here are some limitations and/or ideas for extensions that might be appropriate to our work :

- Connectivity is strictly enforced in this paper; they do not handle topology changes. One might imagine using a similar technique to map cuts, fractures, etc. from one mesh onto another, particularly during deformation.
- The use a relatively naïve and exhaustive triangle mapping, with no way of indicating that some regions are "more important" than others.
- In fact they also allow the source  $\leftrightarrow$  target triangle mapping to be non-invertible, which could cause strange effects in regions with lots of high-frequency activity. They do show results on mostly smooth meshes; it's not clear that this method would always handle corners or high geometric detail well.
- Their method requires user-selected correspondence points, and is in fact very sensitive to errors in that assignment, since it treats those matchings as *hard* constraints... a very logical extension would be to build an automated matching system, that searched the two meshes for similar geometric or visual (texture-based) features and at least proposed possible correspondences.
- Their method does not preserve volume or internal parameters (i.e. compression or strain), which might be as important to some modeling applications as preserving surface similarity.

- Even for “semantically similar” meshes (e.g. a horse and a camel), you need the same “base pose” for both meshes, which might not always be available.
- A nice approach for our work might be to build a library of deformations on interesting surface geometries and material properties using range-scanning or FEM simulation. When computing a deformation in real-time, we could index into that library to find the most appropriate match to the *local* geometry (and possibly material) with which a user is interacting. Lots of work would have to be done in searching, interpolating, computing or estimating required internal material properties (e.g. computing volume compression), and incorporating haptic feedback, but this might be a useful approach for turning a deformation-capture or offline-deformation-simulation tool into a library for real-time presentation.