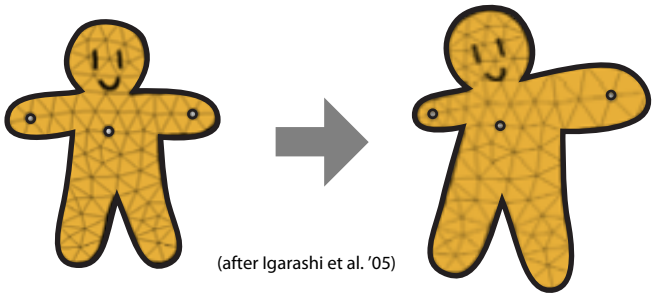
CS 3220 Spring 2010
Homework 7 (v. 1.1)



This homework is about using linear least squares to compute deformations to 2D shapes, for instance in computer-aided design or animation. Suppose we have a 2D shape represented by a collection of triangles in the plane. This means we store a set of n 2D points as the rows of a $n \times 2$ matrix $\mathbf{X}$ and a set of m triangles in an $m \times 3$ matrix $\mathbf{T}$ full of integers: each row contains the indices (row numbers in $\mathbf{X}$) of the three vertices of a triangle.

To edit the shape, the user specifies new positions for some vertices; put the indices of these vertices in a p-vector of integers called $\mathbf{f}$, and their new locations are in the $2 \times p$ matrix $\mathbf{Y}$. Our job is to compute the minimum deformation of the whole shape that matches those constraints, by computing new positions $\mathbf{X}'$ for all the vertices. (The constraints say that $\mathbf{X}'(f_i, :) = \mathbf{Y}(i, :)$.) You can think of the shape as being cut out of a sheet of rubber; we stick pins in some places to control their positions, and the rubber then takes on the shape that minimizes deformation.

There are many ways to compute such a shape, depending on how one measures the degree of deformation. If the deformation measure can be expressed as a sum of squares of linear combinations of the variables, then finding the deformation is a linear least squares problem, making it particularly simple and efficient to solve. Concocting functions that are linear but still have the desired properties is a bit of an art form, and in this homework we use such a measure taken from a recent paper on real-time shape editing [1].[1] It corresponds to a rather odd kind of rubber that

[1] This mapping by itself is not great for shape editing, because it creates a lot of scaling, but with a couple of other least squares problems it becomes a very good method.

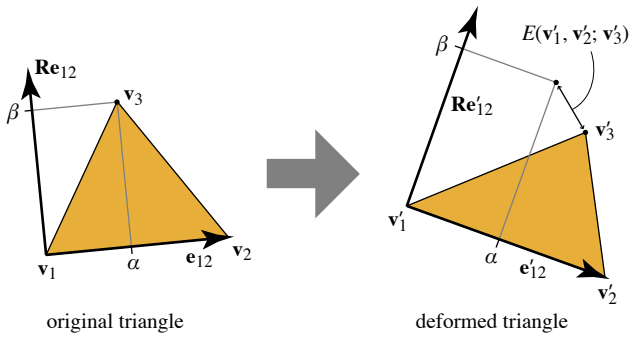


Figure 1: Computing the deformation measure for one vertex of a triangle relative to the opposite edge.

is free to rotate or to stretch uniformly—only changes in shape, not size, count as deformation.

The deformation of a single triangle is a sum of three terms, one measuring how much each vertex has moved relative to the opposite edge. To measure how much the vertex moved, we use its projections onto the opposite edge and onto a perpendicular to that edge, and see how well they predict the new location—if the projections predict the new location exactly, then the triangle is "the same shape."

Looking at Figure 1, call the original vertices of a triangle $\mathbf{v}_1$, $\mathbf{v}_2$, and $\mathbf{v}_3$, and the new locations $\mathbf{v}'_1$, $\mathbf{v}'_2$, and $\mathbf{v}'_3$. (That is, $\mathbf{v}_1$ is a name for $\mathbf{X}(\mathbf{T}(j,1),:)$, and so on, where j is the index of the triangle in question.) Define the edge $\mathbf{e}_{12} = \mathbf{v}_2 - \mathbf{v}_1$. Using $\mathbf{e}_{12}$ and the perpendicular vector $\mathbf{Re}_{12}$ (where $\mathbf{R} = [0\ \ -1; 1\ 0]$ is a rotation of 90 degrees) as a basis, the coordinates of $\mathbf{v}_3$ are:

$$\alpha = (\mathbf{v}_3 - \mathbf{v}_1) \cdot \frac{\mathbf{e}_{12}}{\|\mathbf{e}_{12}\|^2} \quad \text{and} \quad \beta = (\mathbf{v}_3 - \mathbf{v}_1) \cdot \frac{\mathbf{Re}_{12}}{\|\mathbf{e}_{12}\|^2}$$

When $\mathbf{v}_1$ and $\mathbf{v}_2$ move to $\mathbf{v}'_1$ and $\mathbf{v}'_2$, we expect to find $\mathbf{v}'_3$ at

$$\mathbf{v}'_1 + \alpha\mathbf{e}'_{12} + \beta\mathbf{Re}'_{12}$$

and the deformation for that vertex is

$$E_j(\mathbf{v}'_1, \mathbf{v}'_2; \mathbf{v}'_3) = \left\|\mathbf{v}'_1 + \alpha\mathbf{e}'_{12} + \beta\mathbf{Re}'_{12} - \mathbf{v}'_3\right\|$$

In this formula the vertices $\mathbf{v}_{1,2,3}$ are implicitly the three vertices of triangle j.

The total deformation for the whole shape is just the sum over all three vertices of all m triangles:

$$E^2 = \sum_{j=1}^{m} \left(E_j(\mathbf{v}'_1, \mathbf{v}'_2; \mathbf{v}'_3)^2 + E_j(\mathbf{v}'_2, \mathbf{v}'_3; \mathbf{v}'_1)^2 + E_j(\mathbf{v}'_3, \mathbf{v}'_1; \mathbf{v}'_2)^2\right)$$

If you look at how E depends on the deformed vertex positions in $\mathbf{X}'$, you can see that this is a linear least squares problem. (Remember that the $\mathbf{v}$s and $\mathbf{v}'$s are just names for the rows of $\mathbf{X}$ and $\mathbf{X}'$.) Each of the three terms inside the sum contributes two rows (for the x and y differences in position) for each triangle. So minimizing E^2 is a least-squares problem with $2(n-p)$ variables (the positions of all the unpinned vertices) and $6m$ rows (two for each vertex of each triangle).

Problem 1: Implement this method of minimum-deformation shape editing by writing two Matlab functions:

```
[A,b] = deformation_setup(X_old, T, f);
X_new = deformation_solve(A, b, f, Y);
```

The "setup" function takes the connectivity information and sets up the problem, building the matrix and the fixed parts of the right hand side. It is called once after the pinned points have been chosen but before the editing operation is started. The "solve" function is called after each change to the position of the pinned points. It's fine to use the backslash where applicable. These two functions will be called by the provided framework, which implements an interactive system.

Problem 2: For Problem 3 you'll need the ability to solve least squares problems with linear constraints. That is, you need to solve

$$\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\| \quad \text{subject to} \quad \mathbf{C}\mathbf{x} = \mathbf{d}.$$

where $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{C} \in \mathbb{R}^{p \times n}$, and $p < n < m$.

(a) A basis for the null space of $\mathbf{C}$ is useful in this computation. Write a line or two of Matlab code that starts with a matrix $\mathbf{C}$ and computes a matrix $\mathbf{N}$ such that $\text{span}(\mathbf{N}) = \text{null}(\mathbf{C})$.

(b) Any solution for $\mathbf{C}\mathbf{x} = \mathbf{d}$ is $\mathbf{x}_0 + \mathbf{N}\mathbf{z}$ for some $\mathbf{z} \in \mathbb{R}^{n-p}$. Plug this solution into the least squares system to get a new least squares system. Write a line or two of Matlab code to compute the matrix and right-hand side of the new system, starting from $\mathbf{A}$, $\mathbf{b}$, $\mathbf{d}$, and the results of the previous part.

(c) Use your answers to write a Matlab function with the signature

```
x = constrained_lls(A, b, C, d);
```

It's fine to use the backslash where applicable.

(d) Demonstrate your function on a randomly generated 5×4 system with the first variable constrained to the value 1.0 and the average of the second and third variables constrained to 2.0.

Problem 3: The Problem 1 version of the shape editor only supports pinning of vertices, not arbitrary points inside triangles. Any point inside a triangle with vertices $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ can be written as a linear combination $\alpha\mathbf{v}_1 + \beta\mathbf{v}_2 + \gamma\mathbf{v}_3$, with $\alpha + \beta + \gamma = 1$. A more refined version of this system replaces $\mathbf{f}$ and $\mathbf{Y}$ with a p-vector $\mathbf{g}$ listing the triangle index for each pin, a $p \times 3$ matrix $\mathbf{G}$ which has α, β, and γ for the kth pin on row k, and $\mathbf{Y}$, which as before has the desired position of the pinned point. The constraint defined by one row looks like this:

$$\alpha\mathbf{v}'_1 + \beta\mathbf{v}'_2 + \gamma\mathbf{v}'_3 = \mathbf{y}$$

where for the kth constraint $\mathbf{v}'_{1,2,3}$ are respectively $\mathbf{X}(t(g_k, 1), :)$, $\mathbf{X}(t(g_k, 2), :)$, and $\mathbf{X}(t(g_k, 3), :)$ and $\mathbf{y}$ is $\mathbf{Y}(k, :)$. So the p pins provide $2p$ linear equations involving the elements of $\mathbf{X}'$ that need to be satisfied while we solve for $\mathbf{X}'$—this is a set of linear constraints of the type dealt with in Problem 2.

For this problem, modify your code from Problem 1 to support this new kind of constraint, creating functions with signatures:

```
[A,b,C,d] = deformation_setup2(X_old, T, g, G, abg);
x_new = deformation_solve2(A, b, C, d, g, Y);
```

References

[1] Takeo Igarashi, Tomer Moscovich, and John F. Hughes. As-rigid-as-possible shape manipulation. *ACM Transactions on Graphics*, 24(3):1134–1141, August 2005.