Mesh Editing with δ-coordinates

April 5, 2003

General

The purpose of the following document is to introduce you to the mathematical tool that you will implement for editing meshes. In the following, we will denote the given triangle mesh by $M = (V, E)$ where V is the set of vertices and E is the set of edges. M represents a 2-manifold, that is, the graph of M is planar (each triangle in the mesh has at most two neighbours).

1 The δ-coordinates and Laplacian matrix

Let M be a given triangular mesh with n vertices. Each vertex $i \in M$ is conventionally represented using absolute Cartesian coordinates, denoted by $v_i = (x_i, y_i, z_i)$. We define the *relative* or *δ-coordinates* of v_i to be the difference between the absolute coordinates of v_i and the center of mass of its immediate neighbors in the mesh,

$$\delta_i = (\delta_i^{(x)}, \delta_i^{(y)}, \delta_i^{(z)}) = v_i - \frac{1}{d_i} \sum_{k=1}^{d_i} v_{i_k} \ ,$$

where d_i is the number of immediate neighbors of i (the degree or valence of i) and i_k is i's k th neighbor ($(i, i_k) \in E$).

The transformation of the vector of absolute Cartesian coordinates to the vector of relative coordinates can be represented in matrix form. Let A be the adjacency (connectivity) matrix of the mesh:

$$A_{ij} = \begin{cases} 1 & (i,j) \in E \\ 0 & \text{otherwise,} \end{cases}$$

and let D be the diagonal matrix such that $D_{ii} = d_i$. Then the matrix transforming the absolute coordinates to relative coordinates (scaled by D) is $L = D - A$,

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

That is, $Lx = D\delta^{(x)}$, $Ly = D\delta^{(y)}$, and $Lz = D\delta^{(z)}$, where x is an n-vector containing the x absolute coordinates of all the vertices (that is, $x = (x_1, x_2, \ldots, x_n)^T$) and so on. Without loss of generality, we now focus on the vectors x and $\delta = D\delta^{(x)}$.

The matrix L is called the *Laplacian* of the mesh. The Laplacian is symmetric, singular and positive semidefinite. The singularity stems from the fact that the system $Lx = \delta$ has an infinite number of solutions which differ from each other by a vector that is constant on each connected component of the mesh. Note that the elements of each row of L sum up to zero. Thus, we can actually recover x from δ if we know, in addition to δ, the Cartesian coordinate of one x_i in each connected component. The known vertex position is called *anchor point*.

2 Anchor points

Assume for rest of this document that we are working with a mesh that contains one connected component. Suppose we have computed the δ-coordinates of our mesh. We can now use them to manipulate the shape of the mesh in the following way: we choose a vertex to be a special anchor point and alter its position (that is, change its Cartesian coordinates). Then, we restore the Cartesian coordinates of the rest of the mesh by solving for x (and y, z): $Lx = \delta^{(x)}$. However, L is singular, so we need to use our anchor point to find a unique solution. We *add* the information about the anchor as an additional row in the L matrix. The new row will contain 1 at the anchor vertex position and 0 everywhere else (see Figure 1). We also add the x coordinate of the anchor to the solution vector δ.

What we obtain is a new linear system: $\tilde{L}x = \tilde{\delta}$, where $\tilde{L}$ is the extended L matrix and $\tilde{\delta}$ is the extended solution vector. Now the system contains more equations than variables, and it can be uniquely solved in the least-squares sense. That is, there is a unique x that minimizes $||\tilde{L}x - \tilde{\delta}||$. It can be easily shown that $x = \left(\tilde{L}^T\tilde{L}\right)^{-1}\tilde{L}^T\tilde{\delta}$.

You may ask why not simply substitute the anchor point and solve the usual linear system $Lx = \delta$. The answer is that this way we loose the “smoothness” condition at the anchor vertex because we essentially remove from the system the equation $\delta_i = v_i - \frac{1}{d_i}\sum_{k=1}^{d_i} v_{i_k}$, for i that is anchor vertex. This way the resulting mesh will have a "spike" at the anchor vertex. On the other hand, the least-squares solution keeps that equation and thus the resulting mesh should be smooth at the anchor as well.

3 Editing using extended Laplacian matrix

When only one anchor vertex is used, and we move it around, the entire mesh surface will move. Thus, this produces a global effect. If we want to make more local changes, we need to "nail" down the parts of mesh that we do not wish to change. This can be done by adding more anchor vertices to the system, whose positions we do not change (unlike the special anchor that we do move). This way, our system will become "more rectangular". In the extreme case, when we want to make only a small local change, we constrain almost all the vertices of the mesh, and obtain a system that is nearly $2n \times n$. On the other hand, most of the vertices are "known", that is, we have a good initial guess for the solution.

The editing system should define a certain hierarchy over the vertices of the mesh, and let the user to choose which level in the hierarchy he wants. The user is allowed to move the vertices that belong to the current level, and they serve as anchors (all the rest of the vertices are not constrained). So, in the coarsest level we have few anchors and a global shaping is achieved. In the finest level, all the vertices are anchors, and a local editing effect is achieved.

You can see that in the above suggestion the hierarchy over the vertices serves only as means to choose the set of anchors. You may want to think of a way to exploit a real mesh hierarchy to speed up the computation. A real hierarchy means to build several levels of detail for the mesh, such as a *progressive mesh* [1].

For implementing the editing system you need a least-squares solver. There are many packages available, such as the one developed by Prof. Sivan Toledo and his group (you can find the link on the assignment homepage). You can also import code from MATLAB. Note that when you solve a "very rectangular" system, you may want to use an iterative solver because you can provide it with a very good initial guess (you know the positions of all the constrained vertices, and they are a good approximation for the final solution). This may considerably speed up the computation on the finer levels.

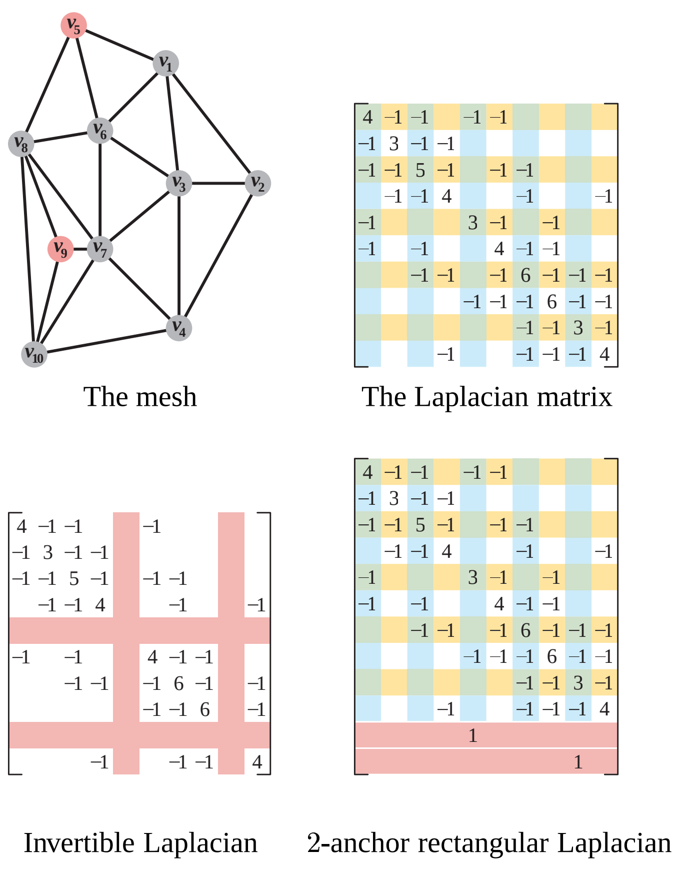

Figure 1: A small example of a triangular mesh and its associated Laplacian matrix (top right). Second row: a 2-anchor invertible Laplacian and a 2-anchor rectangular Laplacian. The anchors are denoted in the mesh in red.

References

[1] Hugues Hoppe. Progressive meshes. In *Proceedings of SIGGRAPH 96*, Computer Graphics Proceedings, Annual Conference Series, pages 99–108, August 1996.