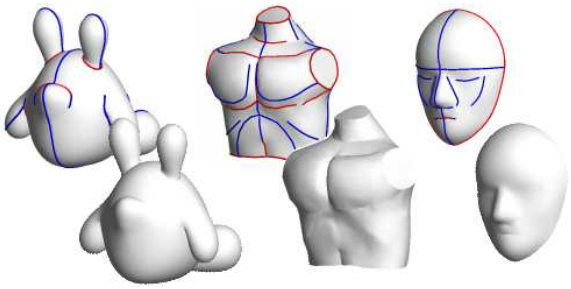
**CS224**: Interactive Computer Graphics

# FiberMesh

*Due: 4/12/10, 11:59 PM*



| | |
|---|---|
| Least Squares Solver | 5 |
| Mesh triangulation | 10 |
| Optimization Problem 1: Curve Dragging | 25 |
| Optimization Problem 2: Surface Optimization | 35 |
| Adding New Control Curves | 15 |
| UI Quality | 10 |
| Extra Credit | +25 |
| **Total** | **100** |

This assignment is new / experimental, and its worth towards your final grade has yet to be determined.

# 1 Introduction

In this assignment, you will be implementing FiberMesh:

Andrew Nealen, et. al, "FiberMesh: Designing Freeform Surfaces with 3D Curves" http://www-ui.is.s.u-tokyo.ac.jp/ takeo/papers/fibermesh.pdf

You should already be familiar with the paper from the FiberMesh HW, and the following papers may also provide useful references:
Sorkine, O. 2004. Differential Coordinates for Interactive Mesh Editing.
- http://www.cs.nyu.edu/ sorkine/ProjectPages/Editing/

Andrew Nealen's Ph.D. Thesis:
- http://opus.kobv.de/tuberlin/volltexte/2007/1714/pdf/nealen_andrew.pdf

This assignment will be difficult and time consuming; it is recommended that you start early.

# 2 Requirements

This project can roughly be broken down into three parts: surface creation / optimization, curve deformation, and adding additional control curves. The first two are fairly independent, and you can therefore implement them in either order you'd like. As a note, the TAs implemented curve deformation before surface optimization because the optimization problem itself is simpler.

## 2.1 Surface Creation and Optimization

This is where you get to put the "Mesh" in FiberMesh.. The skeleton we give you comes with bare-bones example code for responding to mouse events and recording and drawing user-drawn curves. After the user finishes drawing a curve, you must:

- Resample the curve regularly (your meshes will look awful if you don't).
- Compute a bunch of interior points and use Delaunay triangulation to create a mesh. (see MeshUtils::triangulate in the FiberMesh support code which will perform Delaunay triangulation for you) Remember that you'll need to duplicate triangles, being careful not to duplicate boundary vertices, so that our inflated mesh will not be flat.
- Project the curve / mesh into world-space.

- Inflate / optimize the mesh by repeatedly applying the 3 optimization problems listed in the "Surface Optimization" section of the paper (equations 6, 8, and 10). We recommend starting off without using the iterative solution and just trying to get your mesh to inflate just a little bit after one iteration (which will probably take awhile in and of itself), and only then, moving on to the iterative solution which will actually converge to a stable solution (hopefully). Pay particularly close attention to exactly what's supposed to change in terms of constraints in the first iteration versus all of the subsequent iterations. Keep in mind that things Gouraud surface normals won't make much sense for the boundary vertices during the first iteration.

## 2.2 Curve Deformation

Once a curve exists, you must make sure it deforms smoothly when the user grabs and drags it. This means solving the optimization problem discussed in the "Curve Deformation" section of the paper. **NOTE**: you are not required to implement the rotation part of curve deformation. Dealing with rotations when deforming a control curve is extra credit. To be explicit, we expect you to solve the following optimization problem:

$$\min\{\sum_{i \in V} ||L(v)_i - \delta_i||^2 + \sum_{i \in V \setminus \text{ROI}} ||v_i - v_i'||^2\} \quad (1)$$

Note that this is the same as equation 1 in the FiberMesh paper without the rotations.

There are no strict requirements on how your curve drawing/deformation interface has to work, but here's a reasonable approach.

- When the user clicks down, if not near any existing curve vertices, begin drawing a new curve.
- Otherwise, when the user clicks down on or near an existing curve vertex, enter curve deformation mode.
- Select the nearest vertex to the user's click as the "handle" vertex that the user moves directly.
- On each mouse drag, move the "handle" vertex and reoptimize the curve. You'll also want to reoptimize the attached surface once you've implemented the surface optimization portion of FiberMesh.
- End when the mouse button is lifted. Note: during dragging, you may want to optimize the attached surface cheaply by not iterating as much as you would normally, and then only update the surface completely by iterating a lot once the mouse is done dragging.

You can perform mesh creation and surface optimization immediately after a curve is drawn, or you can assign controls to each phase;
e.g. press one key to generate the mesh, and another to perform the surface optimization. For debugging, it may be helpful to have a control to perform one iteration of the optimization problem. That way, you can "step through" the optimization and watch your mesh inflate.

Again, you don't have to follow this scheme exactly, as long as you do something reasonable and explain it in your README.

### 2.3 Additional Control Curves

Once you have surface optimization and curve deformation working, it's time to allow the user to draw new control curves onto the surface of a pre-existing mesh. This is harder than it sounds. Taking additional curves into account during future surface optimizations isn't that bad, but defining which mesh vertices should be bound to your newly drawn curve, assuming that it was drawn entirely on the surface of a mesh, can get messy.

There are multiple ways of binding a newly drawn curve to a mesh's underlying vertices; Nealen et al handle new control curves by subdividing triangles to create new triangles whose vertices match up with points in your curve (Nealen et al's demo implementation is available as a downloadable Java app online on Nealen's website).

An alternative way to bind newly drawn curve vertices to mesh vertices is to connect adjacent curve vertices via their shortest path along the surface of the mesh, throwing out duplicate mesh vertices and along the way.

It is up to you how you choose to implement the handling of additional control curves, but at the end of the day, the user should be able to draw a curve which resides completely on the surface of an existing mesh, and a new constraint curve should be bound to that surface in some sane fashion. Be sure to explain how you chose to handle additional control curves in your README.

## 3 Support Code

Support code is located at: `/course/cs224/asgn/fibermesh`.

A demo executable is available at: `/course/cs224/demo/fibermesh`. Use the CTRL+mouse buttons to rotate, pan, and zoom for left, middle, and right

mouse button respectively. You are by default in 'draw' mode, and drawing a closed-ish curve should create a triangulated surface. Press the spacebar to inflate / optimize the surface. Press '2' to switch to 'drag' mode to deform curves and press '1' to go back to draw mode. Keep in mind that this was a quickly hacked up UI for demonstration purposes only, and we'd like to expect your UI to be a bit more intuitive.

The support code uses a small subset of the Milton codebase, namely Vector, Point, and Mesh. We've tried to provide you with as minimal support code as possible so you can concentrate your effort on implementing FiberMesh. As with every assignment in CS224, our support code is strictly for your convenience, and you are more than welcome to ignore it entirely if you wish.

Here is a breakdown of the classes that we give you:

- *Frontend* is where all the magic happens. It responds to mouse and key events and handles OpenGL (and non-OpenGL) drawing. The sketon version of Frontend that we give you contains a bunch of places marked **TODO**, with specific explanations of what's expected or notes to be aware of. This will be the main entrypoint for all of the code you write in this project, though it is up to you how you design your FiberMesh engine (we recommend separating the deformation engine logic out into a separate class and leaving Frontend as Gui-oriented as possible). The skeleton Frontend we give you also contains basic, Maya-inspired camera interactions via CTRL+mouse buttons, similar in effect to the trackball rotate, pan, and zoom interactions in the Modeler assignment from CS123.
- *LinearSolver* is a wrapper around UMFPack for efficiently solving sparse linear systems of the standard form Ax=b, including a cache of the factorization of A=LU such that repeated invocations of solving Ax=b with the same A=LU can be performed in nearly linear time by back-substitution of LUx=b. Fun bit of trivia: UMFPack is the hardcore C library that Matlab invokes under the hood when you solve $A\backslash b$.
- *SparseMatrix* provides basic functionality for a reasonably efficient, easy-to-use sparse matrix class that should look and feel very similar to the `CS123Matrix` and Milton's `Matrix<M, N, T>` that you may already be familiar with.
- *MeshUtils* is a handy set of utility routines we've provided that you'll definitely want to take a look at. MeshUtils contains thoroughly tested / efficient implementations of several key routines that you will likely need along the way, though it may not be completely obvious at first where each will be used. In particular, MeshUtils::getNeighbors (for finding connectivity information about a mesh), MeshUtils::triangulate, MeshUtils::isPointInPolygon, and MeshUtils::getWorldVertex will also come in very handy, and depending on how you implement drawing curves onto pre-existing surfaces,

MeshUtils::computeVertexShortestPaths may also come in handy. See the inline documentation in MeshUtils.h for more info.

# 4 Extra Credit

- Rotational constraints for curve deformation (see equation (2) in FiberMesh)
- Any additional interaction techniques presented in the paper besides the fairly easy 'erasing' technique (eg, extrusion, change-curve-type, creating holes in a mesh, etc.)

# 5 Handing In

Besides the usual README stuff (compiling instructions, bugs, extra credit, usage, etc.), please identify where your code is for the following:

- Least Squares Solver
- Both Optimization Problems
- Any extra credit

Run the following command from within your source directory to handin your code: `/course/cs224/bin/cs224_handin fibermesh`