CS 177 - Fall 2008

Homework 2: Smoothing

1 Minimizing the Area of a Discrete Surface

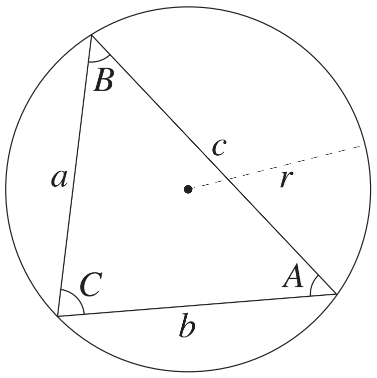
1.1 A Few Identities

1.1.1 Law of Sines

Let A, B, C and a, b, c be the angles and the side lengths of a triangle, respectively. Additionally, let r be the radius of the triangle's circumcircle. Show that the so-called *law of sines* holds, namely

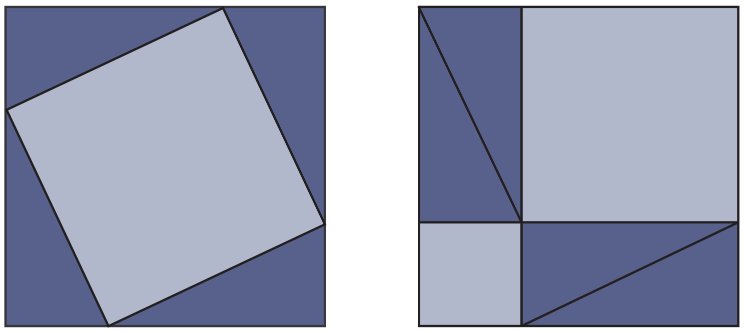
$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C} = 2r.$$

(Hint: use the fact that an inscribed angle equals half of its intercepted arc.)



1.1.2 Pythagorean Theorem

Briefly explain how the following figure justifies the Pythagorean theorem.



1.1.3 Cotangent

Give a simple geometric argument for the Pythagorean identity

$$\sin^2\alpha + \cos^2\alpha = 1$$

and use it to show that the following trigonometric identity holds:

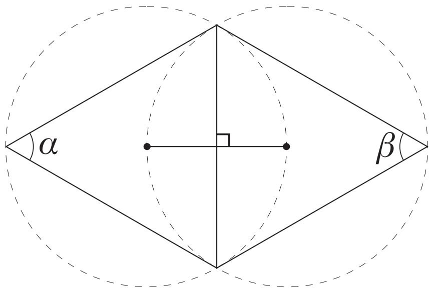
$$\cot\alpha = \frac{\sqrt{1-\sin^2\alpha}}{\sin\alpha}.$$

1.2 Length of a Dual Edge

Consider a pair of triangles sharing a common edge. Connecting the circumcenters of these triangles yields the *circumcentric dual edge*. Show that the dual edge is orthogonal to the primal (i.e., original) edge and has (signed) length

$$\frac{1}{2}(\cot\alpha + \cot\beta)\,l$$

where α and β are the angles opposite the primal edge and l is the length of the primal edge. *(Hint: use your results from above!)*



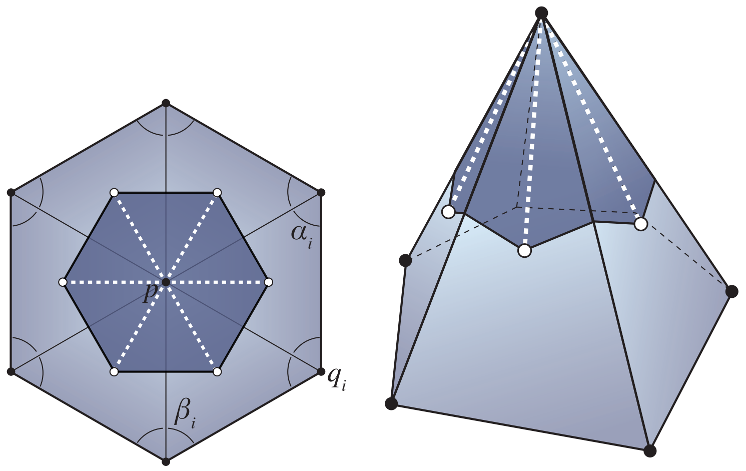
1.3 Gradient of Triangle Area

Let p be the vertex of a triangle and let $\mathbf{u}$ be the vector along its opposing edge. Argue geometrically that the gradient of the area of this triangle with respect to p is given by

$$\nabla_p A = \frac{1}{2}\mathbf{u}^\perp$$

where $\mathbf{u}^\perp$ is the edge vector rotated by an angle $\pi/2$ in the plane of the triangle such that it points toward p. *(Hint: this was done in class!)*

1.4 Gradient of Voronoi Area



Consider the collection of triangles around a vertex p in a simplicial surface. We can form the *Voronoi region* of p by connecting consecutive circumcenters of these triangles along the surface. Additionally, we can tessellate this Voronoi region by connecting p to each circumcenter, as pictured above (left). In particular, consider how the area of the Voronoi changes as we move p, and show that this gradient is equal to

$$\nabla_p = \frac{1}{4}\sum_i (\cot\alpha_i + \cot\beta_i)(p - q_i)$$

with quantities labeled as in the figure above[1]. If we normalize this gradient by the area of the Voronoi region, the resulting quantity corresponds to the *discrete mean curvature normal* of a simplicial surface, which will be large in "spiky" regions and vanish in flat regions. Unlike the discrete *Gaussian* curvature seen in Homework 1, it will also be non-zero in *developable regions*, i.e., regions that can be unfolded into the plane without distortion.[2]

(Hint: use your results from above!)

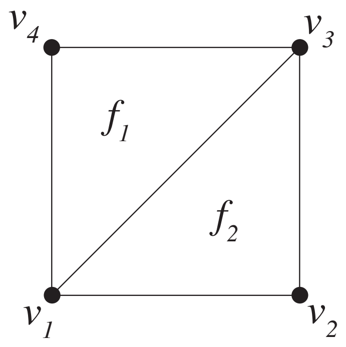
2 Curvature Flow

The curvature derived in the previous section can be used to remove noise from a discrete surface. The basic idea is to push vertices against the direction of the mean curvature normal. Hence, where curvature is large the surface will quickly flatten out, and regions that are already flat will stay flat. In this assignment you will implement mean curvature flow in MATLAB. You may find it *very helpful* to look at the paper Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow by Desbrun et al.

If you do not have access to MATLAB, you can also use GNU Octave, which is a freely available alternative. However, you may find that Octave is too slow to run some of the larger example meshes.

2.1 Mesh I/O

Polygonal surfaces are often stored in a simple text format that specifies vertex positions in terms of their (x, y, z) coordinates and faces as a list of vertex indices (called the *Wavefront OBJ* format, or simply *OBJ*). An example mesh and its specification are shown below.



```
v 0 0 0
v 1 0 0
v 1 1 0
v 0 1 0
f 1 3 4
f 1 2 3
```

Your ability to read and write mesh files will be essential to testing and debugging your smoothing code.

2.1.1 Reading

Write a MATLAB routine that reads an OBJ file into a list of vertices and faces, starting with the template below. You should need nothing more than the basic file I/O functions (`fopen`, `fscanf`, etc.).

[1]There is a subtlety here involving the fact that the circumcenters will also move relative to p – for the purposes of this assignment you may assume that the circumcenters remain fixed.

[2]You may find it interesting to think about whether the notion of a normal field is well-defined on non-orientable surfaces.

```
  function [V,F] = read_mesh( filename )
% function [V,F] = read_mesh( filename )
%
% Loads a mesh in Wavefront OBJ format.
%
% INPUT:
%    filename - path to mesh file
%
% OUTPUT:
%    V - dense 3x|V| matrix of vertex positions
%    F - dense |F|x3 matrix of faces as 1-based indices into vertex list
%
```

2.1.2 Writing

Write a MATLAB routine that writes an OBJ file to disk following the template below.

```
  function write_mesh( filename, V, F )
% function write_mesh( filename, V, F )
%
% Write a mesh to disk in Wavefront OBJ format.
%
% INPUT:
%    filename - path to mesh file
%    V - dense 3x|V| matrix of vertex positions
%    F - dense |F|x3 matrix of faces as 1-based indices into vertex list
```

2.2 Curvature Operator

Write a MATLAB routine that builds a sparse $|V| \times |V|$ matrix that represents the curvature operator

$$K(x_i) = \frac{1}{4A_i} \sum_{j \in N(i)} \left(\cot \alpha_j + \cot \beta_j\right)(x_i - x_j)$$

where x_i is a vertex, $N(i)$ are the indices of its neighboring vertices, and α_j and β_j are the angles opposite the edge between x_i and x_j. The area A_i is typically the area of the Voronoi region around x_i, but for this assignment you can approximate A_i as one-third the total area of the triangles containing x_i. (Note that this definition does not take boundaries into account. Details about handling boundaries can be found in the paper on implicit fairing, though you are not required to implement these features for this assignment.)

If you think of the vertex locations as being stored in a $3 \times |V|$ matrix X, then K will be applied to (the transpose of) each row of X independently to compute the corresponding coordinates of the mean curvature normals. Hence each term of the sum will contribute the value $\left(\cot \alpha_j + \cot \beta_j\right)$ to entry K_{ii}, $-\left(\cot \alpha_j + \cot \beta_j\right)$ to entry K_{ij}, etc.

Note that when filling in entries of K it will be *much easier* to iterate over each face and compute its contribution to each of its vertices than, say, iterating over edges. Also note that you'll want to use a *sparse* matrix (e.g., `K = sparse( n, n )`) since most entries of K equal zero and you'll likely run out of memory (and time) otherwise. Finally, you should be careful not to divide by triangle areas close to zero since this will cause poor numerical behavior. (Terms involving the reciprocal of a small area can simply be omitted from curvature calculations.)

The routine you write should follow the template below.

```
  function K = curvature( V, F )
% function K = curvature( V, F )
%
% Smooths vertex positions using a mean curvature flow with explicit time stepping.
```

```
%
% INPUT:
%     V - dense 3x|V| matrix of vertex positions
%     F - dense |F|x3 matrix of faces as 1-based indices into vertex list
%
% OUTPUT:
%     K - sparse |V|x|V| matrix representing curvature operator
%
```

2.3 Curvature Flow

Write a MATLAB routine that applies the curvature operator K to a mesh using either explicit or implicit time stepping. In this context, explicit time stepping means that we take the current vertex positions, compute the curvature vectors, and add these vectors times a small *time step* to the original positions to get the smoothed positions. In matrix notation, this is equivalent to

$$X_i^{n+1} = (I - \Delta t K)X_i^n$$

where I is an identity matrix of the same dimensions as K, X_i^n is the ith component ($x = 1$, $y = 2$, $z = 3$) of the position vector at timestep n and $\Delta t \in \mathbb{R}$ is the size of the time step. Repeating this process for several time steps will make the surface successively smoother. However, this explicit scheme is susceptible to numerical *blow up* (i.e., the error is unbounded with respect to time) if we pick a Δt that is too large. Instead, we can use an *implicit* scheme:

$$(I + \Delta t K)X_i^{n+1} = X_i^n$$

where we now have to *solve* for X^{n+1}. This scheme allows us to take an arbitrarily large time step without fear of numerical blow up.

Both schemes should be trivial to implement in MATLAB once the curvature operator K is defined. In particular, you may simply use the backslash (\) operator to solve the implicit system. For example, to solve a general linear system $Bx = d$ for the vector x, you would write

```
x = B \ d
```

It is also useful to note that while the matrix arising from the implicit system is asymmetric, we can multiply by a matrix A with Voronoi areas on the diagonal to make it symmetric:

$$(A + \Delta t A K)X_i^{n+1} = AX_i^n$$

(in practice this means we omit the reciprocal area term from our curvature normal computation and store these areas in the matrix A instead). The advantage of applying this transformation is that it allows us to use a number of highly efficient linear solvers (such as the conjugate gradient method) that only work only on symmetric systems.

The routine you write should follow the template below.

```
  function V = smooth( V0, K, h, explicit )
% function V = smooth( V0, K, h, explicit )
%
% Smooths vertex positions using a mean curvature flow.
%
% INPUT:
%          V0 - dense 3x|V| matrix of original vertex positions
%           K - sparse |V|x|V| matrix of cotangent weights
%           h - timestep
%    explicit - 1 to use explicit time stepping; 0 otherwise.
%
% OUTPUT:
%     V - dense 3x|V| matrix of new vertex positions
%
```

2.4 Putting it All Together

Once you have all the above subroutines written, you should be able to run the following script to load in a mesh, apply smoothing, and save the smoothed mesh to disk. It is probably a good idea to test out individual routines (e.g., reading and writing meshes) by commenting out lines you're not currently interested in. In MATLAB, lines can be commented out by putting a percent (%) at the beginning of the line. We will provide you with several meshes on which you can run your code; you should expect to get results similar to those shown below. We will also provide you with a test mesh containing only a few vertices – it can be *extremely* helpful to check the values produced by your code against those computed by hand. You will need a way to view the meshes generated by your program – one free viewer that is fairly easy to use is Wings 3d. Before handing in your code you should verify that you get good results when taking a single, arbitrarily large step using the implicit scheme, and that the explicit scheme blows up with this same time step. (You should also verify that the explicit scheme works for a sequence of smaller steps.) Once you've finished, code should be emailed to the TA.

```
stderr = 2;
input_filename = '../meshes/noisy_torus.obj';
output_filename = '../meshes/out.obj';
timestep = 40.0;
iterations = 1;
explicit = 0;

fprintf( stderr, 'Reading...\n' );
[V,F] = read_mesh( input_filename );

fprintf( stderr, 'Smoothing...\n' );
for i = 1:iterations
   fprintf( stderr, '   iter: %d/%d\n', i, iterations );
   K = curvature( V, F );
   V = smooth( V, K, timestep, explicit );
end

fprintf( stderr, 'Writing...\n' );
write_mesh( output_filename, V, F );

disp( 'Done.' );
```