Mesh Differential Geometry

David Eberly
Geometric Tools, LLC
http://www.geometrictools.com/

Created: May 8, 2005
Last Modified: March 2, 2008

Contents

This document describes how to estimate various differential geometric quantites at the vertices of a triangle mesh. The mesh must be 2-manifold; that is, each edge is shared by one or two triangles and the mesh is oriented so that a consistent set of normals may be generated for the triangles. The orientation allows you to define an "outside" view of the mesh and an "inside" view of the mesh. For a mesh with the topology of a sphere, the labels outside and inside are clearly intuitive. For a mesh with the topology of a rectangle, you just have to associate the labels in some manner.

The triangle mesh is assumed to be an approximation to a C^2 surface and the mesh vertices are assumed to be samples from that surface. Of course the surface is considered to be unknown, otherwise you could use the standard methods of differential geometry applied directly to the surface to obtain what you need to know.

1 Surface Normal Estimates

There are a few methods for estimating surface normals at the vertices. In the computer graphics jargon, the estimates are called *vertex normals*. One method is to sum the unit-length normals of the triangles sharing a vertex, and then normalizing the sum to produce the vertex normal. The mesh is assumed to be stored as a vertex array and a triangle index array. The indices are into the vertex array. Each triple of indices corresponds to a triangle whose vertices are counterclockwise ordered when viewed from outside the mesh.

```
struct Mesh
{
    int numVertices;   // number of mesh vertices
    Point vertex[];    // array of numVertices points
    int numTriangles;  // number of mesh triangles
    int index[];       // array of 3*numTriangles indices into vertex[]
};

// some mesh of interest
Mesh mesh;

// information about the k-th triangle
int i0 = mesh.index[3*k];
int i1 = mesh.index[3*k+1];
int i2 = mesh.index[3*k+2];

// vertices
Point P0 = mesh.vertex[i0];
Point P1 = mesh.vertex[i1];
Point P2 = mesh.vertex[i2];

// edges
Vector E0 = P1 - P0;
Vector E1 = P2 - P1;
Vector E2 = P0 - P2;
```

```
// outer-pointing unit-length normal
Vector N = UnitCross(E0,E1);
```

The operation `UnitCross` for vectors $\mathbf{E}_0$ and $\mathbf{E}_1$ is $\mathbf{E}_0 \times \mathbf{E}_1/|\mathbf{E}_0 \times \mathbf{E}_1|$, if the input vectors are not parallel, or $\mathbf{0}$ if the input vectors are parallel.

The following pseudocode efficiently computes vertex normals.

```
void ComputeVertexNormals (Mesh mesh, Vector vnormal[])
{
    int i;
    for (i = 0; i < mesh.numVertices; i++)
        vnormal[i] = Vector(0,0,0);

    for (i = 0; i < mesh.numTriangles; i++)
    {
        // triangle vertices
        int i0 = mesh.index[3*i];
        int i1 = mesh.index[3*i+1];
        int i2 = mesh.index[3*i+2];
        Point P0 = mesh.vertex[i0];
        Point P1 = mesh.vertex[i1];
        Point P2 = mesh.vertex[i2];

        // triangle edges
        Vector E0 = P1 - P0, E1 = P2 - P1;

        // triangle normal
        Vector N = UnitCross(E0,E1);

        vnormal[i0] = vnormal[i0] + N;
        vnormal[i1] = vnormal[i1] + N;
        vnormal[i2] = vnormal[i2] + N;
    }

    for (i = 0; i < mesh.numVertices; i++)
        vnormal[i] = Normalize(vnormal[i]);
}
```

The input array `vnormal` is assumed to have `mesh.numVertices` elements. The function `Normalize` applied to a vector $\mathbf{V}$ returns $\mathbf{V}/|\mathbf{V}|$ if $\mathbf{V}$ is not the zero vector, or $\mathbf{0}$ if $\mathbf{V}$ is the zero vector.

2 First-Order Derivative Estimates

If the triangle mesh is assumed to approximate a height field, which is the graph of $z = f(x, y)$, then estimates to the first-order partial derivatives $\partial f/\partial x$ and $\partial f/\partial y$ are simple to obtain from the vertex normals. The

height field is implicitly represented by $F(x,y,z) = z - f(x,y) = 0$. The gradient vector is normal to the surface,

$$\nabla F(x,y,z) = \left(\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z}\right) = \left(-\frac{\partial f}{\partial x}, -\frac{\partial f}{\partial y}, 1\right)$$

If $\mathbf{N} = (n_0, n_1, n_2)$ is a unit-length estimate of the surface normal at (x,y,z), then dividing through by the n_2 components produces an estimate of the gradient as shown in the previous displayed equation

$$\left(-\frac{\partial f}{\partial x}, -\frac{\partial f}{\partial y}, 1\right) \doteq \left(\frac{n_0}{n_2}, \frac{n_1}{n_2}, 1\right)$$

This construction does assume $n_2 \neq 0$. If you were to have a vertex normal with a zero last component, you should question your assumption about the triangle mesh representing a height field since height fields do not have zero last components. The estimates of the first-order partial derivatives at a vertex are

$$\frac{\partial f}{\partial x} \doteq -\frac{n_0}{n_2}, \quad \frac{\partial f}{\partial y} \doteq -\frac{n_1}{n_2}$$

If the triangle mesh does not represent a height field, you most likely want to think of the true surface parametrically as $(x(s,t), y(s,t), z(s,t))$ for some choice of parameters (s,t). The first-order partial derivatives to estimate are

$$\frac{\partial x}{\partial s}, \quad \frac{\partial x}{\partial t}, \quad \frac{\partial y}{\partial s}, \quad \frac{\partial y}{\partial t}, \quad \frac{\partial z}{\partial s}, \quad \frac{\partial z}{\partial t}$$

Given a pair of adjacent vertices (connected by an edge of the mesh), we can clearly measure a change in the x-, y-, and z-components of the points. We also need measurements for the change in the u- and v-values of those vertices, which requires that you know such values. In practice, this is generally not the case and you have to assign (s,t) values to the vertices (x,y,z). In computer graphics, this is the problem of *automatic texture coordinate generation*, which is a difficult problem to solve. I consider a conformal mapping approach as one of the best methods to do this. See the paper below for such an approach.

> S. Haker, S. Angenant, A. Tannenbaum, R. Kikinis, G. Sapiro, M. Halle,
> *Conformal Surface Parameterization for Texture Mapping*,
> IEEE Transactions on Visualization and Computer Graphics, vol. 6, no. 2,
> pp. 181-189, April-June 2000

Conformal maps are studied in the field of complex analysis. The practical application to meshes involves finite element methods, solving large sparse matrix systems using conjugate gradient methods, and inverse stereographic projection.

Assuming you have the (s,t) values at the vertices (x,y,z), estimates of the first-order partial derivatives are computed as shown next. The idea is to estimate directional derivatives from the vertex of interest to each of its adjacent vertices. Let $\mathbf{P}_0 = (x_0, y_0, z_0)$ be a mesh vertex with adjacent vertices $\mathbf{P}_i = (x_i, y_i, z_i)$ for $1 \leq i \leq n$. Let the corresponding parameters be (s_i, t_i) for $0 \leq i \leq n$. Finite difference estimates are used, keeping in mind that the denominators must be positive. Also, a weighted average is computed over all the adjacent vertices,

$$\frac{\partial x(s_0,t_0)}{\partial s} \doteq \sum_{i=1}^{n} w_i \left\{ \begin{array}{lll} \frac{x_i - x_0}{s_i - s_0} & , \quad s_i > s_0 & \text{(forward difference)} \\ \frac{x_0 - x_i}{s_0 - s_i} & , \quad s_i < s_0 & \text{(backward difference)} \end{array} \right\}$$

The weights satisfy $w_i > 0$ for all i and $\sum_{i=1}^{n} w_i = 1$. The simplest choice is $w_i = 1/n$; however, you might want to place more emphasis on vertices that are close to $\mathbf{P}_0$ and less emphasis on vertices that are farther away. One such possibility is to compute $L_i = |\mathbf{P}_i - \mathbf{P}_0|$ and choose

$$w_i = \frac{L_i^{-1}}{\sum_{j=1}^{n} L_j^{-1}}$$

Similar estimates are made for all the other partial derivatives. In the event that $s_i = s_0$ for some i or $t_i = t_0$ for some i, you need to decide how to avoid the division by zero.

3 Curvature Estimates

Intuitively, curvature is a measurement of how much the unit-length surface normals varies as you move along a specific path. The motivation comes from studying planar curves $(x(t), y(t))$. The arc length parameter is s and is related to t by

$$\frac{ds}{dt} = |(x'(t), y'(t))|$$

In terms of arc length s, a unit-length tangent vector is

$$\mathbf{T}(s) = (\cos(\theta(t)), \sin(\theta(t)))$$

and a unit-length normal vector is

$$\mathbf{N}(s) = (-\sin(\theta(s)), \cos(\theta(s)))$$

These derivatives are related by the Frenet-Serret equations

$$\frac{d\mathbf{T}}{ds} = \kappa(s)\frac{d\mathbf{N}}{ds}, \quad \frac{d\mathbf{N}}{ds} = -\kappa(s)\frac{d\mathbf{T}}{ds}$$

where $\kappa(s) = d\theta/ds$ is the curvature. In this formulation, the curvature measures how fast the normal angle changes with respect to arc length. If the angle changes rapidly, the curvature is large and the curve itself bends rapidly at the point in question. Along a linear curve, the angle never changes, so the curvature is zero.

At a vertex on a surface, there are infinitely many directions in which the curvature can be measured. The directions are chosen to be in the tangent plane to the surface at the vertex. The minimum and maximum curvatures are called the *principal curvatures*. The corresponding unit-length tangent vectors are called the *principal directions*.

For an implicit surface $F(x, y, z) = 0$, a unit-length normal vector field is

$$\mathbf{N}(x, y, z) = \frac{\nabla F(x, y, z)}{|\nabla F(x, y, z)|}$$

An algorithm presented earlier allowed us to estimate $\mathbf{N}(x, y, z)$ at each vertex of the triangle mesh approximating the surface. The derivative matrix of the normal vector field is

$$D\mathbf{N}(x, y, z) = \left(I - \mathbf{N}(x, y, z)\mathbf{N}(x, y, z)^{\mathrm{T}}\right) \frac{D^2 F(x, y, z)}{|\nabla F(x, y, z)|}$$

Intuitively, $\mathbf{N}$ measures the first-order rate of change of F normalized by the length of the gradient. A similar interpretation applies to $D\mathbf{N}$. It measures the second-order rate of change of F normalized by the length of the gradient, but within the tangent space at the point. The matrix $I - \mathbf{N}\mathbf{N}^{\mathrm{T}}$ is the projection matrix onto the tangent plane.

Given a unit-length normal vector $\mathbf{N}$, we may choose unit-length vectors $\mathbf{U}$ and $\mathbf{V}$ such that $\{\mathbf{U}, \mathbf{V}, \mathbf{N}\}$ is an orthonormal set. Define the matrix $J = [\mathbf{U} \mid \mathbf{V}]$, a 3×2 matrix whose columns are the specified vectors. Define J^{T} to be the transpose of J, a 2×3 matrix. The *shape matrix* is

$$S = J^{\mathrm{T}}(D\mathbf{N})J$$

and is a 2×2 matrix. The matrix describes locally the shape of the surface in that its eigenvalues are the principal curvatures. If κ is a principal curvature, let $\mathbf{E}$ be the 2×1 eigenvector of S associated with it. The 3×1 principal direction vector is $J\mathbf{E}$.

In the case of a triangle mesh, we can estimate $\mathbf{N}$, and then choose vectors $\mathbf{U}$ and $\mathbf{V}$. We do not know $D\mathbf{N}$, but instead estimate its values from what we do know. The intuition is as follows. All the vertices have estimated surface normals, these normals representing a first-order rate of change of F locally. The rate of change of the vertex normals between a pair of adjacent vertices is a measure of a second-order rate of change from which we can estimate $D\mathbf{N}$.

The rate of change of $\mathbf{N}$ in a specified unit-length tangent direction $\mathbf{W}$ is the directional derivative

$$\mathbf{R} = (D\mathbf{N})\mathbf{W},$$

a vector-valued quantity. If we can estimate $\mathbf{R}$ at vertices, then we can estimate $D\mathbf{N}$. Let $\mathbf{P}_i$ and $\mathbf{P}_j$ be two adjacent vertices with corresponding normals $\mathbf{N}_i$ and $\mathbf{N}_j$. Let us work in the tangent space of $\mathbf{P}_i$ to make measurements about how the normal vector changes. The tangent space is the plane containing $\mathbf{P}_i$ and that is perpendicular to $\mathbf{N}_i$. The not necessarily unit-length direction to the adjacent vertex is $\mathbf{D}_{ji} = \mathbf{P}_j - \mathbf{P}_i$, but is not necessarily in the tangent plane. We can project it onto the plane and normalize it by

$$\begin{aligned} \mathbf{P}_{ji} &= \mathbf{D}_{ji} - (\mathbf{N}_i \cdot \mathbf{D}_{ji})\,\mathbf{N}_i \\ \mathbf{W}_{ji} &= \frac{\mathbf{P}_{ji}}{|\mathbf{P}_{ji}|} \end{aligned}$$

The difference of normals is

$$\mathbf{R}_{ji} = \mathbf{N}_j - \mathbf{N}_i$$

The *approximation* to the rate of change of vertex normals is

$$\mathbf{R}_{ji} = (D\mathbf{N})\mathbf{W}_{ji}$$

where $D\mathbf{N}$ is the derivative matrix of the normal vector field at the position $\mathbf{V}_i$. The problem is still that we do not know what is $D\mathbf{N}$.

Think of $D\mathbf{N}$ as a collection of 9 unknown values. Any approximate rate of change may be used to help establish the values of the unknowns. For a closed manifold triangle mesh, each vertex has at least 3 edges emanating from it. Therefore the corresponding approximate rate of change equations give us at least 9 pieces of information about the unknowns. We can formulate construction of the unknowns as a least-squares problem. Suppose that the vertex $\mathbf{P}$ has n adjacent vertices. Each adjacent vertex leads to a unit-length direction vector $\mathbf{W}_j$ in the tangent plane of $\mathbf{P}$ and each adjacent vertex normal leads to a difference normal $\mathbf{R}_j$, $1 \le j \le n$. The approximate rate of change equations can be written jointly as

$$(D\mathbf{N})W = D\mathbf{N}[\mathbf{W}_1 \mid \cdots \mid \mathbf{W}_n] = [\mathbf{R}_1 \mid \cdots \mid \mathbf{R}_n] = R$$

where W is a $3 \times n$ matrix whose columns are the specified unit-length direction vectors and R is a $3 \times n$ matrix whose columns are the specified normal differences. Multiplying by W^{T} leads to

$$(D\mathbf{N})(WW^{\mathrm{T}}) = RW^{\mathrm{T}}$$

The matrix WW^{T} is a 3×3 matrix. If W were to have rank 3, then WW^{T} would be invertible in which case we could solve for $D\mathbf{N}$ directly. The problem is that the columns of W are all tangent vectors at $\mathbf{P}$ and lie in the same plane; consequently W has rank 2 and we cannot invert WW^{T}.

To remedy the problem, we can additionally stipulate that the normal vectors do not change as you move in the normal direction. This is an artificial constraint because you cannot leave the mesh when making surface measurements, but it makes intuitive sense. The mathematical condition for this is

$$(D\mathbf{N})\mathbf{N} = \mathbf{0}$$

Note that this is not the condition implied by having unit-length normals. That is, since $\mathbf{N} \cdot \mathbf{N} = 1$ for all $\mathbf{N}$, we can take the derivative and obtain

$$\mathbf{0} = D(\mathbf{N} \cdot \mathbf{N}) = \mathbf{N}^{\mathrm{T}} D\mathbf{N} + (D\mathbf{N})^{\mathrm{T}}\mathbf{N} = 2\mathbf{N}^{\mathrm{T}} D\mathbf{N}$$

This equation is different than the previous one since $D\mathbf{N}$ is not a symmetric matrix. The artificial constraint can be included in the set of equations that determine $D\mathbf{N}$ as

$$(D\mathbf{N})W = D\mathbf{N}[\mathbf{W}_1 \mid \cdots \mid \mathbf{W}_n \mid \mathbf{N}] = [\mathbf{R}_1 \mid \cdots \mid \mathbf{R}_n \mid \mathbf{0}] = R$$

where W and R are now $3 \times (n+1)$ matrices. The matrix W now has rank 3, so W is invertible. We can solve for the derivative matrix:

$$D\mathbf{N} = RW^{\mathrm{T}}(WW^{\mathrm{T}})^{-1}$$

To handle numerical round-off problems that might cause WW^{T} to be nearly singular, the source code produces the zero matrix for the inverse (rather than aborting) in which case the mean curvature estimate will be zero. The implementation is in `Wm3MeshCurvature.h` and `Wm3MeshCurvature.inl`.