Article 7

Implementation of an Algorithm for Approximating the Curvature Tensor on a Triangular Surface Mesh in the Vish Environment

Edwin Mathews[1], Werner Benger[1], Marcel Ritter[2]
[1]Center for Computation & Technology
at Louisiana State University (CCT/LSU), Baton Rouge, Louisiana, USA
[2] Unit of Hydraulic Engineering,
Department of Infrastructure, University of Innsbruck

Finding local surface properties of a generated mesh is an essential component in applications across several fields. Specifically, Gaussian curvature provides intrinsic geometric information of local shape characteristics on a surface. It finds use in mesh applications like 3-D scanned image noise smoothing, feature recognition, and data analysis. An algorithm was developed for the Vish environment to approximate the shape operator from the curvature tensor using only lists of triangle vertex positions and individual vertex positions. The algorithm is based on a curvature tensor approximation method developed by Gabriel Taubin and does not need information about the mesh edges to be provided explicitly in the calculation. From the curvature tensor, the principle directions and curvatures can be found and used to calculate the Gaussian curvature and mean curvature at each vertex. Using this information, an application is given where the curvature is used to analyze mixing on the surface of a fluid 'virtual bubble.'

7.1 Introduction

Development of a Vish-specific algorithm that takes a mesh surface as an input and returns a field of curvature values is an important foundation in the development of new visualization modules. In current practice, curvature calculation is essential to feature detection and noise filtering of sampled three dimensional geometry and to the detection and analysis of medical images. With the broad scope of polyhedral surfaces in computer graphics in industrial and biomedical engineering, robotics, and several other fields, it is a start for where Vish may find itself in years to come. Here, curvature finds a novel use in the analysis of Computational Fluid Dynamics (CFD) time surfaces [Bohara et al., 2010]. The time surface is a virtual bubble constructed from seed points in a CFD generated vector field [Benger et al., 2009]. It can be seen as a higher dimensional extension of timelines, where a continuous mesh surface constructed from the seed points evolves over time. Being able to visualize a changing surface in a large data model has advantages in observing fluid movement and mixing over timelines. Where timelines are effective in displaying the trajectory and rotation of a particle, time surfaces give a more intuitive visualization of the effects of the time dependent vector field and of the movement of multiple particles in relation to each other.

7.1.1 Application in Analysis

The time surface was intially developed to visualize a simulation of stirred tank data to help improve mixing efficiency of the stirring process [Bohara et al., 2010]. Without a means to quantify the mixing, conclusions about the effectiveness of the stirred tank had to be made via visual inspection of the time surface. In general, mixing is the combining of two or more substances into one mass with a thorough blending of the constituents. So how can we quantify this?

Surface shape information at each vertex is useful in representing the mixing on the time surface because the variation of a vertex position relative to its neighboring vertices can be quantified. If the curvature of a vertex is initially zero then changes in magnitude abruptly, it signifies a quick movement of the particles around the vertex in relation to its neighbors. Unfamiliar particles will then come into contact as a result of replacing the particles that have just moved away. So while the curvature itself does not represent the mixing of the fluid locally, but the change of curvature does. When evaluating over a finite mesh, it is important to compute intrinsic information about the surface so that nominal values of the information will not be dependent on the mesh refinement. The Gaussian curvature is an intrinsic invariant of the surface as opposed to the extrinsic invariant mean curvature; it does not depend on its embedding, and a mesh approximation will only become more accurate with a higher refinement.

7.2 Mathematic Principles

In this application, the method looks to find the shape operator at some point p defined on a surface S. The shape operator, or Weingarten map, is a type of extrinsic curvature that is a linear operator on the tangent space at each point p on the surface S. At any point, it is equal to the Jacobian of $\mathbf{N}$, the function that yields the unit vectors normal to the surface. To calculate this on a grid vertex, the contribution from each edge attached to the vertex must be averaged. Along this edge where the normal vectors at two adjacent points are given, the difference of the normal vectors $dN^k = N_0^k - N_1^k$ with respect to each coordinate direction $dx^i = x_0^i - x_1^i$ can be found. In three dimensions,

$$J_i^k = \begin{pmatrix} \frac{dN}{dx^1} \\ \frac{dN}{dx^2} \\ \frac{dN}{dx^3} \end{pmatrix} = \begin{pmatrix} \frac{dN^1}{dx^1} & \frac{dN^2}{dx^1} & \frac{dN^3}{dx^1} \\ \frac{dN^1}{dx^2} & \frac{dN^2}{dx^2} & \frac{dN^3}{dx^2} \\ \frac{dN^1}{dx^3} & \frac{dN^2}{dx^3} & \frac{dN^3}{dx^3} \end{pmatrix} = \frac{dN^k}{dx^i} \tag{7.1}$$

where J_i^k is the Jacobian along this edge vector given by $x_0 - x_1$. The projection of the surface Jacobian on to the tangent plane can be computed if two vectors tangent to the surface, $\vec{u}$ and $\vec{v}$, are given where $\vec{u}$, $\vec{v}$, and the normal vector at p, form an orthonormal basis on the surface. This shape operator in the surface is equal to

$$S = \begin{pmatrix} J(u,u) & J(u,v) \\ J(v,u) & J(v,v) \end{pmatrix} = \begin{pmatrix} \kappa_p^{11} & \kappa_p^{12} \\ \kappa_p^{21} & \kappa_p^{22} \end{pmatrix} \tag{7.2}$$

where the operation $J(\,\cdot\,,\cdot\,)$ is of the form $J(u,u) = u^t J u$ and $J(u,v) = u^t J v$. The shape operator can also be defined using the Metric Tensor, $I = I_{xx}dx^2 + 2I_{xy}dxdy + I_{yy}dy^2$, and the Extrinsic Curvature, $II = II_{xx}dx^2 + 2II_{xy}dxdy + II_{yy}dy^2$, coordinate description of the tangent space at the surface S at p. Using the coefficients of the first and second fundamental forms

$$S = \frac{1}{det\ I} \begin{pmatrix} II_{xx}I_{yy} - II_{xy}Ixy & II_{xy}I_{yy} - II_{yy}I_{xy} \\ II_{xy}I_{xx} - II_{xx}Ixy & II_{yy}I_{xx} - II_{xy}I_{xy} \end{pmatrix}$$

where $det\ I = I_{xx}I_{yy} - I_{xy}^2$, the area of the surface element. The result for both formulations is a 2×2 symmetric matrix where the eigenvalues are just the principle curvatures at some p on S. Accordingly, the determinant is the Gaussian curvature and half the trace of the shape operator is the mean curvature. To find the directional curvature at some point with a defined shape operator in direction $\vec{T}$ defined in surface coordinates, the following operation is performed

$$\kappa_p = S(T,T) = \begin{pmatrix} t_1 & t_2 \end{pmatrix} \begin{pmatrix} \kappa_p^{11} & \kappa_p^{12} \\ \kappa_p^{21} & \kappa_p^{22} \end{pmatrix} \begin{pmatrix} t_1 \\ t_2 \end{pmatrix} \tag{7.3}$$

where $\vec{T} = t_1\hat{T}_1 + t_2\hat{T}_2$ is a tangent vector on S at p. $\hat{T}_1$ and $\hat{T}_2$ are the principle directions on S at p when the shape operator is diagonal and forms an orthonormal basis of the tangent space. If the normal vector N at p is added to this orthonormal basis, eqn. (7.3) is extended to non-tangential directions

$$\kappa_p = \begin{pmatrix} n & t_1 & t_2 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 \\ 0 & \kappa_p^{11} & \kappa_p^{12} \\ 0 & \kappa_p^{21} & \kappa_p^{22} \end{pmatrix} \begin{pmatrix} n \\ t_1 \\ t_2 \end{pmatrix} \tag{7.4}$$

and $\vec{T}$ is expanded to $\vec{T} = n\hat{N} + t_1\hat{T}_1 + t_2\hat{T}_2$. The vector $\vec{T}$ can be written as a linear combination of another orthonormal system. The directional curvature can still be evaluated using this different coordinate system with another 3×3 symmetric matrix, U_p, as long the three eigenvalues are still 0, κ_p^{11}, and κ_p^{22}. The shape operator can be recovered from the curvature tensor U_p by restricting the matrix to the tangent plane.

Using the method described by Taubin [Taubin, 1995], the curvature tensor is approximated by defining a matrix M by an integral formula that has the same eigenvectors as U_p and has their eigenvalues related by a fixed homogeneous linear transformation. For $-\pi \leq \theta \leq \pi$ on the tangential plane, $\vec{T_\theta}$ at p is the unit length tangent vector $\vec{T_\theta} = cos(\theta)\hat{T}_1 + sin(\theta)\hat{T}_2$. $\hat{T}_1$ and $\hat{T}_2$ represent the same orthonormal principle directions in surface coordinates as previously mentioned. Using this in the expression for directional curvature:

$$\kappa_p = \begin{pmatrix} \cos\theta & \sin\theta \end{pmatrix} \begin{pmatrix} \kappa_p^{11} & \kappa_p^{12} \\ \kappa_p^{21} & \kappa_p^{22} \end{pmatrix} \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix}$$

Because $\hat{T}_1$ and $\hat{T}_2$ are the principle directions, $\kappa_p^{12} = \kappa_p^{21} = 0$. Evaluating the equation results in $\kappa_p(T_\theta) = \kappa_p^{11}cos^2(\theta) + \kappa_p^{22}sin^2(\theta)$. Integrating over the entire surface element about p, the 3×3 approximation matrix M is defined as

$$M = \frac{1}{2\pi}\int_{-\pi}^{\pi} \kappa_p(T_\theta)(T_\theta \otimes T_\theta)d\theta \tag{7.5}$$

In M, the normal vector N is an eigenvector associated with the zero eigenvalue since $T_\theta \otimes T_\theta$ is a rank 1 matrix at every θ and $\vec{T_\theta}$ is tangent to the surface at p. If M is then factorized and and integrated, it can be shown that the eigenvalues of M, m^{11} and m^{22}, are related to the principle curvatures by

$$\kappa_p^{11} = 3m_p^{11} - m_p^{22} \tag{7.6}$$

$$\kappa_p^{22} = 3m_p^{22} - m_p^{22} \tag{7.7}$$

To approximate the curvature κ_p along some tangential vector $\vec{T}$, we use a curve $q(s)$, a normal section to S at p parameterized by arc length. By differentiating and solving at $s = 0$, $q(0) = p$, $q'(0) = T$, and $q''(0) = \kappa_p(T)N$, where N is the unit length normal vector. When $q(s)$ is expanded in Taylor series

$$q(s) = q(0) + q'(0)s + \frac{1}{2}q''(0)s^2 + O(s^3)$$

$$= p + Ts + \frac{1}{2}\kappa_p(T)Ns^2 + O(s^3)$$

Removing higher terms and solving for $\kappa_p(T)$,

$$\kappa_p(T) = \lim_{s\to 0} \frac{2N^t(q(s) - p)}{||q(s) - p||^2} \approx \frac{2N^t(p_j - p_i)}{||p_j - p_i||^2} \tag{7.8}$$

where p_j is another point on the surface close to the point p_i, where the approximation is being done. This allows for the evaluation of the directional curvature directly on a mesh surface between two neighboring points p_j and p_i. Now, all of the components needed to compute the shape operator have been defined. The integral formula for M when solving at p_i is discretized to

$$\tilde{M}_i = \sum_{j\in i} w_{ij}\kappa_{ij}(T_{ij} \otimes T_{ij}) \tag{7.9}$$

where j are the vertexes surrounding vertex i. The weighted value w_{ij} is value based on the area of the triangles bordering the tangent vector. Once $\tilde{M}_i$ is calculated, a Householder matrix Q corresponding to the plane orthogonal to the unit normal vector is used to decompose the 3×3 matrix $\tilde{M}$. The Householder reflection is a transformation that describes a reflection by a plane, in this case the tangential plane at p on S. By construction, the first column of the Householder matrix is equal to the unit normal vector and the other two columns are an orthonormal basis of the tangent space. Because the unit normal vector at p_i is an eigenvector of $\tilde{M}_i$ corresponding to the 0 eigenvalue,

$$Q^t\tilde{M}_iQ = \begin{pmatrix} 0 & 0 & 0 \\ 0 & m_p^{11} & m_p^{12} \\ 0 & m_p^{21} & m_p^{22} \end{pmatrix} \tag{7.10}$$

The values of the 2×2 minor can then be used to find the values of the principle curvature using the relations above for κ_p^{11} and κ_p^{22}. From there, as described above, the Gaussian curvature or mean curvature can be evaluated at that vertex.

7.3 Our Approach

To use the algorithm, an input list of vertices and list of triangles with the three vertices that make up each triangle of the current grid are used. Instead of requiring a list of edges on the grid surface, they will be computed for each triangle. The first step is to compute the weighted unit normal vector, N_{v_i}, of each vertex.

$$N_{v_i} = \frac{\sum_{f_k \in F^i} N_{f_k} A_k}{||\sum_{f_k \in F^i} N_{f_k} A_k||} \tag{7.11}$$

To find N_{v_i}, the normal vector, N_{f_k}, of each triangle face f_k surrounding vertex i is multiplied by its face area A_k. The sum of all faces is then normalized. This is done within a loop for all triangles on the surface. First, coordinates for each of the three vertices in the triangle are needed.

```
for (i = 0; i <Number of Triangles; i++)
  {
      point A, B, C;
      A = ... ; B = ... ; C = ... ;
```

Then, two of the triangle edge vectors are computed for finding the area and normal vector.

```
      tvector BA = B - A;
      tvector CA = C - A;
```

The area is determined by half of the cross product norm of the same two edge vectors. The triangular area is placed into an array.

```
      TriangleArea[ i ] = .5*norm(BA ^ CA);
```

The weighted triangle surface normals are then weighted by their area and placed in a 3-component vector array.

```
      WeightedTriangleNormal[ i ] = (BA^CA);
```

A nested loop is created for each of the vertices in the triangle. The area of the triangle is then summed to an array on the vertices. This `VertexArea` is a scalar equal to the area of all triangular faces surrounding a single vertex. It finds use later in the algorithm when calculating a weighted value and could also be visualized on the time surface to show where the surface is 'stretching'.

```
for(k = 0; k<3; k++)
  { VertexArea[T[k]] += TriangleArea[ i ];
```

Here, `T[k]` is the index of the three vertices in the triangle. For the call `Vertices Area[T[k]] += ...`, the area is being summed to the vertices through the current triangle in the outer loop. The weighted triangle normal vector is then summed to another three-component vector array on each of the vertices in the same fashion.

```
      WeightedVertexNormal[T[k]] += WeightedTriangleNormal[i];
  }
}
```

Both loops are then closed. After this section has completed, each vertex will have an array with the sum of all the triangular areas surrounding it and an array with the sum of all the triangular surface unit normal vectors. A new loop for all vertices on the surface is created to normalize the summed weighted vertex normals.

```
for(v=0; v < Number of Vertices; v++)
  {  WeightedVertexNormal[v] = WeightedVertexNormal[v].unit(); }
```

Next, another loop for all triangles is used to compute the curvature tensor of each vertex. To start, edge vectors must be computed for the first edge.

```
for(i = 0; i < Number of Triangles; i++)
  {    tvector AB, BA;
    AB = ... ; BA = ... ;
```

For the computation of M at the two vertices attached at the ends of this edge, a tensor weight w_{ij} is used. This weight helps to balance the contribution of edges representing a larger portion of the area surrounding the vertex. For all edge contributions to a vertex, $\sum w_{ij} = 1$. The weight for this triangle's contribution to the shape tensor is equal to the area of the triangle divided by twice the area of all the triangles surrounding the vertex to which it is contributing.

```
     double Wij0 = TriangleArea[ i ]/(2*VerticesArea[T[0]]);
```

Here, w_{ij} is computed for triangle vertex A, or 0, denoted by the term `Wij0`. The projection of the edge vector $\vec{AB}$ on the tangential plane is calculated using

$$T_{ij} = \frac{(I - N_{v_i} \otimes N_{v_i})(\vec{AB})}{||(I - N_{v_i} \otimes N_{v_i})(\vec{AB})||} \tag{7.12}$$

where I is the 3×3 identity matrix and N_{v_i} is unit normal vector at the vertex i. This operation can be performed by the single function

```
tvector TijAB = Tijoperator( WeightedVertexNormal[T[0]], AB );
```

Then, for the same edge the directional curvature needs to be found using eqn. (7.8). A function is `Kijoperator()` used for that purpose.

```
double KijAB = Kijoperator( WeightedVertexNormal[T[0]], AB );
```

Now, all of the components are available to compute the contribution of edge $\vec{AB}$ to the approximation of the curvature tensor at vertex A. The tensor product of T_{ij} is computed, multiplied by the two scalar values κ_{ij} and w_{ij} and, finally, added to a 3×3 matrix array indexed for the particular vertex.

```
Matrix33 TijABTensorProduct = ... ;
CurvTensor[T[0]] += Wij0 * KijAB * TijABTensorProduct;
```

where `CurvTensor` is a 3×3 matrix array. This operational procedure is repeated for the other edge vectors in the triangle. Each vertex will have two components of the curvature tensor added to it from each triangle of which it belongs, so for each triangle in the loop above, six curvature tensor components will be computed. To continue with computing the shape operator from the curvature tensor, the loop for all triangles is ended and a new loop for all the vertices on the surface is started. To find the Householder matrix which is needed to restrict the tensor to the tangential plane, the value of the sign in the calculation of W_{vi} must be found:

$$W_{vi} = \frac{E_1 \pm N_{v_i}}{||E_1 \pm N_{v_i}||} \tag{7.13}$$

where E_1 is the first coordinate vector and is equal to $(1, 0, 0)^t$. If the magnitude of $||E_1 - N_{v_i}|| > ||E_1 + N_{v_i}||$, the sign is negative; otherwise, it is positive.

```
}
for(v = 0; v < Number of Vertices; v++)
{ tvector E(1,0,0);
     tvector A = E + WeightedVertexNormal[ v ];
     tvector B = E - WeightedVertexNormal[ v ];
     double MagB = norm(B);
     double MagA = norm(A);
     tvector Wvi;
     if(MagB > MagA)
     { Wvi = B.unit(); }
     else
     { Wvi = A.unit(); }
```

The Householder matrix Q_{v_i} is then found by the formula $Q_{v_i} = I - 2W_{v_i}W_{v_i}^t$ And completing the restriction to the tangential plane, $Q_{v_i}^t \tilde{M}_{v_i} Q_{v_i}$ is computed.

```
Matrix33 QviT = ...;
Matrix33 Holder = QviT*CurvTensor[v]*Qvi;
```

where Holder is the final 3×3 Matrix whose first row and column are zeros. That leaves a 2×2 non-zero minor whose eigenvalues are m^{11} and m^{22} from eqn. (7.6) and (7.7). The two principle curvatures now finally allow the computation of the Gaussian or mean curvature of a triangular surface.

```
GaussCurve[ v ] = (3*m11 - m22) * (3*m22 - m11); }
```

7.4 Results

Using the curvature tensor module and pre-existing modules in Vish to render scalar and tensor data on a surface, we have been able to produce visualizations useful in the analysis of fluid mixing on the surface of time surfaces, such as illustrated in fig. (7.1). Here the Gaussian Curvature is shown on the triangular surface of a fluid blob of the stirred tank data set. The blue area identifies locations with a large positive Gaussian Curvature and red identifies a large negative value. As time elapses in the simulation, mixing is indicated by color change on the surface. As previously stated, this signifies the movement of particles within the surface and contact with new particles. Currently there are singularity points in the approximation related to a incorrect sign in the curvature. This tends to occur at points where the refinement of the mesh is very poor and less than six triangles are surrounding a vertex.

7.5 Future Work

The mesh refinement procedure for the time surface is presently being updated to improve the mesh quality, and may prove to diminish errors due to poor surface quality. Beyond that, values of the approximated curvature are being compared to surfaces with known curvature for validation and to estimate error. It is also of interest to directly compute the time derivative of the Gaussian curvature as a means of analysis instead of a visual interpretation. In future projects, the curvature module is looking to be used in the analysis of chemical dispersant mixing in Gulf of Mexico oil spill modeling. These disperants contain surfactants that dissipate oil slicks but are dependent on wave action and water movement for mixing. This modeling would utilize the time surface for visualization and could interpreted similarly to the stir tank data.

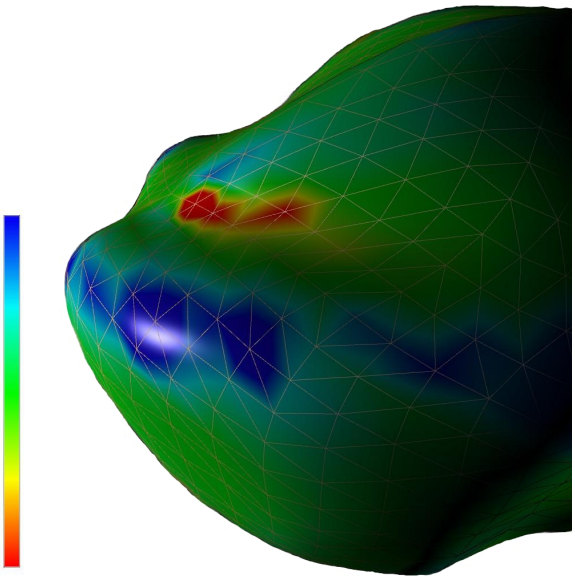

Figure 7.1: Gaussian Curvature on a triangulated time surface representing a fluid blob.

7.6 Summary

After presenting the mathematical foundation of surface curvature, an algorithm for its numerical computation was presented. The curvature tensor approximation was then implemented as a module in the Vish framework. In Vish, the module is currently being employed to compute the Gaussian curvature for the analysis of mixing on a CFD time surface. It provides a visually intuitive and quantifiable method of mixing evaluation in time surface simulations along with providing a characteristic value used in several other graphical applications.

Acknowledgements

Many thanks to the Group of Scientific Visualization at LSU and the Visualization Services Center at LSU for their assistance in the development of the algorithm. Thanks to the Louisiana Optical Network Infrastructure for providing the ability to contribute research in the field of scientific visualization and the Vish framework.

Bibliography

[Benger et al., 2009] Benger, W., Ritter, M., Acharya, S., Roy, S., & Jijao, F. (2009). Fiberbundle-based visualization of a stir tank fluid. In 17^{th} *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision* (pp. 117–124).

[Bohara et al., 2010] Bohara, B., Harhad, F., Benger, W., Brener, N., Iyengar, S., Ritter, M., Liu, K., Ullmer, B., Shetty, N., Natesan, V., Cruz-Neira, C., Acharya, S., & Roy, S. (2010). Evolving time surfaces in a virtual stirred tank. *Journal of WSCG*, 18(1-3), 121–128.

[Taubin, 1995] Taubin, G. (1995). Estimating the tensor of curvature of a surface from a polyhedral approximation.