Estimating Curvatures and Their Derivatives on Triangle Meshes

Szymon Rusinkiewicz
Princeton University

Abstract

The computation of curvature and other differential properties of surfaces is essential for many techniques in analysis and rendering. We present a finite-differences approach for estimating curvatures on irregular triangle meshes that may be thought of as an extension of a common method for estimating per-vertex normals. The technique is stable and robust, offers accuracy comparable to or better than existing methods, and generalizes naturally to computing derivatives of curvature and higher-order surface differentials.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling.

Keywords: curvature, differential properties, irregular meshes.

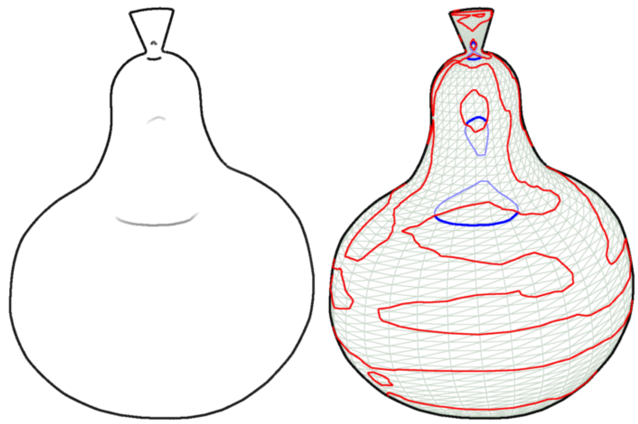


Figure 1: Left: *suggestive contours for line drawings [DeCarlo et al. 2003] are a recent example of a driving application for the estimation of curvatures and derivatives of curvature.* ***Right:*** *suggestive contours are drawn along the zeros of curvature in the view direction, shown here in blue, but only where the derivative of curvature in the view direction is positive (the curvature derivative zeros are shown here in red). This paper describes a general and stable algorithm for estimating curvature and derivative-of-curvature tensors on triangle meshes.*

1 Introduction

As the acquisition and use of sampled 3D geometry become more widespread, 3D models are increasingly becoming the focus of analysis and signal processing techniques previously applied to data types such as audio, images, and video. A key component of algorithms such as feature detection, filtering, and indexing, when applied to both geometry and other data types, is the discrete estimation of differential quantities. In the case of shape, surface differentials such as normals and curvatures arise not only in the context of these "signal analysis" applications, but also in pure graphics algorithms such as illumination and nonphotorealistic rendering (Figure 1). This paper describes a general, robust algorithm for estimating curvatures and higher-order surface differentials on surfaces approximated by triangle meshes.

A key difference between 3D meshes and data types such as images, video, and even volumetric data stored on voxel grids is that meshes are typically irregularly sampled. The distribution of vertices across the surface is not uniform, and connectivity (in particular, the valence of each vertex) is not regular except in special cases. In order to be generally useful, therefore, an algorithm for estimating differential quantities must be robust under different distributions of triangle sizes and shapes. Other properties desirable in an algorithm that operates on commonly-encountered meshes include:

- not placing any requirements on the topology of the surface. In particular, the assumption that a surface is hole-free is often violated.
- being free of degenerate cases, unless the mesh itself is degenerate. For example, we wish to avoid the instability of some methods for particular configurations of vertices, such as collinear points.
- not relying on smoothing or averaging over a large neighborhood to provide robustness. In many cases smoothing is necessary to eliminate noise, but algorithms should not require large neighborhoods for stability.

In the case of estimating per-vertex normals, we note that a commonly-used algorithm, namely taking a weighted average of the normals of faces touching a vertex, satisfies the above properties. It handles arbitrary triangulations, makes no assumptions about topology or the presence of holes, is typically free of degeneracies, and operates on local neighborhoods. This algorithm is also efficient, requiring only a single pass over faces and one over vertices (to rescale the resulting normals to unit length), and does not require any connectivity data structures beyond the usual vertex list and indexed face set.

Inspired by this algorithm, we present a method for computing curvatures and higher-order derivatives in an analogous fashion: we first compute the properties per-face, then estimate the value at each vertex as a weighted average over the immediately adjacent faces. The per-face computations are based directly on the definition of the relevant derivative, using a a finite-difference approximation. The curvature tensor, for example, is defined in terms of the directional derivative of the surface normal, and we calculate it from differences between estimated per-vertex normals.

2 Background and Previous Work

We begin with a brief overview of curvatures on a 3D surface (see, for example, [Cipolla and Giblin 2000] for further details). The *normal curvature* κ_n of a surface in some direction is the reciprocal of the radius of the circle that best approximates a normal slice of surface in that direction. The normal curvature varies with direction, but for a smooth surface it satisfies

$$\kappa_n = \begin{pmatrix} s & t \end{pmatrix} \begin{pmatrix} e & f \\ f & g \end{pmatrix} \begin{pmatrix} s \\ t \end{pmatrix} = \begin{pmatrix} s & t \end{pmatrix} \mathrm{II} \begin{pmatrix} s \\ t \end{pmatrix} \qquad (1)$$

for any unit-length vector (s,t) in the local tangent plane (expressed in terms of an orthonormal coordinate system centered at the point). The symmetric matrix II appearing here, known as the Weingarten matrix or the second fundamental tensor, can be diagonalized by a rotation of the local coordinate system to give

$$\kappa_n = \begin{pmatrix} s' & t' \end{pmatrix} \begin{pmatrix} \kappa_1 & 0 \\ 0 & \kappa_2 \end{pmatrix} \begin{pmatrix} s' \\ t' \end{pmatrix} = \kappa_1 s'^2 + \kappa_2 t'^2, \qquad (2)$$

where κ_1 and κ_2 are the *principal curvatures* and (s',t') is now expressed in terms of the *principal directions*, which are the directions in which the normal curvature reaches its minimum and maximum. The principal curvatures and principal directions have been widely used in computer graphics, appearing in applications such as remeshing [Alliez et al. 2003], smoothing [Desbrun et al. 1999], segmentation [Trucco and Fisher 1995], visualization [Interrante et al. 1995], and nonphotorealistic rendering [Hertzmann and Zorin 2000; DeCarlo et al. 2003].

We may classify existing methods for estimating principal curvatures and directions (as opposed to methods that estimate only the mean curvature $H = (\kappa_1 + \kappa_2)/2$ or Gaussian curvature $K = \kappa_1 \kappa_2$) into three general categories:

- **Patch fitting methods** fit an analytic surface (usually a polynomial) to points in a local region, then compute curvatures of the fit surface analytically. These methods clearly produce exact results if the vertices are on a surface of the class being fit, and Cazals and Pouget [2003] have shown that in the case of a general smooth surface the estimated curvatures converge to the true ones, at least in nondegenerate cases. The weakness of patch fitting methods that only consider vertex positions is that they become unstable near degenerate configurations, most notably if the points lie near a pair of intersecting lines (Figure 2). Goldfeather and Interrante [2004] have shown that the degenerate cases can be avoided, and accuracy improved in general, by including not only points but also estimated per-vertex normals in the fit.
- **Normal curvature-based methods** first estimate the normal curvature in the direction of each edge leaving a vertex, then use the κ_n estimates to find the second fundamental tensor. Most commonly, the formula

$$\kappa_{ij} = \frac{2n_i \cdot (p_i - p_j)}{|p_i - p_j|^2} \tag{3}$$

  is used to find the normal curvature at point p_i, in the direction of some neighboring point p_j. The principal curvatures may then be found from a function of the eigenvalues of $\sum_j \kappa_{ij}(p_i - p_j)(p_i - p_j)^{\mathrm{T}}$ [Taubin 1995; Page et al. 2001; Hameiri and Shimshoni 2002], which is accurate only when the distribution of the directions to neighboring points is uniform. Alternatively, and more generally, the κ_n samples may be fit to (1) using least squares [Chen and Schmitt 1992; Hameiri and Shimshoni 2002]. Meyer et al. use a similar fit, constrained to match estimates of mean and Gaussian curvature obtained using a different technique [Meyer et al. 2002]. As shown by Goldfeather and Interrante [2004], least-squares fitting a curvature tensor to normal curvature samples is equivalent to patch fitting, with spheres as the class of surfaces being fit. This implies that, in most cases, such techniques have the same weakness as point-fitting: they become unstable when the neighbors of a vertex are close to a pair of intersecting lines.

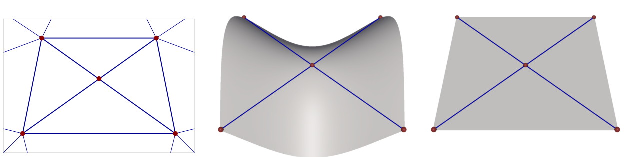


__Figure 2:__ __Left:__ degenerate configuration for algorithms that estimate curvature by fitting a parametric patch to points in a neighborhood, as well as for algorithms that use a least-squares fit of normal curvatures. Configurations similar to this one, including any number of vertices lying on two intersecting lines, will yield unstable estimates. __Center, right:__ continuous surfaces of different curvatures consistent with the given vertices.

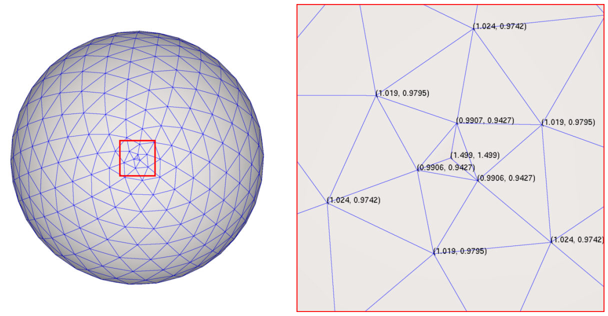


__Figure 3:__ Many "tensor averaging" algorithms produce significant errors in estimating curvature, even for densely-spaced samples and simple geometries such as a sphere. Here we show a tessellated sphere of radius 1 (which should have curvature 1 everywhere), together with the principal curvature estimates computed by the algorithm of Alliez et al. [2003]. Note the error in the curvature estimate at the central vertex: this error does not decrease as the sampling gets more dense.

- **Tensor averaging methods** compute the average of the curvature tensor over a small area of the polygonal mesh [Cohen-Steiner and Morvan 2003; Alliez et al. 2003]. The curvature of a polyhedron is zero within a face and infinite along the edges, but its average over a region of non-zero measure (such as the Voronoi region of a vertex) is finite and well-defined. Tensor averaging definitions of curvature on a mesh are elegant and free of unstable configurations, but produce large errors for certain vertex arrangements (Figure 3).

3 Algorithm

As mentioned earlier, our method for computing curvatures and derivatives of curvature is based on the common algorithm for finding per-vertex normals by averaging adjacent per-face normals. To extend this to the case of curvatures, we first define how curvature is computed over a face, then show how to combine curvature estimates expressed in terms of different coordinate systems. Finally, we describe how the algorithm generalizes to higher-order surface differentials.

3.1 Per-Face Curvature Computation

The second fundamental tensor II, already seen in equation 1, is defined in terms of the directional derivatives of the surface normal:

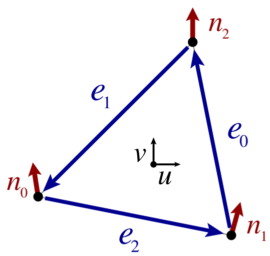
$$\mathrm{II} = \begin{pmatrix} D_u n & D_v n \end{pmatrix} = \begin{pmatrix} \frac{\partial n}{\partial u} \cdot u & \frac{\partial n}{\partial v} \cdot u \\ \frac{\partial n}{\partial u} \cdot v & \frac{\partial n}{\partial v} \cdot v \end{pmatrix}, \tag{4}$$

where (u,v) are the directions of an orthonormal coordinate system in the tangent frame (the sign convention used here yields positive curvatures for convex surfaces with outward-facing normals). Multiplying this tensor by any vector in the tangent plane gives the derivative of the normal in that direction:

$$\mathrm{II}\,\mathbf{s} = D_{\mathbf{s}} n. \tag{5}$$

Note that the derivative of the normal is itself a vector in the tangent plane: it often has a component in direction **s**, but can also have a component in the perpendicular direction (caused by "twist" in the surface).

Although this definition holds only for smooth surfaces, we can approximate it in the discretized case by using finite differences. For example, for a triangle we have three well-defined directions (the edges) together with the differences in normals in those directions (computed from the per-vertex normals). Thus, we have



$$\mathrm{II}\begin{pmatrix} e_0 \cdot u \\ e_0 \cdot v \end{pmatrix} = \begin{pmatrix} (n_2 - n_1)\cdot u \\ (n_2 - n_1)\cdot v \end{pmatrix}$$
$$\mathrm{II}\begin{pmatrix} e_1 \cdot u \\ e_1 \cdot v \end{pmatrix} = \begin{pmatrix} (n_0 - n_2)\cdot u \\ (n_0 - n_2)\cdot v \end{pmatrix}$$
$$\mathrm{II}\begin{pmatrix} e_2 \cdot u \\ e_2 \cdot v \end{pmatrix} = \begin{pmatrix} (n_1 - n_0)\cdot u \\ (n_1 - n_0)\cdot v \end{pmatrix}$$

This provides a set of linear constraints on the elements of the second fundamental tensor, which may then be determined using least squares. Note that this estimate is always well-defined, unless the triangle itself has three collinear vertices.

Although a discretization error is introduced by this finite-difference approximation, we have found that it has a high degree of accuracy in many common cases. For example, when the vertices of the triangle lie on the surface of a sphere and the vertex normals are the normals of the sphere, the curvatures produced by this technique are exact regardless of the shape of the triangle.

3.2 Coordinate System Transformation

Once we have a curvature tensor expressed in the (u_f, v_f) coordinate system of a face, we must average it with contributions from adjacent triangles. To do this, we assume that each vertex p has its own orthonormal coordinate system (u_p, v_p), defined in the plane perpendicular to its normal, and derive a change-of-coordinates formula for transforming a curvature tensor into the vertex coordinate frame.

We first consider the case when the face and vertex normals are equal, so that (u_f, v_f) and (u_p, v_p) are coplanar. The first component of II, expressed in the (u_p, v_p) coordinate system, may be found as

$$e_p = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} e_p & f_p \\ f_p & g_p \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = u_p^{\mathrm{T}}\, \mathrm{II}\, u_p. \tag{6}$$

Thus, we can find e_p in terms of the coordinates of u_p expressed in the old (u_f, v_f) coordinate system:

$$e_p = u_p^{\mathrm{T}}\, \mathrm{II}\, u_p = \begin{pmatrix} u_p \cdot u_f \\ u_p \cdot v_f \end{pmatrix}^{\mathrm{T}} \mathrm{II} \begin{pmatrix} u_p \cdot u_f \\ u_p \cdot v_f \end{pmatrix}. \tag{7}$$

Similarly, we find that $f_p = u_p^{\mathrm{T}}\, \mathrm{II}\, v_p$ and $g_p = v_p^{\mathrm{T}}\, \mathrm{II}\, v_p$.

When the old and new coordinate systems are noncoplanar, we cannot simply project the new u_p and v_p axes into the old (u_f, v_f) coordinate system. The projections would not, in general, be unit-length, which would cause a "loss" of curvature at each each change of coordinates (specifically, the mean curvature would be multiplied by the square of the cosine of the angle between the normals). Instead, we first rotate one of the coordinate systems to be coplanar with the other, rotating around the cross product of their normals. This avoids the $\cos^2\theta$ curvature loss and increases the accuracy of estimates on coarsely-tessellated surfaces.

3.3 Weighting

The question of weighting, i.e. how much of the face curvature should be accumulated at each vertex, has been addressed by prior work. Following Meyer et al. [2002], we take $w_{f,p}$ to be the portion of the area of f that lies closest to vertex p. We have found that this "Voronoi area" weighting produces the best estimates of curvature for triangles of varying sizes and shapes. This contrasts with the weights used for estimating normals, for which we take $w_{f,p}$ to be the area of f divided by the squares of the lengths of the two edges that touch vertex p. As shown by Max [1999], this produces more accurate normal estimates than other weighting approaches, and is exact for vertices that lie on a sphere.

3.4 Algorithm and Extension to Curvature Derivatives

We may now state our final algorithm for per-vertex computation of the curvature tensor:

Compute per-vertex normals
Construct an initial (u_p, v_p) coordinate system in the
  tangent plane of each vertex
for each face:
  Compute edge vectors e and normal differences Δn
  Solve for II using least squares
  for each vertex p touching the face:
    Re-express II in terms of (u_p, v_p)
    Add this tensor, weighted by $w_{f,p}$, to vertex curvature
for each vertex:
  Divide the accumulated II by the sum of the weights
  If desired, find principal curvatures and directions
    by computing eigenvalues and eigenvectors of II

One of the most important features of this algorithm is that it generalizes to higher-order differential properties. Just as the curvature tensor gives the change in the normal with motion along the surface, one may define a "derivative of curvature" tensor that gives the change in the curvature along the surface. This is a $2 \times 2 \times 2$ rank-3 tensor or "cube of numbers," and because of symmetry it has only four unique entries. Writing it as a vector of matrices, the derivative-of-curvature tensor C has the form

$$\mathrm{C} = \begin{pmatrix} D_u\mathrm{II} & D_v\mathrm{II} \end{pmatrix} = \left(\begin{pmatrix} a & b \\ b & c \end{pmatrix} \begin{pmatrix} b & c \\ c & d \end{pmatrix} \right). \tag{8}$$

Multiplying C by a direction vector *three* times gives the derivative of curvature in that direction. Although derivatives of curvature have not been applied in as many contexts as curvatures themselves, they have been used for such applications as fair surface design [Moreton and Séquin 1992; Gravesen and Ungstrup 2002], detecting creases in surfaces [Lengagne et al. 1996; Watanabe and Belyaev 2001], and producing line drawings [DeCarlo et al. 2003].

A simple extension to our curvature-estimation algorithm can be used to estimate derivatives of curvature and, where needed, any higher-order derivatives. Just as curvatures are estimated per-face by considering the differences in normals along the edges, we estimate C with a least-squares fit to the differences in the curvature tensor along the edges. The algorithm uses the change-of-coordinate-system formula to transform curvatures from vertex coordinates to face coordinates, and an analogous formula to transform the C back into vertex coordinates.

4 Results

Figures 4 and 5 show the results of computing curvatures and derivatives of curvature on a large (1.5 million polygon) scanned mesh. We found that the algorithm is efficient enough in both time (curvature computation took 4 seconds) and space (no additional connectivity data structures are required) to be practical even for data sets of this size.

Although the main goals of our algorithm are robustness and easy generalizability to derivatives of any order, we have found that the quality of the curvature estimates it produces on analytic models is competitive with other methods in the literature. Figure 6 shows results for a torus mesh, examining the effects of both noise (perpendicular to the surface) and differences in surface tessellation. The top graph of Figure 6 shows curvature error for a uniform tessellation of the sphere, for both our algorithm and contemporary algorithms in each of the three categories considered earlier, Note

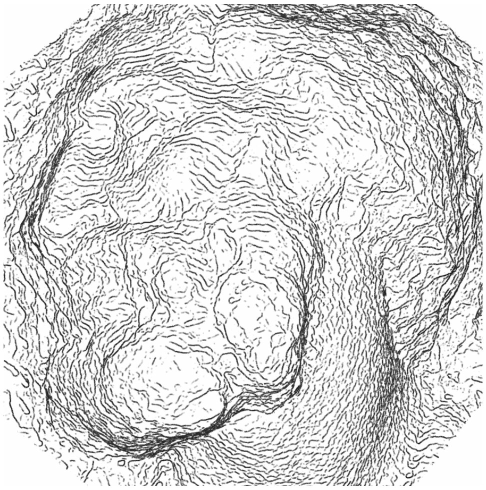


Figure 4: *Suggestive contour rendering of a model of the face of St. Matthew (1.5M polygons). To filter out scanning noise, the normal field was smoothed by a Gaussian filter of width 0.5 mm (this also had the effect of smoothing the curvature estimates). Computation of curvatures and curvature derivatives took 4.0 and 5.2 seconds, respectively.*

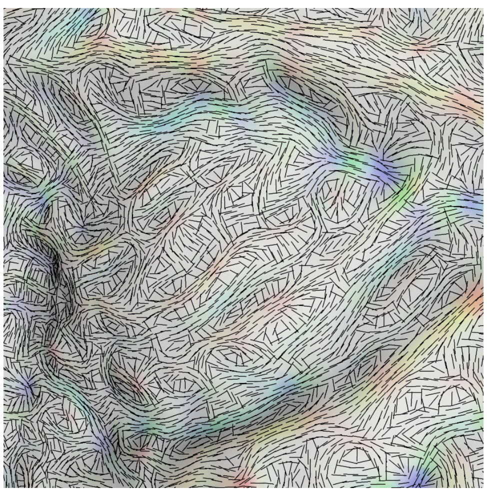


Figure 5: *Detail of minimum principal curvature directions around the eye of the St. Matthew model.*

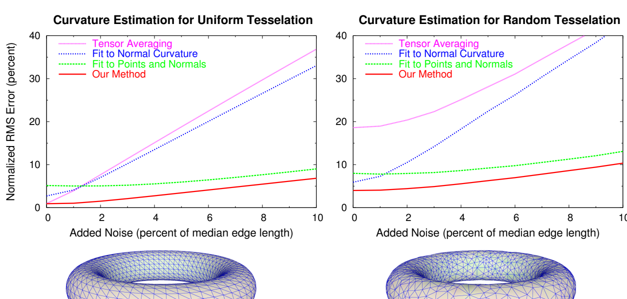


Figure 6: *Estimation errors for our curvature estimation algorithm, compared to contemporary alternatives based on tensor averaging [Alliez et al. 2003], fitting to normal curvature estimates, and patch fitting to points and normals [Goldfeather and Interrante 2004]. We show results for both regular and irregular tessellations, as the amount of noise is increased. We report the RMS difference between the estimated and exact normal curvatures, integrated over all directions and averaged over all points on the mesh. The results are averaged over 1000 trials, and results are normalized by the exact RMS curvature over the model.*

that both the normal curvature fitting and tensor averaging methods produce good results for perfect data, but degrade rapidly with the addition of noise. The bottom row of this figure shows the effect of varying the tessellation on the performance of the algorithms. We see that although the performance of all algorithms deteriorates relative to the regular tessellation, the results of the tensor averaging method, in particular, degrade significantly. The results in all cases, of course, could be improved by either pre-smoothing the mesh or by smoothing the final curvature field.

5 Conclusion

This paper presents a general algorithm for computing curvatures, derivatives of curvature, and higher-order differential properties on triangular meshes. The algorithm is efficient, robust, and free of degenerate configurations, and yields accurate estimates in the presence of irregular tessellation and moderate amounts of noise.

References

ALLIEZ, P., COHEN-STEINER, D., DEVILLERS, O., LÉVY, B., AND DESBRUN, M. 2003. Anisotropic Polygonal Remeshing. *ACM Transactions on Graphics, Vol. 22*, No. 3.

CAZALS, F., AND POUGET, M. 2003. Estimating Differential Quantities Using Polynomial Fitting of Osculating Jets. In *Proc. Symposium on Geometry Processing*.

CHEN, X., AND SCHMITT, F. 1992. Intrinsic Surface Properties from Surface Triangulation. In *Proc. European Conference on Computer Vision*.

CIPOLLA, R., AND GIBLIN, P. J. 2000. *Visual Motion of Curves and Surfaces*. Cambridge University Press.

COHEN-STEINER, D., AND MORVAN, J.-M. 2003. Restricted Delaunay Triangulations and Normal Cycle. In *Proc. Symposium on Computational Geometry*.

DECARLO, D., FINKELSTEIN, A., RUSINKIEWICZ, S., AND SANTELLA, A. 2003. Suggestive Contours for Conveying Shape. *ACM Transactions on Graphics, Vol. 22*, No. 3.

DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. H. 1999. Implicit Fairing of Irregular Meshes Using Diffusion and Curvature Flow. In *Proc. ACM SIGGRAPH*.

GOLDFEATHER, J., AND INTERRANTE, V. 2004. A Novel Cubic-Order Algorithm for Approximating Principal Direction Vectors. *ACM Transactions on Graphics, Vol. 23*, No. 1.

GRAVESEN, J., AND UNGSTRUP, M. 2002. Constructing Invariant Fairness Measures for Surfaces. *Advances in Computational Mathematics, Vol. 17*.

HAMEIRI, E., AND SHIMSHONI, I. 2002. Estimating the Principal Curvatures and the Darboux Frame from Real 3D Range Data. In *Proc. 3DPVT*.

HERTZMANN, A., AND ZORIN, D. 2000. Illustrating Smooth Surfaces. In *Proc. ACM SIGGRAPH*.

INTERRANTE, V., FUCHS, H., AND PIZER, S. 1995. Enhancing Transparent Skin Surfaces with Ridge and Valley Lines. In *Proc. IEEE Visualization*.

LENGAGNE, R., FUA, P., AND MONGA, O. 1996. Using Crest Lines to Guide Surface Reconstruction from Stereo. In *Proc. International Conference on Pattern Recognition*.

MAX, N. 1999. Weights for Computing Vertex Normals from Facet Normals. *Journal of Graphics Tools, Vol. 4*, No. 2.

MEYER, M., DESBRUN, M., SCHRÖDER, P., AND BARR, A. H. 2002. Discrete Differential-Geometry Operators for Triangulated 2-Manifolds. In *Proc. VisMath*.

MORETON, H. P., AND SÉQUIN, C. H. 1992. Functional Optimization for Fair Surface Design. *Computer Graphics (Proc. ACM SIGGRAPH 92), Vol. 26*, No. 2.

PAGE, D. L., KOSCHAN, A., SUN, Y., PAIK, J., AND ABIDI, A. 2001. Robust Crease Detection and Curvature Estimation of Piecewise Smooth Surfaces from Triangle Mesh Approximations Using Normal Voting. In *Proc. Conference on Computer Vision and Pattern Recognition*.

TAUBIN, G. 1995. Estimating the Tensor of Curvature of a Surface from a Polyhedral Approximation. In *Proc. International Conference on Computer Vision*.

TRUCCO, E., AND FISHER, R. B. 1995. Experiments in Curvature-Based Segmentation of Range Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 17*, No. 2.

WATANABE, K., AND BELYAEV, A. G. 2001. Detection of Salient Curvature Features on Polygonal Surfaces. *Computer Graphics Forum (Proc. Eurographics 2001), Vol. 20*, No. 3.