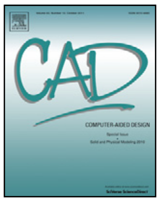
Computer-Aided Design 43 (2011) 1561–1566

Contents lists available at SciVerse ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad



# Curvature estimation for meshes based on vertex normal triangles

Mao Zhihong[a,b,*], Cao Guo[d], Ma Yanzhao[c], Kunwoo Lee[b]

[a] Department of Computer Science, Sun Yat-Sen University, 510275, China
[b] Department of Mechanical and Aerospace Engineering/SNU-IAMD, Seoul 151-742, Republic of Korea
[c] Human-Centered CAD Laboratory, Department of Mechanical and Aerospace Engineering, Seoul National University, Seoul 151-742, Republic of Korea
[d] The School of Computer Science and Technology, Nanjing University of Science and Technology, 210094, China

ARTICLE INFO

*Article history:*
Received 4 January 2011
Accepted 14 June 2011

*Keywords:*
Curvature estimation
Surface
3D meshes

ABSTRACT

The estimation of surface curvature is essential for a variety of applications in computer graphics because of its invariance with respect to rigid transformations. In this article, we describe a curvature estimation method for meshes by converting each planar triangular facet into a curved patch using the vertex positions and the normals of three vertices of each triangle. Our method interpolates three end points and the corresponding normal vectors of each triangle to construct a curved patch. Then, we compute the per triangle curvature of the neighboring triangles of a mesh point of interest. Similar to estimating per vertex normal from the adjacent per triangle normal, we compute the per vertex curvature by taking a weighted average of per triangle curvature. Through some examples, we demonstrate that our method is efficient and its accuracy is comparable to that of the existing methods.

## 1. Introduction

The development of 3D scanning techniques enables us to convert large and highly complex objects with arbitrary accuracy into a polygonal mesh. This development enables acquisition of richly detailed features of objects from a mesh model, which are helpful in shape modeling, shape analysis, and shape understanding. Some elementary differential geometry properties such as normals and curvatures are as important as surface positions for the perception and understanding of shapes. In recent studies, curvature as a fundamental descriptor for shape analysis was an important parameter for solving a variety of basic tasks in computer graphics, including surface segmentation [1,2], surface classification [3], surface smoothing [4], matching [5], reconstruction [6], re-meshing [7,8], symmetry detection [9], non-photorealistic rendering [10,11], and feature line extraction [12,13].

The ability to compute curvature from meshes is complicated by the lack of an analytical definition of surface shape. Curvature estimation methods generally fall into one of the two main categories. The first category, surface fitting, involves finding a parametric surface patch fitted to the neighborhood of each data point. This method is a popular technique for curvature estimation. The accuracy of this method always depends on the parametric model, which is usually computed using the least squares approach. The second category, the discrete method, involves developing discrete approximation formulas based on the definition of curvature to operate on the triangulated data directly; it avoids solving a least squares problem. There is no consensus on the most appropriate approach to curvature estimation on discrete meshes. From the results of some analysis test cases, we find that the fitting method has better overall performance than the discrete method, but the discrete method is appealing because of its speed.

The motivation for undertaking this study was to develop a fitting method based on vertex normal triangles to robustly estimate curvatures. Generally speaking, *vertex normal triangles*, VN triangles in short, are defined by the three vertices and three corresponding vertex normals and used for a local fitting. Our method interpolates three end points and corresponding normal vectors of each triangle to construct a curved patch. With the curved patches, we can estimate the per triangle curvature distribution as well as the curvature at the vertices. Similar to the estimation of per vertex normal from the adjacent per triangle normal, we compute the per vertex curvature by taking a weighted average of per triangle curvature. We believe that our method is the only one to estimate curvature by interpolating VN triangles. As we will show in the next sections, the specific contributions of this study are the following.

- Avoid solving a least squares problem, so our algorithm runs faster than other similar fitting methods do.
- Handle arbitrary triangulations. The method is typically free of degeneracy, and it has no under-constrained problem.

[*] Corresponding author at: Department of Computer Science, Sun Yat-Sen University, 510275, China. Tel.: +86 1086885581.
*E-mail address:* maozhh@mail.sysu.edu.cn (M. Zhihong).

0010-4485/$ – see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cad.2011.06.006

- Compute curvature per face. With the introduction of programmable geometry shaders, this per face computation can be performed directly on graphics hardware.

This paper is organized as follows. In Section 2, we review related works. In Section 3, we first employ a triangular cubic Bézier scheme to create a curved patch, and then, we estimate the curvature based on the patch. We also present formulas necessary for curvature estimation in this section. In Section 4, we evaluate various curvature estimation methods using our suite of test cases with a number of different mesh schemes. We conclude our paper in Section 5.

## 2. Related works

Reliable curvature computation has become a basic step in several computer graphics algorithms. The curvature estimation techniques for triangle meshes are usually classified in two categories: the surface fitting method and the discrete method.

*Surface fitting method*. This method yields the best approximation of the underlying surface that was the source of the triangulation. It has been one of the more popular and stable techniques for curvature estimation. Hamann [14] proved that a surface can be locally represented as a graph of a bivariate polynomial that is independent of the two unit vectors determining a local orthonormal coordinate system. The approach implies that the principal curvatures of a surface are independent of the two unit vectors. Yokoya and Levine constructed a general bivariate quadratic polynomial function approximating the vertex and its neighbors [15]. Instead of taking the quadric analytical function approach, Razdan and Bae [16] computed the curvature at a mesh vertex by a biquadratic Bézier surface. The use of the parametric form offers some advantages over the use of the analytical form, such as flexibility of the surface fitting and the ability to easily add smoothing constraints. Owing to its simplicity, the quadric surface model is incapable of accurately modeling the local surface. A variation of the quadratic fitting method is the cubic-order approximation method developed by Goldfeather and Interrante [17]. This method used neighboring points and corresponding normal vectors to fit the surface, and it performed better than the quadratic fitting method. However, the third-order fit of the surface greatly increased both time and space required for computation. From the method of Goldfeather and Interrante, we know that normal vectors of neighboring points are important factors for improving the surface fitting method. Recently, point-sampled geometry has received growing attention in computer graphics; several methods have been proposed for computing curvature information directly on point clouds [12,18].

One difference between fitting methods is the function chosen to model the local surface shape. A bivariate quadratic polynomial may simplify computation, whereas a cubic-order polynomial provides more accurate results. We can also use normal vectors (known or approximated) to improve the accuracy of the fitting method.

Another difference is the number of points fitted by the function. A minimum number of points, depending on the number of coefficients in the equation being fitted, should be provided. Otherwise, the problem is under-constrained. A larger number of points may be required to improve the stability of the solution. Our method considers each triangle of the mesh to compute the curvature and has no under-constrained problem. Moreover, unlike common surface fitting methods, our method avoids solving a least squares problem, so it runs faster.

*Discrete method*. The discrete method employs either a direct approximation formula for the curvature, or an approximation of the curvature tensor, from which curvature and other differential properties can be estimated. One of the main motivations for developing this method is to avoid the computational costs associated with the surface fitting method. Generally, the discrete method leads to some gain in computation time at the cost of attainable accuracy.

Meek and Walton [19] and Meyer et al. [20] estimated curvatures directly on the discrete triangle meshes on the basis of the Gauss–Bonnet theorem, known as the angle deficit method, which approximates Gaussian curvature as $2\pi$ minus the sum of the angles for the faces at a vertex, divided by an area associated with the vertex. The Gauss–Bonnet scheme provides a convenient and elegant formula for computing both the Gaussian and the mean curvatures. Taubin [21] estimated the tensor of curvature of surface at the vertices in a polyhedral approximation. The principal curvature directions were obtained as two eigenvectors of the symmetric matrix, and principal curvatures were obtained as linear combinations of the two eigenvalues of the same symmetric matrix. A key advantage of Taubin's method is its simplicity, with the complexity being linear in both time and space. Magid et al. [22] considered two modifications in Taubin's method: (1) the coefficient $\omega_{ij}$ was selected to be proportional to the angles instead of the surface areas; (2) the normal curvature $\kappa_{ij}$ was selected as an average of the adjacent directional curvature values. Zhang [23] thought that large errors could be introduced in Taubin's method because it did not employ normal vectors of neighboring points. Therefore, he used the normal vectors of neighboring points to construct a normal section circle and estimated the normal curvature from the positions and normal vectors of two points—the object point and one of its neighbors. Theisel et al. [24] and Rusinkiewicz [25] estimated the curvature tensor per triangle (together with the normals in its vertices), instead of computing the tensor per vertex. With the introduction of programmable geometry shaders, this per face computation can be performed directly on graphics hardware. Griffin [26] introduced a graphics processing unit (GPU) algorithm to extend Rusinkiewicz's method [25] to deformable models. These per face computation methods compute the curvatures by applying a linear interpolation of triangle points and the corresponding normals. A discretization error is introduced by the linear interpolation. Our method interpolates each triangle by a Bézier patch to estimate the per triangle curvature and has a higher degree of accuracy.

Most papers [17,22,23] provided comparisons between different methods. Overall, they concluded that the fitting methods perform better than the discrete methods, but the discrete methods are appealing because of their speed. Among all curvature estimation methods proposed in recent years, Goldfeather and Interrante's method employs normal vectors and a cubic polynomial function and is probably the best.

## 3. Curvature estimation based on vertex normal triangles

All the fitting approaches mentioned in Section 2 have one feature in common that they locally approximate the surface by a least squares technique. In this section, we propose an alternative approach to estimate the curvature: instead of computing the curvature per point, we do the computation per triangle. We consider each neighboring triangle (VN triangle) of a mesh point and interpolate the VN triangle by a triangular cubic Bézier patch; then, we estimate the curvature by Eqs. (3.7)–(3.10).

### *3.1. Triangular Bézier patch net on a vertex normal triangle*

Several local parametric triangular surface schemes have been developed over the years [27,28]. Details of fitting a surface to triangulated data can be found in [29].

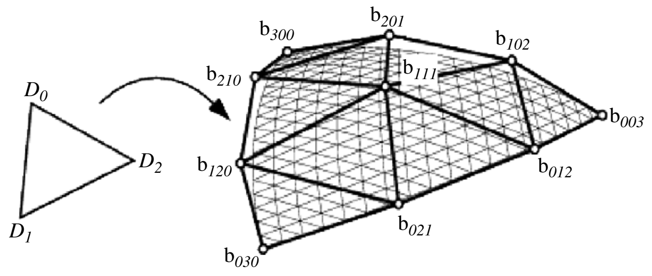


**Fig. 1.** Triangular cubic Bézier patch control nets.

### 3.1.1. *Triangular Bézier patch*

Bézier curves are expressed in terms of Bernstein polynomials, defined explicitly as

$$f(t)=\sum_{i=0}^{n}\mathbf{b}_iB_i^n(t),\qquad B_i^n(t)=\binom{n}{i}t^i(1-t)^{n-i}. \tag{3.1}$$

Bézier surfaces are of two types: rectangular and triangular Bézier patches. A triangular Bézier surface of degree $n$ with control point $\mathbf{b}_{i,j,k}$ is defined as

$$F(u,v,w)=\sum_{i=0}^{n}\sum_{j=0}^{n-i}\mathbf{b}_{i,j,k}B_{i,j,k}^n(u,v,w);$$
$$B_{i,j,k}^n(u,v,w)=\frac{n!}{i!j!k!}u^iv^jw^k\quad 0\le u,v,\ w\le 1;$$
$$u+v+w=1;\ i+j+k=n;\ i,j,\ k\ge 0. \tag{3.2}$$

### 3.1.2. *Bézier control net on a vertex normal triangle*

The input of our algorithm is the neighboring VN triangles of mesh points. If no exact normals are available, we can find per vertex normal by averaging adjacent per face normal of the triangular facets. For a triangular cubic Bézier patch, we need to compute the Bézier control points $\mathbf{b}_{i,j,k}$ (see Fig. 1). We group the points $\mathbf{b}_{i,j,k}$ together as

end vertices (three points of the input triangle): $\mathbf{b}_{003}$, $\mathbf{b}_{030}$, $\mathbf{b}_{300}$
boundary vertices: $\mathbf{b}_{210}$, $\mathbf{b}_{120}$, $\mathbf{b}_{102}$, $\mathbf{b}_{201}$, $\mathbf{b}_{021}$, $\mathbf{b}_{012}$
center vertex: $\mathbf{b}_{111}$.

For each edge of a triangle, we have points $\mathbf{V_0}$, $\mathbf{V_1}$ and normals $\mathbf{N_0}$, $\mathbf{N_1}$. In order to compute the boundary control points, we first compute the plane $\mathbf{P}$ passing through $\mathbf{V_1}-\mathbf{V_0}$ and $(\mathbf{N_0}+\mathbf{N_1})/2$. Project $\mathbf{N_0}$, $\mathbf{N_1}$ onto the plane $\mathbf{P}$ to get $\mathbf{N_0'}$, $\mathbf{N_1'}$. Then, our choice of boundary control points is as follows (see the left side of Fig. 2).

1. $\mathbf{b_0}=\mathbf{V_0}$, $\mathbf{b_3}=\mathbf{V_1}$.
2. Rotate $\mathbf{N_0'}$, $\mathbf{N_1'}$ clockwise through 90° around the normal $\mathbf{N}_P$ of the plane $\mathbf{P}$ to get two tangents $\mathbf{T_0}$, $\mathbf{T_1}$.
3. $\mathbf{p}_1$ and $\mathbf{p}_2$ trisect the edge $\mathbf{V_0V_1}$. Projecting $\mathbf{p}_1$ into the line $\mathbf{T_0}$ and $\mathbf{p}_2$ into the line $\mathbf{T_1}$, we get boundary control points $\mathbf{b_1}$ and $\mathbf{b_2}$, respectively:

$$b_1=V_0+\frac{[(V_1-V_0)\bullet\mathbf{T}_0]}{3|\mathbf{T}_0|}\frac{\mathbf{T}_0}{|\mathbf{T}_0|}$$
$$b_2=V_1-\frac{[(V_1-V_0)\bullet\mathbf{T}_1]}{3|\mathbf{T}_1|}\frac{\mathbf{T}_1}{|\mathbf{T}_1|}. \tag{3.3}$$

Next, we need to determine the center control vertex (see the right side of Fig. 2). Computing the center point $\mathbf{V}_m$ of an edge $\mathbf{V}_0\mathbf{V}_1$ by Eq. (3.1) at $t=1/2$, we set its normal $\mathbf{N}_m=(\mathbf{N_0}+\mathbf{N_1})/2$. Rotate $\mathbf{N}_m$ clockwise through 90° to get the tangent $\mathbf{T}_m$. Together with the opposite vertex, the center point $\mathbf{b}_m$ is determined as

$$\mathbf{b}_m=\mathbf{V}_m-\frac{[(\mathbf{V}_m-\mathbf{V_2})\bullet\mathbf{T}_m]}{3|\mathbf{T}_m|}\frac{\mathbf{T}_m}{|\mathbf{T}_m|}. \tag{3.4}$$

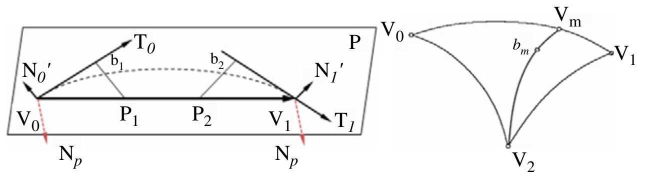


**Fig. 2.** Construction of a cubic Bézier curve and a triangular cubic Bézier patch.

Similarly, we can determine the two center points for other edges and opposite vertices. The three values are averaged to form the center control point $b_{111}$.

## 3.2. *Curvature estimation based on vertex normal triangles*

Our method does the estimation per triangle by a triangular Bézier patch. It avoids solving a least squares problem. First, we begin with a review of several important definitions and theorems from differential geometry.

### 3.2.1. *Essential terms of differential geometry [30,31]*

For the discussion below, we will consider the local behavior of a surface $\mathbf{S}(u,v)$ at a single point $\mathbf{P}$ with a given local parameterization $(u,v)$. We can express the first fundamental form as

$$\mathbf{I}=E\mathrm{d}u^2+2F\mathrm{d}u\mathrm{d}v+G\mathrm{d}v^2$$
$$E=\mathbf{S}_u\cdot\mathbf{S}_u,\qquad F=\mathbf{S}_u\cdot\mathbf{S}_v,\qquad G=\mathbf{S}_v\cdot\mathbf{S}_v \tag{3.5}$$

and the second fundamental form as

$$\mathbf{II}=L\mathrm{d}u^2+2M\mathrm{d}u\mathrm{d}v+N\mathrm{d}v^2$$
$$L=\frac{(\mathbf{S}_u,\mathbf{S}_v,\mathbf{S}_{uu})}{|\mathbf{S}_u\times\mathbf{S}_v|},\qquad M=\frac{(\mathbf{S}_u,\mathbf{S}_v,\mathbf{S}_{uv})}{|\mathbf{S}_u\times\mathbf{S}_v|},$$
$$N=\frac{(\mathbf{S}_u,\mathbf{S}_v,\mathbf{S}_{vv})}{|\mathbf{S}_u\times\mathbf{S}_v|}. \tag{3.6}$$

An alternative way of computing the principal curvature values and directions is by using the Weingarten operator $W$, also known as the shape operator. $W$ is the inverse of the first fundamental form multiplied by the second fundamental form:

$$W=\mathbf{I^{-1}II}=\frac{1}{EG-F^2}\begin{bmatrix}GL-FM & GM-FN\\ EM-FL & EN-FM\end{bmatrix}. \tag{3.7}$$

Then, we compute the eigendecomposition of $W$. The eigenvalues are the principal curvatures, and the eigenvectors (the coordinate coefficients over the given parametric basis vectors) are the principal directors.

$$\kappa_{1,2}=\frac{GL-2FM+EN}{(2EG-F^2)}\mp\frac{\sqrt{(GL-2FM+EN)^2-4(EG-F^2)(LN-M^2)}}{2(EG-F^2)}$$

$$\begin{bmatrix}\mathrm{d}u\\ \mathrm{d}v\end{bmatrix}=\begin{bmatrix}\frac{1}{2}(EN-GL)+(EG-F^2)\sqrt{(H^2-K)}\\ FL-EM\end{bmatrix},\ \min$$
$$\begin{bmatrix}\mathrm{d}u\\ \mathrm{d}v\end{bmatrix}=\begin{bmatrix}GM-FN\\ \frac{1}{2}(EN-GL)+(EG-F^2)\sqrt{(H^2-K)}\end{bmatrix},\ \max \tag{3.8}$$

$H$ is the mean curvature:

$$H=\frac{\kappa_1+\kappa_2}{2}=\frac{GL-2FM+EN}{2(EG-F^2)}=\frac{\mathrm{Trace}(W)}{2} \tag{3.9}$$

$K$ is the Gaussian curvature:

$$K=\kappa_1\kappa_2=\frac{LN-M^2}{EG-F^2}=\mathrm{Det}(W). \tag{3.10}$$

### 3.2.2. *Curvature estimation based on vertex normal triangles*

Now, the neighboring triangle of the vertex under consideration can be locally approximated by $F(u,v,w)=\sum_{i=0}^{3}\sum_{j=0}^{3-i}\mathbf{b}_{i,j,k}B^3_{i,j,k}(u,v,w)$. Set $w = 1-u-v$ and replace $B^3_{i,j,k}(u,v,w)$ with Eq. (3.2):

$$F(u,v)=\sum_{i=0}^{3}\sum_{j=0}^{3-i}\frac{3!\mathbf{b}_{i,j,k}}{i!j!k!}u^iv^j(1-u-v)^k. \tag{3.11}$$

The first fundamental form is completely defined by its first-order partials.

$$\begin{aligned}F_u &= \sum_{i=1}^{3}\sum_{j=0}^{3-i}\frac{3!\mathbf{b}_{i,j,k}}{(i-1)!j!k!}u^{i-1}v^j(1-u-v)^k\\ &\quad-\sum_{i=0}^{2}\sum_{j=0}^{2-i}\frac{3!\mathbf{b}_{i,j,k}}{i!j!(k-1)!}u^iv^j(1-u-v)^{k-1}\\ F_v &= \sum_{j=1}^{3}\sum_{i=0}^{3-j}\frac{3!\mathbf{b}_{i,j,k}}{(j-1)!i!k!}u^iv^{j-1}(1-u-v)^k\\ &\quad-\sum_{j=0}^{2}\sum_{i=0}^{2-j}\frac{3!\mathbf{b}_{i,j,k}}{i!j!(k-1)!}u^iv^j(1-u-v)^{k-1}.\end{aligned} \tag{3.12}$$

The second fundamental form is defined by its first- and second-order partials:

$$\begin{aligned}F_{uu} &= \sum_{i=2}^{3}\sum_{j=0}^{3-i}\frac{3!\mathbf{b}_{i,j,k}}{(i-2)!j!k!}u^{i-2}v^j(1-u-v)^k\\ &\quad-\sum_{i=1}^{2}\sum_{j=0}^{2-i}\frac{2*3!\mathbf{b}_{i,j,k}}{(i-1)!j!(k-1)!}u^{i-1}v^j(1-u-v)^{k-1}\\ &\quad+\sum_{i=0}^{1}\sum_{j=0}^{1-i}\frac{3!\mathbf{b}_{i,j,k}}{(i-1)!j!(k-2)!}u^iv^j(1-u-v)^{k-2}\\ F_{uv} &= \sum_{i=1}^{3}\sum_{j=1}^{3-i}\frac{3!\mathbf{b}_{i,j,k}}{(i-1)!(j-1)!k!}u^{i-1}v^{j-1}(1-u-v)^k\\ &\quad-\sum_{i=1}^{2}\sum_{j=0}^{2-i}\frac{3!\mathbf{b}_{i,j,k}}{(i-1)!j!(k-1)!}u^{i-1}v^j(1-u-v)^{k-1}\\ &\quad-\sum_{i=0}^{2}\sum_{j=1}^{2-i}\frac{3!\mathbf{b}_{i,j,k}}{i!(j-1)!(k-1)!}u^iv^{j-1}(1-u-v)^{k-1}\\ &\quad+\sum_{i=0}^{1}\sum_{j=0}^{1-i}\frac{3!\mathbf{b}_{i,j,k}}{i!j!(k-2)!}u^iv^j(1-u-v)^{k-2} \end{aligned}\tag{3.13}$$

$$\begin{aligned}F_{vv} &= \sum_{j=2}^{3}\sum_{i=0}^{3-j}\frac{3!\mathbf{b}_{i,j,k}}{(j-2)!i!k!}u^iv^{j-2}(1-u-v)^k\\ &\quad-\sum_{j=1}^{2}\sum_{i=0}^{2-j}\frac{2*3!\mathbf{b}_{i,j,k}}{(j-1)!i!(k-1)!}u^iv^{j-1}(1-u-v)^{k-1}\\ &\quad+\sum_{j=0}^{1}\sum_{i=0}^{1-j}\frac{3!\mathbf{b}_{i,j,k}}{i!j!(k-2)!}u^iv^j(1-u-v)^{k-2}.\end{aligned}$$

We follow Eqs. (3.7)–(3.10), (3.12) and (3.13) to get the barycenter curvature per triangle and then extend the common algorithm for finding per vertex normal by averaging adjacent per face normal to the case of curvatures. Usually, the weights $\omega_i$ are selected to be the same or proportional to the surface areas of the triangles. Rusinkiewicz [25] followed the method of Meyer et al. [20] and found that the "Voronoi area" weighting produces the best estimation of curvature for triangles of varying sizes and shapes.

Finally, we summarize our algorithm in Fig. 3.

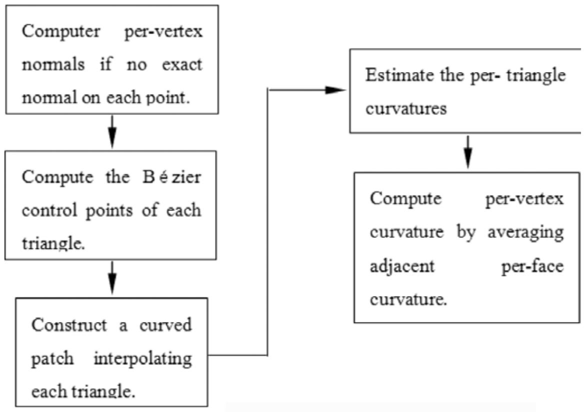

Computer per-vertex normals if no exact normal on each point.
Compute the Bézier control points of each triangle.
Construct a curved patch interpolating each triangle.
Estimate the per- triangle curvatures
Compute per-vertex curvature by averaging adjacent per-face curvature.

**Fig. 3.** The diagram of our algorithm.

## 4. Experimental results

A comparison of the state-of-the-art curvature estimation methods [17,22,23] reveals that Goldfeather and Interrante's method [17] employs the cubic-order polynomial function and normal vectors and is perhaps the best in recent years. Taubin's approach [21] is a representative of the discrete methods; it is not the best, but it runs fast. In order to investigate the performance of the proposed algorithm in Section 3, we compare our method with the two above-mentioned methods in the case of two categories of data: synthetic models and discrete meshes. We measure the curvature errors by the absolute average curvature errors at all points.

### 4.1. *Comparison with synthetic data*

First, we test the proposed algorithm on synthetic models. A sphere is represented as

$$\mathbf{x}^2+\mathbf{y}^2+\mathbf{z}^2=9.83^2$$

and the equation of a torus is

$$r(u,v)=[(3+\cos u)\cos v,(3+\cos u)\sin v, r\sin u]\quad (0\le u\le 2\pi, 0\le v\le 2\pi).$$

For synthetic models, we generated several polyhedral approximations with a varying number of triangles as shown in Fig. 4. Gaussian and mean curvatures have differential invariant properties and are extensively used in the computer graphics domain. Therefore, we consider performing the comparison tests based on Gaussian and mean curvatures.

Of course, the surface normal is very important for calculating the surface curvature. The exact surface normal will decrease the curvature estimation error. In order to gain insight into the factor, we compared the results of using approximated surface normals and exact surface normals on the sphere model. The approximated normals are calculated as the weighted average of the adjacent face normal.

The results of our testing are summarized in Figs. 5 and 6. Overall, our method and Goldfeather and Interrante's method work better than Taubin's method for the mean curvature estimation and Taubin's method does better when the Gaussian curvature is computed.

In the case of the mean curvature estimation with the calculated normals, Goldfeather and Interrante's method outperforms our method, and our method performs slightly better than the former for the Gaussian curvature estimation. The reason is that Goldfeather and Interrante's method approximates the local surface based on the neighborhood of the target point and our

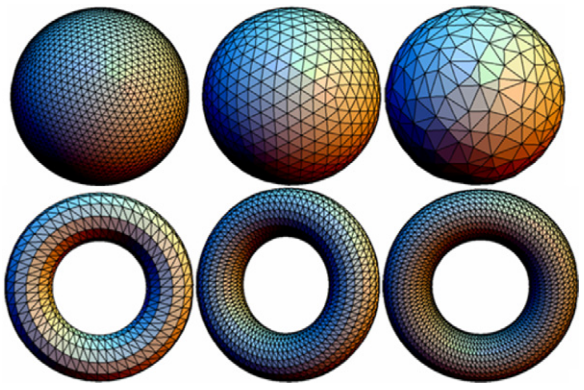


**Fig. 4.** Synthetic sphere and torus models for curvature estimation tests. Sphere models (from left to right): with 2562 triangles, with 642 triangles, with 313 triangles. Torus models (from left to right): with 600 triangles, with 1660 triangles, with 2500 triangles.

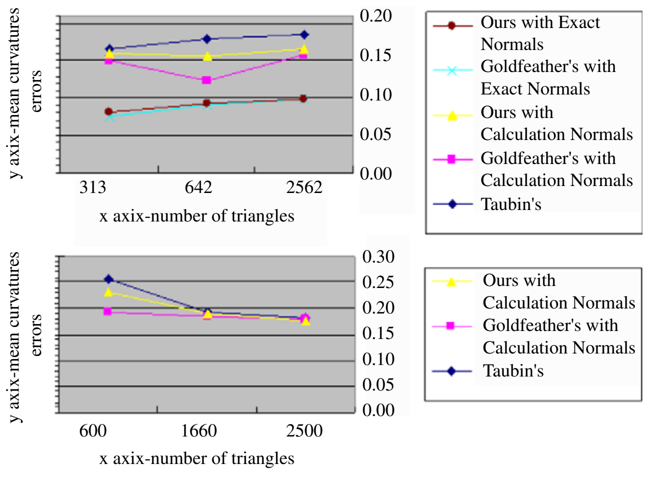


**Fig. 5.** Mean curvatures estimated by different methods. For synthetic surfaces, we generated several polyhedral approximations with a varying number of triangles (*x*-coordinate: number of triangles; *y*-coordinate: mean curvature errors). Top: the sphere model; Bottom: the torus model.

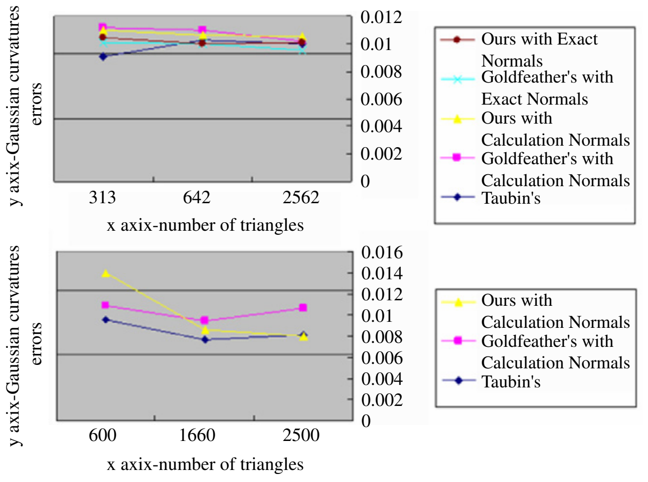


**Fig. 6.** Gaussian curvatures estimated by different methods. For synthetic surfaces, we generated several polyhedral approximations with a varying number of triangles (*x*-coordinate: number of triangles; *y*-coordinate: Gaussian curvature errors). Top: the sphere model; Bottom: the torus model.

method only interpolates each vertex normal triangle. Figs. 5 and 6 also show that both our method and Goldfeather and Interrante's method produce almost the same results in the case of the mean and Gaussian curvature estimation with the exact normal.

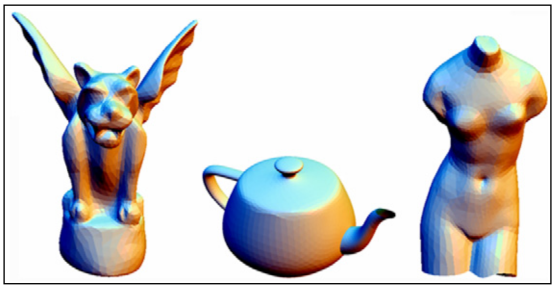


**Fig. 7.** Discrete mesh models for curvature estimation tests. Left: Gargoyle model with 4002 triangles; center: Teapot model with 4255 triangles; right: Venus model with 2817 triangles.

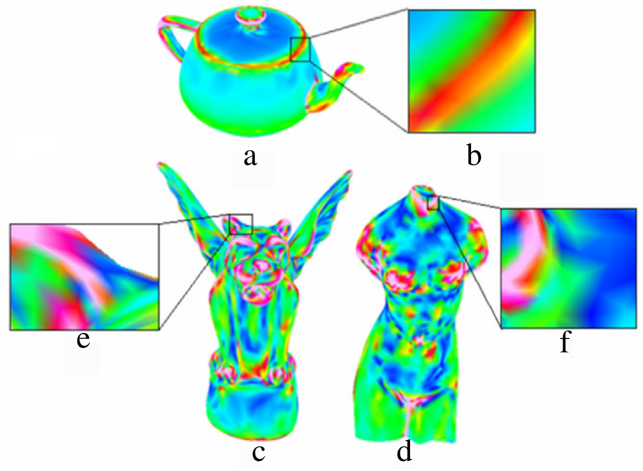


**Fig. 8.** Visualization of the mean curvature of each mesh vertex with color. Warm colors correspond to the sharp features and cool colors correspond to the flat regions. (a) Teapot model; (b) enlarged view of lid brim area in (a); (c) gargoyle model; (d) Venus model; (e) enlarged view of ear in (c); (f) enlarged view of neck in (d).

## 4.2. Comparison with discrete mesh

In this section, we test our algorithm on discrete mesh models. The images of the tested objects are shown in Fig. 7. The gargoyle, teapot, and Venus models have 4002, 4255, and 2817 triangles, respectively.

In Fig. 8, we visualize the magnitude of the mean curvature values of each point with color. Warm colors correspond to the sharp features and cool colors correspond to the flat regions. For the mesh models, the salient features are identified as regions of high curvature, which are usually the characteristic of part boundaries, such as the lid brim of the teapot model (see Fig. 8(b)), eyes, nose, and ears of the gargoyle model (see Fig. 8(e)), and shoulders, breasts, and neck of the Venus model (see Fig. 8(f)). The mean curvatures estimated by our method can visualize the sharp features of the discrete mesh models reliably. The CAD surfaces of revolution in Fig. 9 are derived from NURBs surfaces by CAD modeling environment. The results of mean curvature errors testing on CAD model are shown in Fig. 10.

We do not approximate the neighborhood of the target point using a least squares approach. Instead, we interpolate three end points and the corresponding normal vectors of each triangle to construct a curved patch. Therefore, our method runs faster. Table 1 lists the times for different curvature estimation methods on the Venus, gargoyle, and teapot models. All of our tests were performed on a PC with Intel® Core$^{TM}$ 2 Duo CPU 3.0 GHz processor and 2.0 GB of RAM memory.

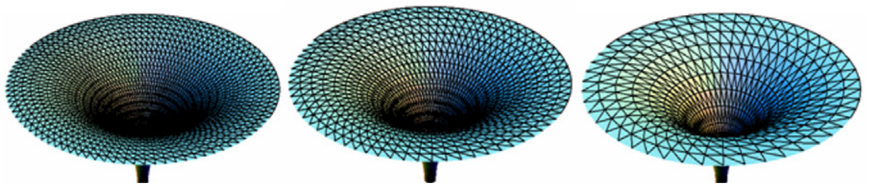


**Fig. 9.** The CAD surfaces of revolution (from left to right): with 4608 triangles, with 2592 triangles, with 1152 triangles.

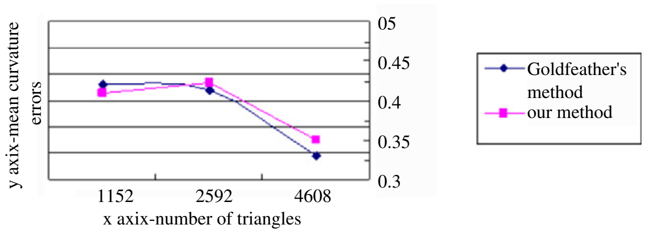


**Fig. 10.** Comparison on CAD surfaces of revolution. (*x*-coordinate: number of triangles; *y*-coordinate: mean curvature errors).

**Table 1**
Times for different methods on discrete models (time unit: second).

| Model | Number of triangles | Time for Goldfeather and Interrante's method | Time for our method | Time for Taubin's |
|---|---|---|---|---|
| Venus | 2817 | 331.66 | 6.15 | 6.24 |
| Gargoyle | 4002 | 491.87 | 9.11 | 8.90 |
| Teapot | 4255 | 493.60 | 9.22 | 9.36 |

## 5. Conclusions

A curvature estimation technique based on vertex normal triangles has been presented. This technique constructs a triangular cubic Bézier patch that interpolates three end points and corresponding normal vectors of each triangle. With the curved patches, we estimate the per triangle curvature. Similar to the estimation of per vertex normal from the adjacent per triangle normal, we compute the per vertex curvature by taking a weighted average of the curvatures of triangles touching a mesh point. The proposed method avoids solving a least squares problem and runs faster. Moreover, the method can handle arbitrary triangulations, is typically free of degeneracy, and has no under-constrained problem. Test results show the algorithm is efficient and yields accurate estimations in the case of both synthetic data and discrete meshes.

## Acknowledgments

This work was supported in part by the Basic Science Research Program through the National Research Foundation of Korea funded by the Ministry of Education, Science and Technology (2010-0001645); in part by the Doctoral Fund of Ministry of Education of China for new teachers (20100171120045); in part by the International Academic Program of The Korea Foundation for Advanced Studies 2010–2011; in part by the National Natural Science Foundation of China (61003108).

## References

[1] Attene M, Robbiano F, Spagnuolo M, Falcidieno B. Characterization of 3D shape parts for semantic annotation. Comput Aided Des 2009;41(10):756–63.
[2] Lavoue G, Dupont F, Baskurt A. A new CAD mesh segmentation method based on curvature tensor analysis. Comput Aided Des 2005;37(10):975–87.
[3] Lai YK, Zhou QY, Hu SM, Wallner J, Pottmann H. Robust feature classification and editing. IEEE Trans Vis Comput Graphics 2007;13(1):34–45.
[4] Lange C, Polthier K. Anisotropic smoothing of point sets. Comput Aided Geom Design 2005;22(7):680–92.
[5] Huang OX, Flory S, Gelfand N, Hofer M, Pottmann H. Reassembling fractured objects by geometric matching. ACM Trans Graph 2006;25(3):569–78.
[6] Jenke P, Wand M, Straber W. Patch-graph reconstruction for piecewise smooth surfaces. In: Proc. of vision, modeling and visualization. 2008. p. 3–12.
[7] Lai YK, Kobbelt L, Hu SM. An incremental approach to feature aligned quad-dominant remeshing. In: Proc. ACM symp. solid and physical modeling. 2008. p. 137–45.
[8] Alliez P, Cohen-Steiner D, Devillers O, Levy B, Desbrun M. Anisotropic polygonal remeshing. ACM Trans Graph 2003;22(3):485–93.
[9] Mitra NJ, Guibas LJ, Pauly M. Partial and approximate symmetry detection for 3D geometry. ACM Trans Graph 2006;25(3):560–8.
[10] Cole F, Sanik K, Decarlo D, Finkelstein A, Funkhouser T, Rusinkiewicz S, et al. How well do line drawings depict shape? ACM Trans Graph 2009;28(3):1–9.
[11] Judd T, Durand F, Adelson EH. Apparent ridges for line drawing. ACM Trans Graph 2007;26(3):19–26.
[12] Merigot Q, Ovsjanikov M, Guibas L. Robust Voronoi-based curvature and feature estimation. In: Conf. geometric and physical modeling. 2009. p. 1–12.
[13] Kolomenkin M, Shimshoni I, Tal A. Demarcating curves for shape illustration. ACM Trans Graph 2008;27(5):157–66.
[14] Hamann B. Curvature approximation for triangulated surfaces. In: Computing supplements. 1993. p. 139–53.
[15] Yokoya N, Levine MD. Range image segmentation based on differential geometry: a hybrid approach. IEEE Trans Pattern Anal Mach Intell 1989;11(6):643–9.
[16] Razdan A, Bae MS. Curvature estimation scheme for triangle meshes using biquadratic Bezier patches. Comput Aided Des 2005;37(14):1481–91.
[17] Goldfeather J, Interrante V. A novel cubic-order algorithm for approximating principal direction vectors. ACM Trans Graph 2004;23(1):45–63.
[18] Yang P, Qian X. Direct computing of surface curvatures for point-set surfaces. In: Proc. of SBG. 2007. p. 9–36.
[19] Meek DS, Walton DJ. On surface normal and Gaussian curvature approximations given data sampled from a smooth surface. Comput Aided Geom Design 2000;17(6):521–43.
[20] Meyer M, Desbrun M, Schroder P, Barr A. Discrete differential-geometry operators for triangulated 2-manifolds. Vis Math 2003;III:35–57.
[21] Taubin G. Estimating the tensor of curvature of a surface from a polyhedra approximation. In: ICCV. 1995. p. 902–7.
[22] Magid E, Soldea O, Rivlin E. A comparison of Gaussian and mean curvature estimation methods on triangular meshes of range image data. Comput Vis Image Underst 2007;107(3):139–59.
[23] Zhang X, Li H, Cheng Z. Curvature estimation of 3D point cloud surfaces through the fitting of normal section curvatures. In: ASIAGRAPH Proceedings. 2008. p. 72–9.
[24] Theisel H, Rossl C, Zayer R, Seidel H-P. Normal based estimation of the curvature tensor for triangular meshes. In: Conference PG. 2004. p. 288–97.
[25] Rusinkiewicz S. Estimating curvatures and their derivatives on triangle meshes. In: Proc. 3D data processing, visualization and transmission. 2004. p. 486–93.
[26] Griffin W. Real-time curvature estimation on deformable models. 2010.
[27] Nielson G. A Transfinite visually continuous, triangular interpolant. In: Farin G, editor. Geometric modeling: algorithms and new trends. SIAM; 1987. p. 235–45.
[28] Liu Y, Mann S. Parametric triangular Bézier surface interpolation with approximate continuity. In: Proc. 2008 ACM symposium on solid and physical modeling. 2008. p. 381–7.
[29] Mann S, Loop C, Lounsbery M. A survey of parametric scattered data fitting using triangular interpolants. In: Hagen H, editor. Curve and surface modeling. SIAM; 1992.
[30] Sinha SS, Besl PJ. Principal patches—a viewpoint-invariant surface description. In: Proc. of robotics and automation. 1990. p. 226–31.
[31] Do Carmo MP. Differential geometry of curves and surfaces. China Machine Press; 2004.