The Visual Computer manuscript No.
(will be inserted by the editor)

Harlen Costa Batagelo · Wu, Shin - Ting

Estimating Curvatures and their Derivatives on Meshes of Arbitrary Topology from Sampling Directions

Abstract Estimation of local differential geometry properties becomes an important processing step in a variety of applications, ranging from shape analysis and recognition to photorealistic image rendering. This paper presents yet another approach to compute those properties, with comparable numerical and accuracy performance to the previous works. The key difference of our approach is simplicity, which makes it directly implementable on the GPU. Experimental results are provided to underpin our statement.

Keywords Curvatures · Principal Directions · Differential Geometry Properties · Computational Aided Geometric Design

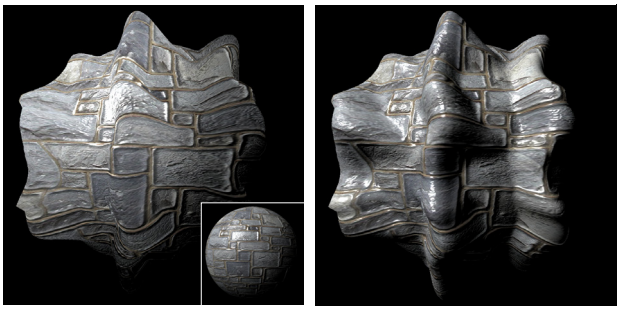


Fig. 1 Lighting a bump-mapped sphere deformed on the GPU. Left: Using differential geometry properties of the original non-deformed sphere (inset). Right: Using updated properties computed on the GPU after the deformation.

1 Introduction

The differential geometric properties characterize the vicinity of each point of a surface, such as the area, the shape, the maximum and minimum of the normal curvature, and the principal directions associated to these curvatures. Some of these quantities are of interest in image synthesis and analysis applications, such as anisotropic fairing of digital models (13), extraction of visually suggestive contours in order to enhance shape cues (7), enrichment of visual appearance with fine-scale details (20) and hatching strokes in pen-and-ink style painting (17). In the context of 3D interactions, they have also been shown useful for weighting the gravity function that causes the cursor to snap to ridges or valleys of a surface (21).

The advent of highly capable GPUs has propelled researches on estimation of local geometric properties on such hardware. Today, real-time applications may employ GPU shader programs to efficiently deform geometry immediately before rendering. Hence, the associated per-vertex geometric properties, such as the tangent vectors, normal vectors, principal curvatures and directions, need to be updated accordingly for using 3D detail mapping techniques and performing correct lighting calculations. Figure 1 shows a bump-mapped sphere deformed in the vertex shader by a Perlin Noise displacement function and lit using Phong shading. In the left, the sphere is lit without updating the tangent frames after the deformation (the tangent frames are the same of the non-deformed sphere, shown in the inset). In the right, the sphere is accurately lit with tangent frames computed in real-time on the GPU after the per-vertex deformation. For rendering 3D details with correct silhouettes, higher order quantities should be computed on-the-fly as well (20).

Basically, we may distinguish three approximation techniques for computing differential geometry quantities of discrete surfaces: patch-fitting, averaging, and curve sampling methods. Averaging and curve sampling techniques are defended, respectively, by Rusinkiewicz (18), Agam and Tang (1). They claim that patch-fitting techniques tend to smooth sharp corners and ridges, often requiring connectivity data structure beyond the usual 1-ring of vertices or faces around each vertex. Inspired by the accurate results achieved by Max for the per-vertex estimation of normal vectors (14), Rusinkiewicz proposes to take a weighted average of the geometric quantities of all adjacent faces of each point of interest for calculating its curvatures and higher order deriva-

Harlen Costa Batagelo and Wu, Shin-Ting
School of Electrical and Computer Engineering, Unicamp
Tel.: +55-19-35213795
Fax: +55-19-35213845
E-mail: {harlen,ting}@dca.fee.unicamp.br

tives. His proposal requires that normal vectors at each vertex are provided and that the weights are appropriately selected. Agam and Tang show that the curve sampling method may accurately compute both the normal vectors and the curvatures in the same framework. They point out that, as these methods have a large degree of freedom, they are capable of modeling the local surface geometry much more flexibly compared with other techniques. However, instead of applying the formulation of the Weingarten equations, they determine the extrema of the normal curvatures by computing the normal curvatures of all sampling curves and, then, selecting the directions that presents minimum and maximum normal curvatures. The quality of the curvature estimation depends, therefore, on the sampling coverage.

We present in this work a proposal for computing the local geometry properties of second or higher order derivatives on the basis of a set of neighboring sampling points that lie on the distinct directions with respect to a vertex of interest. It differs from the work of Agam and Tang in that we do not select the extreme values of the normal curvature among the samples. Adopting the scheme devised by Rusinkiewicz, we use the quadratic differential forms and the finite-difference form of the shape operator to compute the maximum and the minimum of the normal curvature. Consequently, the results tend to be more stable using only the 1-ring neighborhood. We, however, take advantage of the curve sampling approach to improve efficiency of the algorithm devised by Rusinkiewicz. Instead of two processing passes – one over adjacent faces and one over vertices, we present a procedure that requires only one pass. From the experimental results, we observed that the gain in efficiency compensates the slight degradation in accuracy for irregularly sampled meshes.

To be self-containing, a brief overview of surface differential geometry is provided in Section 2. Section 3 presents related works, while Section 4 describes our proposal for estimating curvatures and their derivatives on the curve sampling basis. Although surface normals play an important role in the curvature estimation (8), we will consider in this work that the tangent frames, composed by normal, tangent and bitangent vectors, are somehow accurately computed. This is because that there is a variety of efficient algorithms for computing these elements of first derivatives. In Section 5, comparisons of our proposal to previous work in terms of efficiency and robustness are given. Finally, some concluding remarks are drawn in Section 6.

2 Background

We may represent a surface $\mathscr{S}$ as a net of parametric curves, such that every point $\mathscr{P}$ on the surface is a crossing point of two of them. Mathematically, we may express $\mathscr{S}$ as a function $\mathbf{r}(u,v) = (x(u,v), y(u,v), z(u,v))$ that maps a point (u,v) in a certain closed interval Ω onto a three-dimensional space $\Re^3$. The real variables u, v are called *coordinates* on $\mathscr{S}$. At $\mathscr{P}$ the vector $\mathbf{r}_u = \frac{\partial \mathbf{r}}{\partial u}$ is tangent to the curve $\mathbf{r}(u, v_{constant})$, and $\mathbf{r}_v = \frac{\partial \mathbf{r}}{\partial v}$ is tangent to the curve $\mathbf{r}(u_{constant}, v)$. If the vectors $\mathbf{r}_u$ and $\mathbf{r}_v$ do not vanish and have different directions at every point, we say that $\mathbf{r}(u,v)$ is a *regular surface*.

Let $\alpha(t) = \mathbf{r}(u(t), v(t))$ be a curve on $\mathscr{S}$ that passes through $\mathscr{P}$. The vector at $\mathscr{P}$

$$d\mathbf{r} = \mathbf{r}_u du + \mathbf{r}_v dv = \begin{bmatrix} \mathbf{r}_u & \mathbf{r}_v \end{bmatrix} \begin{bmatrix} du \\ dv \end{bmatrix}. \tag{1}$$

is tangent to the curve $\alpha(t)$ and therefore to $\mathscr{S}$.

The infinitesimal squared length ds of an *element of arc* of $\alpha(t)$ in the vicinity of $\mathscr{P}$ can then be expressed in a quadratic or bilinear form, in terms of the differentials $\mathscr{U} = \begin{bmatrix} du & dv \end{bmatrix}^t$, which is known as the first fundamental form

$$\begin{aligned} ds = d\mathbf{r} \cdot d\mathbf{r} &= \Big[\begin{bmatrix} \mathbf{r}_u & \mathbf{r}_v \end{bmatrix} \begin{bmatrix} du \\ dv \end{bmatrix} \Big]^t \cdot \begin{bmatrix} \mathbf{r}_u & \mathbf{r}_v \end{bmatrix} \begin{bmatrix} du \\ dv \end{bmatrix} \\ &= \begin{bmatrix} du & dv \end{bmatrix} \begin{bmatrix} \mathbf{r}_u \cdot \mathbf{r}_u & \mathbf{r}_u \cdot \mathbf{r}_v \\ \mathbf{r}_v \cdot \mathbf{r}_u & \mathbf{r}_v \cdot \mathbf{r}_v \end{bmatrix} \begin{bmatrix} du \\ dv \end{bmatrix} \\ &= \begin{bmatrix} du & dv \end{bmatrix} \begin{bmatrix} E & F \\ F & G \end{bmatrix} \begin{bmatrix} du \\ dv \end{bmatrix} = \mathscr{U}^t \mathbb{I}_S \mathscr{U}. \end{aligned} \tag{2}$$

Observe that the inner product is symmetric at every point $F = \mathbf{r}_u \cdot \mathbf{r}_v = \mathbf{r}_v \cdot \mathbf{r}_u$.

According to Eq. 1, all vectors $d\mathbf{r}$ at $\mathscr{P}$ are linear combinations of $\mathbf{r}_u$ and $\mathbf{r}_v$, thus lying on the plane spanned by these two vectors. This plane is the *tangent plane* at $\mathscr{P}$ to $\mathscr{S}$. Its *unit normal vector* is given by

$$\mathbf{n} = \frac{\mathbf{r}_u \times \mathbf{r}_v}{\|\mathbf{r}_u \times \mathbf{r}_v\|} = \frac{\mathbf{r}_u \times \mathbf{r}_v}{\sqrt{EF - G^2}}. \tag{3}$$

The variations of the tangent vector of the curve $\alpha(t) = \mathbf{r}(u(t), v(t))$ at $\mathscr{P}$ define the curvature vector $\mathbf{k}$ of $\alpha(t)$. Thus, the curvature vector is an element of second derivative. Its projection over the normalized normal vector $\mathbf{n}$ of $\mathscr{S}$ at $\mathscr{P}$ is called the *normal curvature vector*

$$\mathbf{k}_n(\alpha(t)) = k_n(\alpha(t)) \cdot \mathbf{n}\,,$$

where $k_n(\alpha(t))$ is the *normal curvature* of $\alpha(t)$ at $\mathscr{P}$ and depends on the *shape operator* $-d_{\alpha(t)}\mathbf{n}$:

$$k_n(\alpha(t)) = -\frac{d\alpha(t) \cdot d_{\alpha(t)}\mathbf{n}}{d\alpha(t) \cdot d\alpha(t)} = -\frac{d\alpha(t) \cdot d_{\alpha(t)}\mathbf{n}}{\mathbb{I}_S}. \tag{4}$$

The normal curvature depends on the shape operator or the derivative of the normal $\mathbf{n}$ along the tangent direction of $\alpha(t) = (u(t), v(t))$ at $\mathscr{P}$,

$$d_{\alpha(t)}\mathbf{n} = \frac{\partial \mathbf{n}}{\partial u} du + \frac{\partial \mathbf{n}}{\partial v} dv. \tag{5}$$

Eq. 5 is called the *second fundamental form* and may also be written in the bilinear form

$$\begin{aligned} -d\alpha(t) \cdot d_{\alpha(t)}\mathbf{n} &= \Big[\begin{bmatrix} -\mathbf{r}_u & -\mathbf{r}_v \end{bmatrix} \begin{bmatrix} du \\ dv \end{bmatrix} \Big]^t \cdot \begin{bmatrix} \mathbf{n}_u & \mathbf{n}_v \end{bmatrix} \begin{bmatrix} du \\ dv \end{bmatrix} \\ &= \begin{bmatrix} du & dv \end{bmatrix} \begin{bmatrix} -\mathbf{r}_u \cdot \mathbf{n}_u & -\mathbf{r}_u \cdot \mathbf{n}_v \\ -\mathbf{r}_v \cdot \mathbf{n}_u & -\mathbf{r}_v \cdot \mathbf{n}_v \end{bmatrix} \begin{bmatrix} du \\ dv \end{bmatrix} \\ &= \begin{bmatrix} du & dv \end{bmatrix} \begin{bmatrix} e & f \\ f & g \end{bmatrix} \begin{bmatrix} du \\ dv \end{bmatrix} = \mathscr{U}^t \mathbb{II}_S \mathscr{U}. \end{aligned} \tag{6}$$

As the coefficients of $\mathbb{I}_S$ and $\mathbb{II}_S$ transform appropriately under allowable change of coordinates, they are called the *metric* and the *curvature tensor*, respectively.

For the basis $\{\mathbf{r}_u, \mathbf{r}_v\}$, the Weingarten equations express the derivatives of the normal to a regular surface in terms of this basis

$$\begin{bmatrix} \mathbf{n}_u \\ \mathbf{n}_v \end{bmatrix} = \begin{bmatrix} \frac{fF-eG}{EG-F^2} & \frac{eF-fE}{EG-F^2} \\ \frac{gF-fG}{EG-F^2} & \frac{fF-gE}{EG-F^2} \end{bmatrix} \begin{bmatrix} \mathbf{r}_u \\ \mathbf{r}_v \end{bmatrix} = \mathbb{W} \begin{bmatrix} \mathbf{r}_u \\ \mathbf{r}_v \end{bmatrix} \quad (7)$$

Although there is an infinity of curves on $\mathscr{S}$ passing through $\mathscr{P}$ and possessing the same tangent at this point, the Theorem of Meusnier tells us that all of these curves have the same normal curvature.

The direction of the normal curvature vector always remains that of the surface normal. Only its length may vary for different directions. The directions in which the normal curvature becomes extreme at $\mathscr{P}$ are called *principal directions*, $\{\mathbf{e}_1, \mathbf{e}_2\}$, and the corresponding normal curvatures are denominated *principal curvatures*, κ_{min} and κ_{max}. The principal directions and the principal curvatures are, respectively, the eigenvectors and the eigenvalues of $\mathbb{W}$. The product

$$K = \kappa_{min}\kappa_{max}. \quad (8)$$

of the two principal normal curvatures is called *Gaussian curvature* and their arithmetic mean

$$H = \frac{\kappa_{min} + \kappa_{max}}{2} \quad (9)$$

is called the *mean curvature*.

In some applications, the elements of third derivatives, namely the differentiations of $\mathbb{II}_S$, are of interest. It is because that they are helpful in quantifying the surface fairness (10). To define a differentiation in such a manner that the derivative of any tensor is again a tensor, the concept of *absolute differentiation* or *covariant derivatives* is introduced. Gravesen and Ungstrup show that the covariant derivatives of $\mathbb{II}_S$ is a trilinear symmetric tensor $2 \times 2 \times 2$ with coefficients c_{ijk} given by (10)

$$\begin{aligned} a &= c_{111} = \mathbf{r}_{uuu} \cdot \mathbf{n} + 3\mathbf{r}_{uu} \cdot \mathbf{n}_u \\ b &= c_{112} = c_{121} = c_{211} = \mathbf{r}_{uuv} \cdot \mathbf{n} + \mathbf{r}_{uu} \cdot \mathbf{n}_v + 2\mathbf{r}_{uv} \cdot \mathbf{n}_u \\ c &= c_{221} = c_{122} = c_{212} = \mathbf{r}_{uvv} \cdot \mathbf{n} + \mathbf{r}_{vv} \cdot \mathbf{n}_u + 2\mathbf{r}_{uv} \cdot \mathbf{n}_v \\ d &= c_{222} = \mathbf{r}_{vvv} \cdot \mathbf{n} + 3\mathbf{r}_{vv} \cdot \mathbf{n}_v \end{aligned} \quad (10)$$

These derivatives define the *tensor of curvature derivative* $\mathbb{C}_S$.

Using the principal directions $\{\mathbf{e}_1, \mathbf{e}_2\}$ as the local basis, these coefficients may be written as the directional derivatives of the principal curvature vectors $\mathbf{k}_{min} = \kappa_{min}\mathbf{n}$ and $\mathbf{k}_{max} = \kappa_{max}\mathbf{n}$ along the principal directions, that is

$$a = D_{\mathbf{e}_1}\mathbf{k}_{min} \quad b = D_{\mathbf{e}_2}\mathbf{k}_{min} \quad c = D_{\mathbf{e}_1}\mathbf{k}_{max} \quad d = D_{\mathbf{e}_2}\mathbf{k}_{max} \quad (11)$$

3 Related Work

Given a mesh of points, an accurate estimation of local differential geometric properties of the underlying surface for each sample vertex is, despite of a number of works, still a challenge issue.

Several techniques have been proposed for computing the elements of a tangent reference frame, which is an orthonormal basis composed of a normal vector at a surface point and two perpendicular vectors (the tangent and bitangent vectors). While most of them consider that the tangent and bitangent vectors are aligned with the texture coordinates (12), there is a variety of proposals for accurately estimating the normal vectors on the basis of the adjacent face normals (9; 5; 19; 14). Using the functionality of texture sampling on the vertex shader in Shader Model (SM) 3.0 (16), Calver proposes making available to the GPU vertex and adjacency data stored in textures. Based on this approach, the author devises an algorithm for computing face and vertex normals entirely on the GPU, after the vertex-level deformations (4).

To our knowledge, the geometric properties of second and third order, namely the curvatures and their derivatives, are still not robustly computable on GPUs due to its complexity. In this section, a brief description of three paradigms of techniques for estimation of geometric properties is provided: patch-fitting methods, averaging methods, and curve sampling methods.

3.1 Patch-fitting Techniques

One of the most traditional methods for estimating the differential properties from triangle meshes is by approximating a parametric surface to a local region of mesh points, then computing the matrix $\mathbb{W}$ from Eq. 7 analytically using the parametric derivatives of this surface (11). Mostly, this surface is represented in the tangent frame associated to the vertex of interest $\mathscr{P}$, so that we may consider that $\mathbb{W}$ is equal to $-\mathbb{II}_S$. For estimating properties of second order, the lowest order suitable is a quadratic surface (19).

Goldfeather and Interrante (8) find, however, that the robustness of the tensor $\mathbb{II}_S$ depends on how the local region around $\mathscr{P}$ is approximated by parabolas along the direction of the edges passing through the adjacent vertices. Approximation errors are greater in meshes sampled irregularly since in such cases the number of analytical surfaces that may fit in the same neighborhood of points is greater. To obtain better estimates of the principal directions, they propose higher order surfaces combined with the information of surface normals available at the adjacent vertices. The idea of exploiting the information of the normals at each adjacent vertex was followed by other techniques based on the 1-ring neighborhood beyond patch-fitting methods (18). It has shown to be especially important for obtaining robust results in irregularly sampled meshes.

According to Agam and Tang, the main drawback of this class of techniques is to be incapable of accurately modeling the local surface geometry and to introduce a strong element of smoothing t that may reduce the accuracy of the curvature estimation (1). Rusinkiewicz points out the ambiguity of the calculus of curvature at configurations of geometry with vertices coincident with two intersecting lines. In such regions, either a planar or hyperbolic surface fits the point set (18). We hypothesize that, if we model the vicinity of a point as a set of curves, we may gain flexibility in describing abrupt variations around a vertex.

3.2 Averaging Methods

This approach is based on the ability of estimating a curvature tensor for each edge and then averaging the tensors computed for the edges within a mesh region around each vertex. It considers that, for each edge E of a triangle mesh, there is an associated minimum curvature in the direction along the edge, and a maximum curvature in the perpendicular direction that crosses the edge on the tangent plane. Such condition allow us to define a curvature tensor $\mathbb{III}_S$ for any point on the edge, and the average of the tensors computed for the edges inside a given region around any vertex $\mathscr{P}$. Because the point has its own tangent frame, the coordinate transformation law must be applied on $\mathbb{III}_S$ of each edge before averaging.

The techniques of approximation by curvature tensor averaging do not suffer from the deficiencies of the patch-fitting approach at hyperbolic points. However, other configurations may accumulate errors that are not minimized even when increasing the number of edges used in the evaluation of the average. One of these configurations is found at the poles of a geodesic sphere build up from the subdivision of a tetrahedron (18). At these pole points, the estimation error does not decrease as the mesh refinement increases.

Trying to solve the deficiencies of curvature tensor averaging techniques (6; 2), Rusinkiewicz proposes a new technique of approximation (18) based on the idea of Goldfeather and Interrante (8). He uses the already available vertex normals at the vertices in the 1-ring neighborhood. For computing weights, he found that the "Voronoi area" weighting presented by Meyer et al. (15) produces the best estimates of curvature for triangles of varying size and shape. Similar to the traditional algorithms for computing vertex normals, the tensor $\mathbb{III}_S$ is first computed for the faces and then estimated for the vertices through averaging of the tensors of the adjacent faces. Hence, two passes are required. Moreover, the computation of the Voronoi area weighting is somehow cumbersome. Our conjecture is that, by applying the curve sampling approach we may simplify the procedure without degrading the accuracy of results.

3.3 Curve Sampling Techniques

Instead of fitting a patch to points in a local neighborhood of the point of interest $\mathscr{P}$, a set of curves leaving $\mathscr{P}$ in distinct directions is sampled for curvature estimation.

For discrete surfaces, the estimative of the normal curvature in the direction of an edge between a vertex $\mathscr{P}$ to q can be obtained by the formula of the curvature of the osculating circle that spans these vertices (5; 15):

$$\kappa_{\mathscr{P}q} = 2\frac{(\mathscr{P}-q)\cdot \mathbf{n}_{\mathscr{P}}}{\|\mathscr{P}-q\|^2} \tag{12}$$

where $\mathbf{n}_{\mathscr{P}}$ is the surface normal at $\mathscr{P}$. By combining equations 4, 7 and 12, we obtain a linear equation system. This system can be solved for $\mathbb{III}_S$ through linear least squares (5). When comparing this set of equations with those obtained with the quadratic surface approximation, we observe that they differ only with respect to the type of curves approximated in the direction of each edge: parabolas, for the approximation by quadratic surfaces; circles, for the approximation by normal curvature. Therefore, the estimative by normal curvature suffers from the same weaknesses of the approximation by analytical surfaces.

To overcome this restriction, Agam and Tang propose to fit the local neighborhood of the vertex $\mathscr{P}$ with a set of quadratic curves, then calculate the normal curvature of each curve separately (1). The directions along the minimum and maximum curvatures are chosen as principal directions. The estimation accuracy relies strongly on the sampling coverage and on the estimation accuracy of the normal curvatures.

To be less sensitive to the sampling frequency, we follow Goldfeather and Interrante and propose to overcome the mentioned restriction by substituting for Eq. 12 the expression of directional derivative of normal curvature given in (18), which also considers the information of the vertex normals at each adjacent vertex.

4 Our Proposal

We propose a new method for computing second and higher order elements of discrete differential geometry of arbitrary meshes. The approach is capable of computing, for each vertex, the curvature tensor $\mathbb{III}_S$ and therefore the principal curvatures and directions, but also the coefficients of the tensor of curvature derivative $\mathbb{C}_S$. Our proposal focuses on the following requisites:

– **Efficiency**: The total amount of time for estimating the curvature tensor, principal curvatures and directions, and tensor of curvature derivative should allow the algorithm to be used in real-time for models composed of thousands of vertices on current desktop hardware. In order to do that, we consider the possibility of using the bandwidth and stream processing power of today's GPUs for performing the estimation. The instruction set and data structures used by the algorithm should be simple enough to allow such portability.

– **Coverage**: The algorithm should avoid restrictions with respect to the topology of the model. In particular, the algorithm should be capable of dealing with surfaces with holes, meshes with irregular sampling and different vertex valences. It also should handle the lack of data about face orientation, so that the only required connectivity data should be the 1-ring neighborhood of each vertex, possibly without a winding order.
– **Robustness**: The algorithm should satisfactorily estimate differential geometry properties for the configurations considered problematic for the methods cited before: collinear points and points of intersection between two lines. In digitized models, the presence of outliers should be minimized in comparison with the cited methods.

None of the techniques presented fulfill all the requisites exposed above. Though the techniques based on the patch-fitting and curve sampling approaches are suitable for stream processor architectures and, therefore, are very efficient, they do not produce results robust enough for arbitrary meshes. On the other hand, the techniques based on curvature tensor averaging are robust, but cumbersome to implement on a stream processor due to the need of complex data structures or multiple execution passes. For instance, the technique by Alliez et al. (2) requires a costly pre-processing for determining which edges (and in which amount) are inside a region around each vertex. Likewise, in the algorithm proposed by Rusinkiewicz (18), the averaging between the curvature tensors at the faces requires at least two processing passes, which hinders its use on the GPU.

In order to satisfy the requisites above, our proposal tries to combine the robustness of the algorithms based on the averaging method, and the simplicity and efficiency of the methods based on the curve sampling technique. It stands out as an adaptation of the algorithm by Rusinkiewicz, both for computing the curvature tensor as well as the tensor of curvature derivative.

4.1 Estimating the curvature tensor

For each vertex $\mathscr{P}$, we estimate $\mathbb{II}_S$ in the tangent reference frame $\{\mathbf{U},\mathbf{V},\mathbf{n}\}$, so that we may assume that $\mathbb{II}_S = -\mathbb{W}_S$. Replacing it in Eq. 7, the derivative of the normal vector $\mathbf{n}$ in a given direction $\mathbf{x}$ on the tangent plane passing through $\mathscr{P}$ is expressed by

$$\mathbb{II}_S\mathbf{x} = D_{\mathbf{x}}\mathbf{n} \tag{13}$$

On a smooth surface, the vector $\mathbf{x}$ may assume any direction perpendicular to the normal vector $\mathbf{n}$. On a mesh with 1-ring neighborhood, these directions can be approximated by vectors from $\mathscr{P}$ to each vertex q_i of the 1-ring neighborhood. Similarly, the directional derivatives of the normal vectors can be approximated by the difference of the normal vectors previously estimated at the vertices. We remark that both the differences of positions and normals should be expressed in the reference frame $\{\mathbf{U},\mathbf{V},\mathbf{n}\}$. Hence, Eq. 13 may be written in the following finite-difference form

$$\mathbb{II}_S \begin{bmatrix} (q_i-\mathscr{P})\cdot\mathbf{U} \\ (q_i-\mathscr{P}\cdot\mathbf{V} \end{bmatrix} = \begin{bmatrix} (\mathbf{n}_{q_i}-\mathbf{n}_{\mathscr{P}})\cdot\mathbf{U} \\ (\mathbf{n}_{q_i}-\mathbf{n}_{\mathscr{P}})\cdot\mathbf{V} \end{bmatrix}$$

which may be expanded into

$$\begin{bmatrix} (q_i-\mathscr{P})\cdot\mathbf{U} & (q_i-\mathscr{P})\cdot\mathbf{V} & 0 \\ 0 & (q_i-\mathscr{P})\cdot\mathbf{U} & (q_i-\mathscr{P})\cdot\mathbf{V} \end{bmatrix} \begin{bmatrix} e \\ f \\ g \end{bmatrix} = \\ = \begin{bmatrix} (\mathbf{n}_{q_i}-\mathbf{n}_{\mathscr{P}})\cdot\mathbf{U} \\ (\mathbf{n}_{q_i}-\mathbf{n}_{\mathscr{P}})\cdot\mathbf{V} \end{bmatrix}, \tag{14}$$

where q_i, $i = 1,2,\ldots,n$, are n vertices in the 1-ring neighborhood of $\mathscr{P}$. This system is solved for $\mathbb{II}_S$ using a linear least squares algorithm. As in the implementation by (18), we use the LDT^t decomposition method.

Differently from Rusinkiewicz, our proposal performs the calculation directly with respect to the vertices, in a single running loop, because it does not require additional computations for transforming the tensor $\mathbb{II}_S$ computed on the faces to the tangent plane at the vertices.

4.2 Estimating the tensor of curvature derivative

To estimate the tensor of curvature derivative, we resort to Eq. 11 which gives us a simple way to obtain $\mathbb{C}_S$ in coordinates relative to the principal directions. In order to do so, we first compute $\mathbb{II}_S$ and then extract its eigenvalues (principal curvatures) and eigenvectors (principal directions). The principal directions become the new basis vectors $\{\mathbf{e}_1,\mathbf{e}_2\}$ on the tangent plane at $\mathscr{P}$. Now we can express each direction $\mathbf{E}_i = \overrightarrow{\mathscr{P}q_i}$ in these new coordinates. We then obtain the following system of equations which expresses, by finite differences, the minimal and maximal curvature variations along distinct directions $\mathbf{E}_i$ with regard to the principal directions:

$$\mathbb{C}_S \begin{bmatrix} \mathbf{E}_i\cdot\mathbf{e}_1 \\ \mathbf{E}_i\cdot\mathbf{e}_2 \end{bmatrix} = \begin{bmatrix} \Delta\kappa_{min_i} & \Delta\kappa_{max_i} \\ \Delta\kappa_{min_i} & \Delta\kappa_{max_i} \end{bmatrix}, i = 1,2,\ldots,n, \tag{15}$$

where $\Delta\kappa_{min_i}$ and $\Delta\kappa_{max_i}$ are, respectively, the difference between the minimal and maximal normal curvatures at vertices q_i and $\mathscr{P}$, that is,

$$\Delta\kappa_{min_i} = \kappa_{min_{q_i}} - \kappa_{min_{\mathscr{P}}} \qquad \Delta\kappa_{max_i} = \kappa_{max_{q_i}} - \kappa_{max_{\mathscr{P}}}$$

In an analogous way that we obtain the finite-difference form for computing $\mathbb{II}_S$, we may expand Eq. 15 to explicit the coefficients a, b, c and d of the tensor $\mathbb{C}_S$

$$\begin{bmatrix} \mathbf{E}_i\cdot\mathbf{e}_1 & \mathbf{E}_i\cdot\mathbf{e}_2 & 0 & 0 \\ 0 & \mathbf{E}_i\cdot\mathbf{e}_1 & \mathbf{E}_i\cdot\mathbf{e}_2 & 0 \\ 0 & \mathbf{E}_i\cdot\mathbf{e}_1 & \mathbf{E}_i\cdot\mathbf{e}_2 & 0 \\ 0 & 0 & \mathbf{E}_i\cdot\mathbf{e}_1 & \mathbf{E}_i\cdot\mathbf{e}_2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} \Delta\kappa_{min_i} \\ \Delta\kappa_{min_i} \\ \Delta\kappa_{max_i} \\ \Delta\kappa_{max_i} \end{bmatrix}, \tag{16}$$

and solve it through a linear least squares technique, again using LDL^t decomposition. The computation of the eigenvectors and eigenvalues is done by Jacobi iteration.

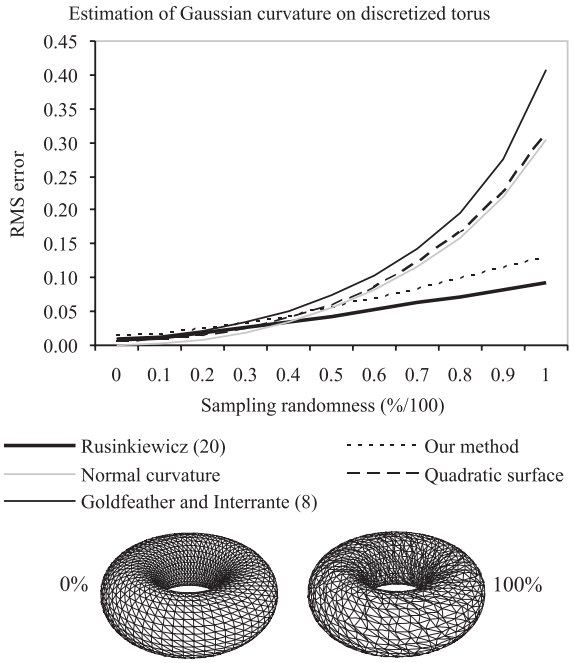


Fig. 2 RMS error of Gaussian curvature estimation on a discretized torus as a function of increasing sampling randomness. Bottom: Wireframe torus with minimum sampling randomness (left) and maximum randomness (right).

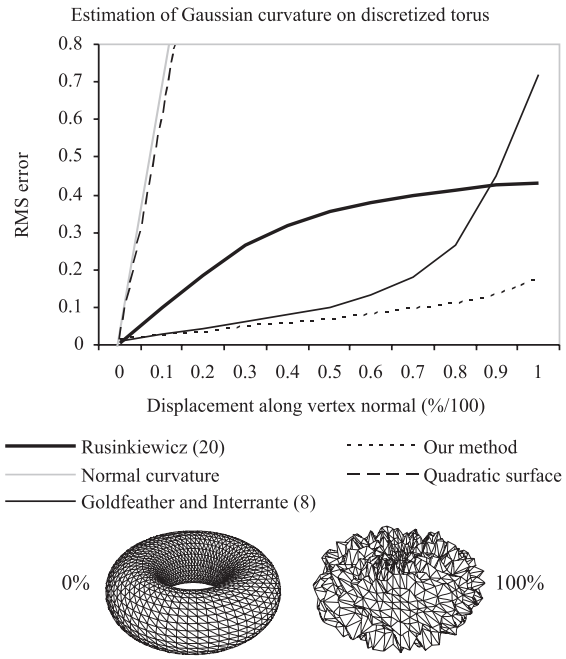


Fig. 3 RMS error of Gaussian curvature estimation on a discretized torus as a function of increasing vertex displacement along the exact vertex normal. Bottom: Wireframe torus with minimum (left) and maximum (right) displacement.

5 Experimental Results and Discussions

We have implemented our algorithm both on the CPU and GPU. On the CPU, the algorithm was implemented in C++ using the LAPACK library (3). Methods based solely on the 1-ring neighborhood of vertices, such as the approximation by quadratic surface, cubic surface (8), normal curvature and tensor averaging (18), were also implemented with this library in order to perform a fair comparison with our algorithm when running the robustness tests.

From our experiments, we observed that the technique proposed by Agam and Tang, though delivers good results, is sensitive to the sampling coverage and frequency. The nonexistence of a robust algorithm for adaptively choosing appropriate sampling coverage makes it impractical in some situations. Hence, we have decided to limit the comparisons of our model with the algorithms presented by Rusinkiewicz, Goldfeather and Interrante.

Figure 2 shows a comparison, for different techniques, of the RMS error for estimating the Gaussian curvature (Eq. 8) of a discretized torus as the surface sampling becomes more irregular. Two wireframe renderings of the torus show the minimum and maximum mesh sampling irregularity used. The normals were computed as the normalized sum of the normal vectors at the faces (9). These results show that the algorithm by Rusinkiewicz has the smaller proportional increase in the estimation error due to sampling irregularity. Our algorithm produces similar results, though slightly less accurate. The remaining techniques are more accurate only when the torus is regularly sampled, as they assume the surface is locally polynomial, a condition fully satisfied by the torus. Nevertheless, they are more sensitive to sampling noise even when considering polynomial surfaces.

Figure 3 compares the RMS error for estimating the Gaussian curvature of the discretized torus as noise is added to the vertices. The vertices are displaced along fixed normal vectors computed analytically. Our technique produces the most accurate results in this case. In contrast, the techniques of approximation by quadratic surfaces and normal curvature produce gross estimation errors, as they do not use the normals to counterbalance the displacement noise.

Figure 4 shows the color-coded visualization of the principal curvatures for a horse model (48,484 vertices, 96,964 triangles) and a bunny model (72,027 vertices, 144,460 triangles). The colors are modulated with the model's diffuse shading. The amount of outliers, which come out as small isolated colored dots, is relatively low in our technique and visually equivalent to the results of Rusinkiewicz. The remaining techniques produce more outliers as a result of the existence of local configurations considered problematic for these approaches. The results of the estimation by curve sampling of normal curvature are not shown, as they are visually identical to the quadratic approximation.

For the timing tests, we have compared our technique with that of Rusinkiewicz (18) due to the similarity of robustness results with those of our algorithm. In that case, both algorithms were implemented using optimized code. The test platform was a AMD Athlon XP 2000+ 1.67GHz

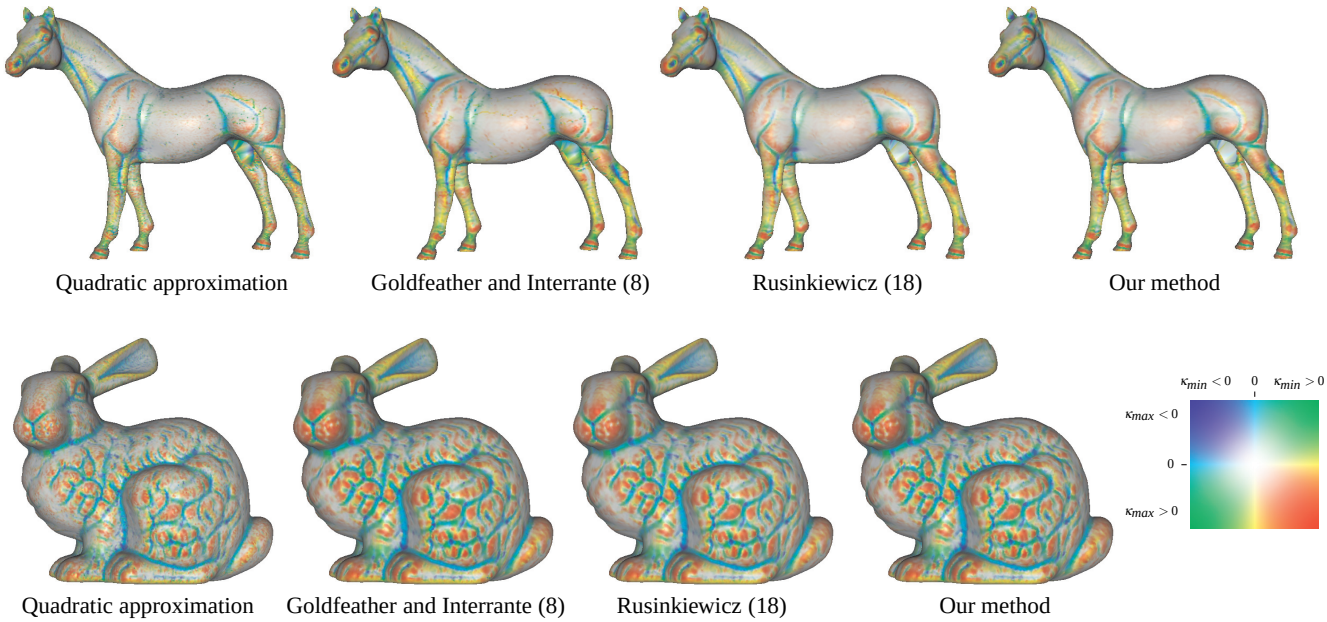


Fig. 4 Color-coded principal curvatures for a horse and bunny model according to different estimation approaches.

with 512MB RAM, and a low-end GPU NVIDIA GeForce FX 6200 AGP 8x with 256MB VRAM.

On the GPU, the algorithm has been implemented as two SM 3.0 pixel shaders in HLSL (16), one for estimating the tensor of curvature and another one for the tensor of curvature derivative. The implementation of the linear least squares algorithm and Jacobi iteration on the GPU is straightforward, since the operations involved are conventional arithmetic ones that are supported by current graphics hardware. Following the paradigm of using the GPU as a general-purpose stream processor, geometry and connectivity data are encoded previously in floating point textures. These textures are accessed on the pixel shader that outputs the computed values to render textures which are then used by the application.

Figure 5 shows the performance for computing the curvature tensor (including the computation of principal curvatures and directions) and tensor of curvature derivative in comparison to (18). The processing time for our algorithm on the GPU is shown as well and includes the time for transferring the textures to and from the GPU. From these results we see that our method on the CPU is about two times more efficient for the tested model. This is mainly because the algorithm performs all the computations in a single pass. This simplicity, which allows us to port the algorithm to the GPU, improves the efficiency to real-time rates even in low-end graphics hardware. In general, we have seen that our algorithm has a significant increase in performance with respect to the technique of Rusinkiewicz, at a minor cost of decreasing the accuracy on irregularly sampled meshes. For

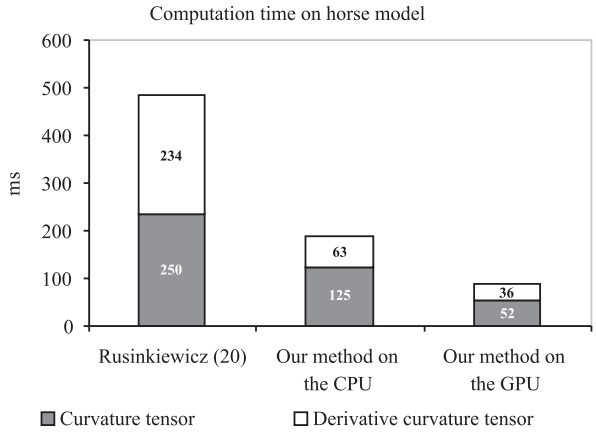


Fig. 5 Performance for estimating the curvature tensor and tensor of derivative curvature on the horse model, in milliseconds.

digitized models, the decrease of accuracy is hardly noticeable.

Figure 6 shows some screenshots of a demonstration that we have implemented to visualize the estimated differential geometry elements computed in real-time on the GPU. In (a) we show a knot model with principal curvatures visualized as colors using the color code shown in Figure 4. In (b) we show the Gaussian curvature of a teapot model in grayscale. Darker and lighter tones correspond to negative and positive curvatures, respectively. Middle gray corresponds to zero. Notice that the surface is locally hyperbolic when $K < 0$ (darker), parabolic or planar when $K = 0$ (mid-gray), and elliptical when $K > 0$ (lighter). In (c) we

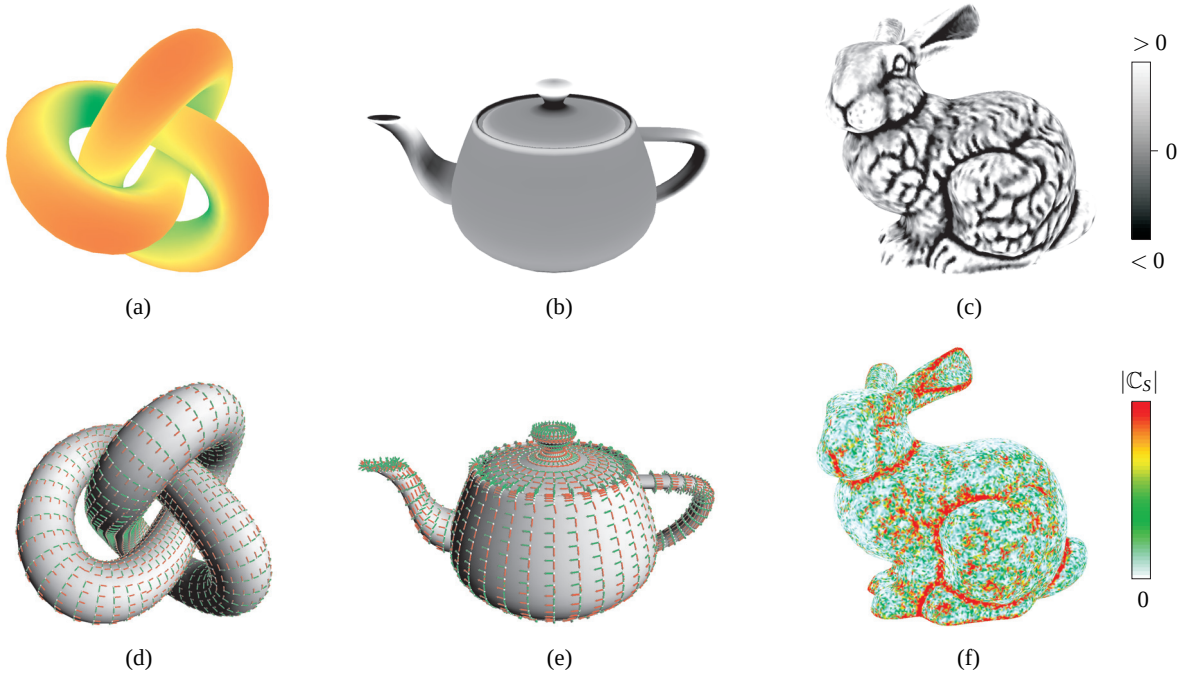


Fig. 6 Visualization of differential geometry properties estimated by our algorithm on the GPU. (a) principal curvatures using the color code shown in Figure 4; (b) Gaussian curvature, in grayscale; (c) mean curvature, in grayscale; (d, e) principal directions; (f) magnitude of the tensor of curvature derivative.

show the mean curvature (Eq. 9) of a bunny model, also in grayscale. The principal directions of a teapot and knot model are shown in (d) and (e), respectively. The red arrow points to the tangent direction of the minimum curvature, while the green arrow points to the tangent direction of the maximum curvature. In (f) we show the color-coded visualization of the magnitude of the tensor of curvature derivative ($|\mathbb{C}_S| = c_1^2 + c_2^2 + c_3^2 + c_4^2$, where c_1, c_2, c_3 and c_4 are the non-symmetrical components of the tensor). Images with higher resolution and source code of the demonstration are available at `http://www.dca.fee.unicamp.br/projects/mtk/batagelo`.

6 Conclusions

We have presented a curve sampling algorithm for estimating second or higher order local elements of discrete differential geometry for arbitrary meshes. The algorithm is simple to implement, as it requires only the 1-ring adjacency information of each point (not necessarily ordered) and the associated normal vectors to increase accuracy in comparison to traditional patch-fitting and curve sampling methods. On a single loop over the vertices, it performs the estimation by sampling the curves in the direction of each edge leaving each vertex, then solving a system of linear equations that relate the directional derivative of the normal curvatures to the tensor.

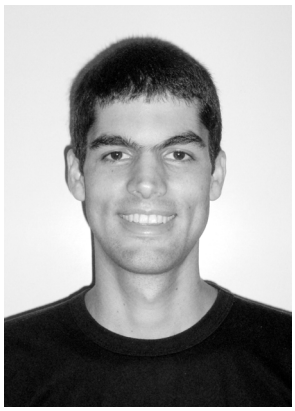
The algorithm is efficient in comparison to methods with similar accuracy, and is suitable for porting to the stream architecture of current GPUs. As shown in our timing tests, this allows the real-time computation of curvature tensors (including the extraction of principal directions and principal curvatures) and tensors of derivative of curvature even for models composed of thousands of vertices. As a further work, it motivates us to develop a 3D interaction toolkit that use the differential geometry properties to correctly perform direct manipulation with geometry deformed on the GPU.

According to the robustness tests, the accuracy of our method is greater than previous methods based on 1-ring neighborhood of vertices when considering irregularly sampled meshes and digitized models. In comparison to the techniques implemented, it produces the most accurate results on models with displacement noise. For irregular meshes, it produces slightly inferior results to the tensor averaging method proposed by Rusinkiewicz. However, the differences are hardly distinguishable, and the benefits of efficiency (almost half the time of (18)) and portability (adaptation to stream architectures) greatly overcome this. Nevertheless, we would like to further compare our results with the ones computed analytically, especially with the values of derivative of curvature.

Acknowledgements This project was supported by the National Research Council (CNPq) under the grant number 141685/2002-6 and the State of São Paulo Research Support Foundation (FAPESP) under the grant numbers 1996/0962-0 and 03/13090-6.

References

1. Agam, G., Tang, X.: A sampling framework for accurate curvature estimation in discrete surfaces. IEEE Transactions on Visualization and Computer Graphics **11**(5), 573–583 (2005)
2. Alliez, P., Cohen-Steiner, D., Devillers, O., Lévy, B., Desbrun, M.: Anisotropic polygonal remeshing. ACM Transactions on Graphics **22**(3), 485–493 (2003)
3. Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.: LAPACK Users' Guide, 3rd edn. Society for Industrial and Applied Mathematics, Philadelphia, PA (1999)
4. Calver, D.: Accessing and modifying topology on the gpu. In: W. Engel (ed.) ShaderX3: Advanced Rendering with DirectX and OpenGL. Charles River Media (2004)
5. Chen, X., Schmitt, F.: Intrinsic surface properties from surface triangulation. In: ECCV '92: Proceedings of the Second European Conference on Computer Vision, pp. 739–743. Springer-Verlag, London, UK (1992)
6. Cohen-Steiner, D., Morvan, J.M.: Restricted delaunay triangulations and normal cycle. In: SCG '03: Proceedings of the Nineteenth Annual Symposium on Computational Geometry, pp. 312–321. ACM Press, New York, NY, USA (2003)
7. DeCarlo, D., Finkelstein, A., Rusinkiewicz, S.: Interactive rendering of suggestive contours with temporal coherence. In: NPAR '04: Proceedings of the 3rd International Symposium on Non-photorealistic Animation and Rendering, pp. 15–145. ACM Press, New York, NY, USA (2004)
8. Goldfeather, J., Interrante, V.: A novel cubic-order algorithm for approximating principal direction vectors. ACM Transactions on Graphics **23**(1), 45–63 (2004)
9. Gouraud, H.: Continuous shading of curved surfaces. IEEE Transactions on Computers **20**(6), 623–629 (1971)
10. Gravesen, J., Ungstrup, M.: Constructing invariant fairness measures for surfaces. Advances in Computational Mathematics **17**, 67–88 (2002)
11. Hamann, B.: Curvature approximation for triangulated surfaces. Computing Suppl. **8**, 139–153 (1993)
12. Kilgard, M.J.: A practical and robust bump-mapping technique for today's gpus. In: Game Developers Conference 2000: Advanced OpenGL Game Development. NVIDIA Corporation (2000)
13. Lange, C., Polthier, K.: Anisotropic smoothing of point sets. Comput. Aided Geom. Des. **22**(7), 680–692 (2005)
14. Max, N.: Weights for computing vertex normals from facet normals. Journal of Graphics Tools **4**(2), 1–6 (1999)
15. Meyer, M., Desbrun, M., Schröder, P., Barr, A.H.: Discrete differential-geometry operators for triangulated 2-manifolds. In: H.C. Hege, K. Polthier (eds.) Proceedings of Visualization and Mathematics III, pp. 35–57. Springer-Verlag, Heidelberg, Germany (2003)
16. Microsoft: DirectX SDK Programmer's Reference. Microsoft Corporation, Redmond, WA, USA (2006)
17. Praun, E., Hoppe, H., Webb, M., Finkelstein, A.: Real-time hatching. In: SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, p. 581. ACM Press, New York, NY, USA (2001)
18. Rusinkiewicz, S.: Estimating curvatures and their derivatives on triangle meshes. In: 3DPVT '04: Proceedings of the 3D Data Processing, Visualization, and Transmission, 2nd International Symposium on (3DPVT'04), pp. 486–493. IEEE Computer Society, Washington, DC, USA (2004)
19. Taubin, G.: Estimating the tensor of curvature of a surface from a polyhedral approximation. In: ICCV '95: Proceedings of the Fifth International Conference on Computer Vision, p. 902. IEEE Computer Society, Washington, DC, USA (1995)
20. Wang, L., Wang, X., Tong, X., Lin, S., Hu, S., Guo, B., Shum, H.Y.: View-dependent displacement mapping. ACM Transactions on Graphics **22**(3), 334–339 (2003)
21. Yoo, K.H., Ha, J.S.: Geometric snapping for 3d meshes. In: International Conference on Computational Science, pp. 90–97. Krakow, Poland (2004)



Harlen Costa Batagelo is a Ph.D. student in Electrical Engineering at the School of Electrical and Computer Engineering of the State University of Campinas, Brazil. He received his Master's Degree in 2002 from the same institution, and his Bachelor's in Computer Science from the University of the West of Santa Catarina, in 1999. His main research interests are real-time rendering, computer games, graphics hardware and interaction.

Wu, Shin - Ting is an associate professor of School of Electrical and Computer Engineering (FEEC), State University of Campinas in Brazil. She received Dr.-Ing. in Informatics from the Technical University of Darmstadt, Master in Electrical Engineering from State University of Campinas, and Bachelor in Electrical Engineering from Federal University of Minas Gerais, in 1991, 1985, and 1981, respectively. Before she joined the FEEC, she worked as researcher at Renato Archer Research Center (CenPRA) and did her post-doc fellow in Institute for Computer Graphics of Fraunhofer Gesellschaft. Her research interests include geometric modeling, scientific visualization, and 3D interactions.