Silhouette Smoothing for Real-Time Rendering of Mesh Surfaces

Lu Wang, Changhe Tu, Wenping Wang, Xiangxu Meng, Bin Chan, and Dongming Yan

Abstract—Coarse piecewise linear approximation of surfaces causes the undesirable polygonal appearance of silhouettes. We present an efficient method for smoothing the silhouettes of coarse triangle meshes using efficient 3D curve reconstruction and simple local remeshing. It does not assume the availability of a fine mesh and generates only a moderate amount of additional data at runtime. Furthermore, polygonal feature edges are also smoothed in a unified framework. Our method is based on a novel interpolation scheme over silhouette triangles, and this ensures that smooth silhouettes are faithfully reconstructed and always change continuously with respect to the continuous movement of the viewpoint or objects. We speed up computation with GPU assistance to achieve real-time rendering of coarse meshes with the smoothed silhouettes. Experiments show that this method outperforms previous methods for silhouette smoothing.

Index Terms—Silhouette smoothing, polygonal mesh, interpolation, Hermite curves.

1 Introduction

Polygonal meshes are widely used for representing 3D shapes in computer graphics. Simplification methods produce meshes of small triangle count that are needed in many applications where there is considerable resource limitation, such as 3D real-time graphics on mobile devices. The resulting coarse meshes have conspicuous polygonal silhouettes or polygonal feature edges, causing impression of low visual quality, since human vision is particularly acute to silhouettes or *feature curves* and their nonsmoothness. (A *feature curve* is a smooth curve defined by transversal intersection, that is, nontangent intersection, of two smooth surfaces. See Fig. 2.)

The challenge of rendering a coarse mesh with smoothed silhouettes is well recognized by the computer graphics community [5], [11]. A small triangle count, which is required by efficient storage and transmission, and a faithful smooth appearance are two conflicting requirements. Simply refining a coarse mesh overall to increase the number of triangles would be inefficient for the purpose of silhouette smoothing, since much extra resources would then be wasted in creating, storing, and rendering an increased number of triangles in the interior (that is, nonsilhouette region) of a mesh.

1.1 Overview and Contributions

We present an efficient method that fixes polygonal silhouettes and feature edges into smooth curves for real-time rendering of coarse meshes (Fig. 1). The method requires only a coarse triangle mesh as input. It performs view-dependent 3D curve reconstruction and simple local remeshing to generate smooth silhouettes at runtime; feature edges are smoothed in preprocessing. Unlike previous approaches, our method does not require an LOD model or a fine mesh and avoids global smooth surface reconstruction. The local remeshing is based on the notion of *silhouette triangles*, which ensures that smooth silhouettes are faithfully reconstructed and coherent (that is, free of visual discontinuity) when rendered with respect to a moving viewpoint or with continuous object movement. Note that our method only focuses on the smoothing of silhouette and features edges, and it does not attempt to alter the appearance (for example, shading and texture) of a coarse mesh in regions away from its silhouettes or feature curves.

The major contribution of this paper is a new method for computing smooth silhouettes of coarse triangle mesh surfaces. The smooth silhouette curves are computed using an interpolation scheme over *silhouette triangles*, a concept that we will introduce later. As a result, the silhouette curves thus computed are more accurate than those by previous methods and possess visual coherence for moving objects or with respect to a moving viewpoint.

Some other advantages of our method are given as follows: Polygonal feature edges are smoothed in the same way as polygonal silhouette edges. We will (Section 3.4) see that this not only makes feature edges look smooth, but also ensures that these smooth feature curves possess visual coherence during continuous motion. We also apply an effective vertex perturbation scheme to saddle regions to prevent the smoothed silhouette from being blocked by the neighboring faces of the original mesh, which is a critical issue that has not been addressed by previous silhouette smoothing methods. The method runs efficiently with GPU assistance.

After reviewing related works in the rest of this section, we will present an outline of our method in Section 2. Curve approximation to silhouettes and feature curves and the

- *L. Wang, C. Tu, and X. Meng are with the School of Computer Science and Technology, Shandong University, Jinan 250100, P.R. China. E-mail: luwang@mail.sdu.edu.cn, {chtu, mxx}@sdu.edu.cn.*
- *W. Wang, B. Chan, and D. Yan are with the Department of Computer Science, University of Hong Kong, Hong Kong. E-mail: {wenping, bchan, dmyan}@cs.hku.hk.*

Manuscript received 11 July 2007; revised 29 Oct. 2007; accepted 26 Nov. 2007; published online 2 Jan. 2008.
Recommended for acceptance by P. Slusallek.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-2007-07-0082.
Digital Object Identifier no. 10.1109/TVCG.2008.8.

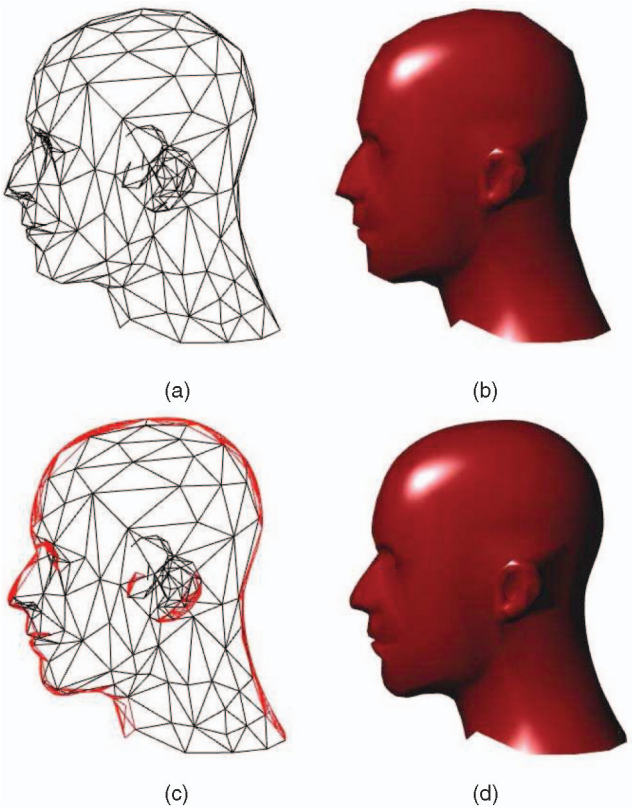


Fig. 1. An example of silhouette smoothing by our proposed method. (a) A coarse mesh model (500 triangles). (b) The shaded coarse mesh. (c) The coarse meshed is refined near silhouettes by our method (904 triangles). (d) Phong shading of the mesh in (c), with smoothed silhouettes.

remeshing scheme are described in detail in Section 3. GPU speedup is discussed in Section 4. We present experimental results in Section 5 and conclude the paper in Section 6.

1.2 Related Work

The silhouette rendering problem has extensively been studied for various geometric models (for example, [4] and [22]). Here, we focus on polygonal meshes. Many methods for real-time mesh rendering require a fine mesh to start with. For better rendering efficiency, a progressively simplified mesh is usually displayed in a view-dependent manner to ensure the smooth appearance of silhouettes. These methods include techniques based on multiple (or even continuous)

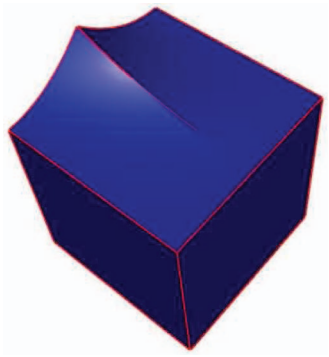


Fig. 2. Feature curves (in red) are local traversal intersections of two surfaces.

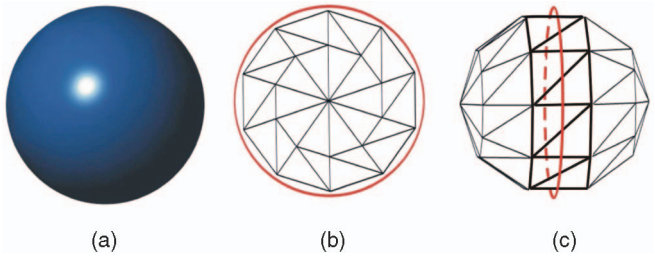


Fig. 3. (a) A sphere. (b) A coarse mesh sampled from the sphere and the silhouette curve (in red). (c) The coarse mesh and silhouette curve in (b) viewed from another angle.

levels of detail [21], [13], [8], [9], [6], [15], [1]. The silhouette clipping technique [17] also needs a fine mesh for assistance. However, in many applications, only relatively coarse meshes are available; very fine meshes cannot be used, either because they are not available or because of limited bandwidth or memory.

The PN-triangle method [19] and its variant [3] do not assume a fine mesh as input; rather, they construct a smooth cubic surface patch for each triangle of a coarse triangle mesh. These methods use global surface patch reconstruction to achieve the overall smoothness of a coarse mesh for rendering, thus improving the silhouette smoothness, as well as the shading quality of the interior region. Normally, the level of subdivision of cubic patches is uniform across the model and fixed before rendering, in a viewpoint-independent manner. Therefore, the polygonal appearance (that is, the nonsmoothness) of silhouettes still becomes apparent when one zooms in on a silhouette. (See the comparison of our method with the PN-triangle method in Fig. 24, Section 5.)

For fast silhouette smoothing and rendering, the method in [20] takes a local and view-dependent approach that does not assume the availability of fine meshes and does not involve global geometry reconstruction. It performs local 2D curve approximation to the projected boundary of a mesh surface on the view plane. This boundary curve corresponds to the *silhouette edges* of the mesh; an edge of a mesh surface is called a *silhouette edge* if one of its two incident triangles is visible and the other is invisible, that is, a back-facing triangle.

Many existing methods extract silhouette edges from polygonal meshes [2], [10], [14], [16]. However, there are some fundamental difficulties in computing smooth silhouettes from silhouette edges of a mesh model. Consider the mesh approximating a sphere in Fig. 3. Suppose that all the mesh vertices are on the sphere. Fig. 3b shows that in one view, the silhouette curve (that is, the circle in red) does not pass through the vertices of the silhouette polygon. The side view in Fig. 3c shows that the silhouette curve in Fig. 3b passes over a series of mesh triangles, which are marked in thick lines, rather than corresponds to any edges of the mesh.

This observation suggests that it is inherently inaccurate and essentially incorrect to use silhouette edges for reconstructing smooth silhouette curves, because a smooth silhouette curve does not necessarily correspond to silhouette edges. Furthermore, besides the accuracy consideration, the smooth silhouette curves thus computed are not visually coherent with respect to a moving viewpoint or if

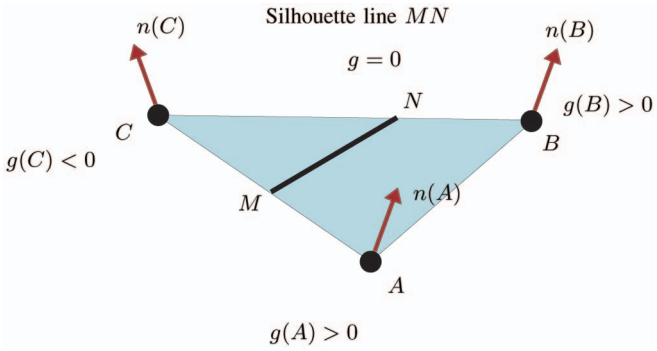


Fig. 4. Silhouette line MN as defined in [7].

the mesh model moves continuously, since they depend on the silhouette edges that appear or disappear abruptly for a rotating mesh model.

Hertzmann and Zorin in [7] noticed the visual coherence issue and proposed the following method for extracting silhouette curves. Suppose that a smooth surface is represented by a triangular mesh. Let E be the viewpoint. Let $n(p)$ be the estimated unit normal vector at a mesh vertex p. Then, consider the function $g(p) = n(p) \cdot (p - E)$ defined at all mesh vertices p. Now, extend the domain of $g(p)$ to be over each triangle by the linear interpolation of its values at the three vertices of the triangle. Then, the silhouette curve is defined to consist of all those points p on the mesh surface satisfying $g(p) = 0$ (for example, the line segment MN in Fig. 4). The silhouette thus computed possesses visual coherence since it is dependent on the viewpoint E continuously. However, the smoothness issue is still not addressed, as the silhouette curve thus constructed is a polyline lying on the faces of the mesh.

In the present paper, we focus on local silhouette processing, without using a fine mesh or global surface reconstruction. Rather than using the silhouette edges of a mesh as in [2], [4], [10], [16], and [20], we compute silhouette curves based on *silhouette triangles*. The idea here of associating a silhouette segment to a triangle bears similarity to the treatment by Hertzmann and Zorin in [7]. We aim at faithful smooth silhouette reconstruction and visual coherence of the silhouettes of a moving mesh model to support the real-time rendering of a coarse mesh with smooth silhouettes.

2 Outline of the Proposed Method

We consider triangle meshes that approximate a piecewise smooth surface. There are two types of edges in such mesh surfaces: *feature edges* and *nonfeature edges*. *Feature edges* are those that approximate smooth feature curves (or creases) defined by the traversal intersection of two smooth surfaces, whereas *nonfeature edges* are those mesh edges located in regions approximating smooth parts of the original surface.

To circumvent the problem pointed out in Section 1 with using silhouette edges for silhouette curve reconstruction, we propose the use of *silhouette triangles* for computing smooth silhouette curves of a mesh. Let M be a mesh approximating a smooth surface Γ. A triangle T of the mesh M is then an approximation to a triangular surface patch T_p on the surface Γ. For some viewing direction, the silhouette curve of Γ can lie on the patch T_p, and in this case,

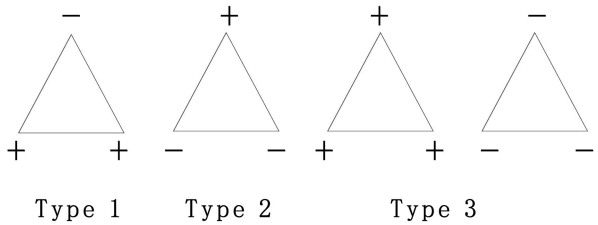


Fig. 5. Different types of triangles in a mesh. Only triangles of types 1 and 2 are silhouette triangles.

we need to compute a smooth curve on the triangle T to approximate that part of the smooth silhouette of Γ. This naturally gives rise to the notion of the *silhouette triangle*, which is the triangle T in this case. The silhouette triangles are viewpoint dependent and are exactly those mesh faces that contain silhouette curves given by $g(p) = 0$ in [7] (see Section 1.2 and Fig. 4).

Let N_V be a shading normal defined at mesh vertex V. Let D_V denote the viewing direction vector from the vertex V to the viewpoint (that is, the eye). Then, V is said to be *visible* if the inner product $(N_V, D_V) \geq 0$ and *invisible* otherwise. Note that the notion of visibility here is a local one; it is different from occlusion where a front-facing triangle may be blocked by another object in front of it. We label a visible vertex by "+" and an invisible one by "−." A mesh triangle face is called a *silhouette triangle* if its three vertices do not have the same visibility status. For example, among the four triangles with visibility labels in Fig. 5, the triangles of types 1 and 2 are silhouette triangles, whereas the other two are not.

Smooth silhouette curves of a mesh can be computed using the property that the surface normal of any point on a silhouette is perpendicular to the viewing direction. First, all silhouette triangles are identified by checking the visibility of all mesh vertices. Each silhouette triangle has exactly two edges labeled as "$(-,+)$," V_1V_2 and V_1V_3, as shown in Fig. 6. Using the endpoint information of these two edges, we compute two cubic Hermite interpolation curves, $S_1(u)$ to connect V_1 and V_2 and $S_2(v)$ to connect V_1 and V_3, which are called *silhouette bridges* (see Fig. 6).

Then, as an approximation, assuming a linear change of normal vectors along the silhouette bridges, it is easy to compute a *silhouette point* on each silhouette bridge such that

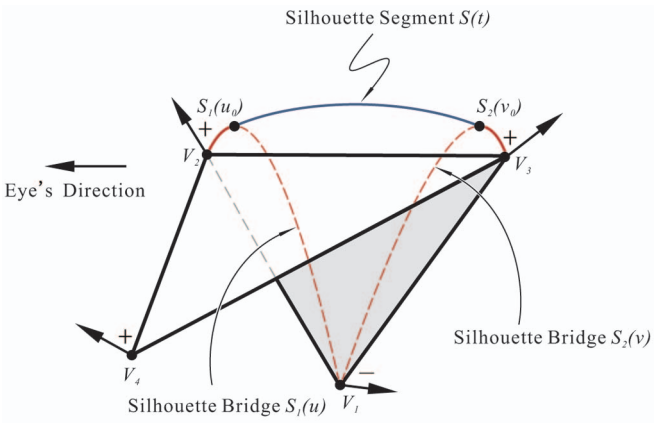


Fig. 6. Construction of silhouette bridges $S_1(u)$ and $S_2(v)$ and silhouette segment $S(t)$.

the interpolated surface normal vector at that point is perpendicular to the viewing direction. Thus, we get two silhouette points, which are $S_1(u_0)$ and $S_2(v_0)$, as shown in Fig. 6. Finally, the *silhouette curve segment*, or simply *silhouette segment*, connecting the two silhouette points is given by a cubic Hermite interpolation curve $S(t)$, as shown in Fig. 6. The silhouette segment $S(t)$ is finally sampled for local remeshing for rendering the mesh with the smoothed silhouettes.

Another goal of our method is to make polygonal feature lines smooth in a coarse mesh. (Two meshes with such polygonal feature lines are shown in Figs. 20a and 21b.) Feature edges are connected to form polygonal feature lines that approximate the original smooth feature curves. We suppose that feature edges are already labeled as such in the input mesh. Based on the two endpoints and estimated tangent vectors at the two endpoints of each feature edge, we use the cubic Hermite curve to generate a smooth *feature curve segment* (or simply *feature segment*) to replace the straight feature edge. Because of their shared tangents, consecutive feature segments are joined with G^1 continuity and thus form a smooth piecewise cubic curve approximating the original feature curve.

Clearly, feature segments need to be rendered whenever visible. To speed up processing, they are precomputed and stored for easy retrieval through quick lookup during runtime processing. The extra space requirement of this preprocessing scheme is justified by that the number of feature edges is usually small compared to the number of nonfeature edges, so storing a fixed set of feature segments normally does not cause a significant increase in memory consumption.

Below is the main flow of our method, as illustrated in Fig. 7. Details about the main steps will be presented in Section 3:

1. **Feature edge smoothing**. For each feature edge, compute its corresponding feature curve segment by Hermite curve interpolation in preprocessing. This step is view independent.
2. **Finding silhouette triangles**. Given a viewing direction D for a parallel projection or a viewpoint E for a perspective projection, locate all *silhouette triangles* with respect to D or E.
3. **Computing silhouette bridges**. Compute two *silhouette bridges* for each silhouette triangle using Hermite interpolation.
4. **Computing silhouette segments**. Compute a *silhouette point* on each silhouette bridge obtained in step 3. Use a Hermite interpolation curve again to construct a smooth *silhouette segment* connecting the two *silhouette points* of each silhouette triangle. (Note that each silhouette bridge—as well as the silhouette point on it—is computed only once in each frame, since it is shared by two adjacent silhouette triangles.)
5. **Local remeshing**. Sample points on silhouette segments and visible feature segments adaptively, according to their curvature and perceived size. Use these sample points to perform local remeshing for rendering.

We use the GPU to perform local remeshing and render the remeshed surface. Details about the implementation are given in Section 4.

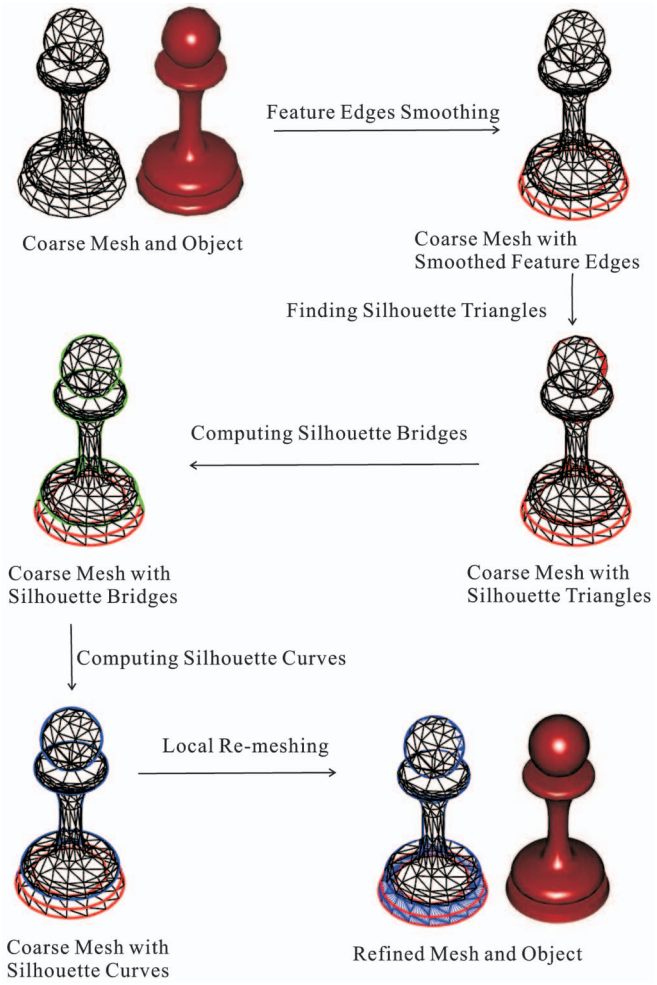


Fig. 7. Flowchart of our silhouette smoothing method.

3 Smooth Curve Construction

In this section, we explain the five main steps of our method as outlined in the preceding flow of algorithm.

3.1 Feature Edge Smoothing

The key to computing a feature segment via Hermite interpolation is providing properly estimated tangent vectors at the two endpoints of its corresponding feature edge. Feature edges can be grouped into maximal polylines, called *feature polylines*. A feature polyline either terminates at a nonfeature vertex or meets other feature polylines at a feature vertex where more than two smooth surfaces intersect (see Fig. 2). Note that no smoothing is needed when a feature polyline contains only a single feature edge.

Suppose that a feature polyline contains at least three consecutive vertices $(V_0, V_1, \ldots, V_k)$, $k \geq 2$. For an internal vertex V_i, $i = 1, 2, \ldots, k-1$, the tangent direction $\hat{T}_i$ at V_i is set to be the tangent to the circle uniquely determined by the three consecutive points V_{i-1}, V_i, and V_{i+1}. Let $W_0 = V_i - V_{i-1}$ and $W_1 = V_{i+1} - V_i$. Then, it is an elementary exercise to show that

$$\hat{T}_i = |W_1| \cdot W_0 + |W_0| \cdot W_1. \quad (1)$$

The estimated tangent vector T_i at V_i to be used for Hermite interpolation has the same direction of $\hat{T}_i$ but has a length determined in such a way that the resulting Hermite curve

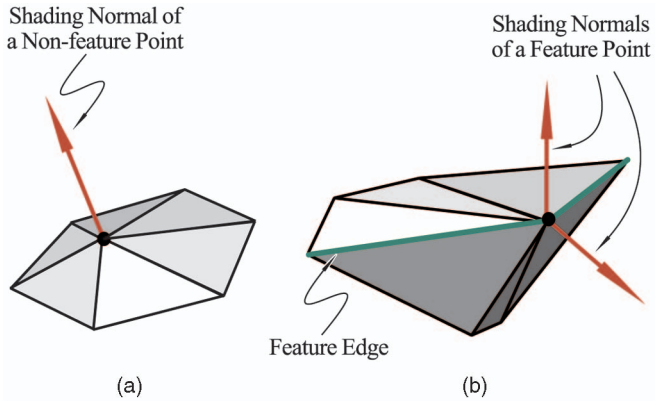


Fig. 8. (a) One shading normal is associated with a nonfeature vertex. (b) Here, two normals are assigned to a feature vertex lying at the intersection of two smooth surface patches, represented by two groups of triangles.

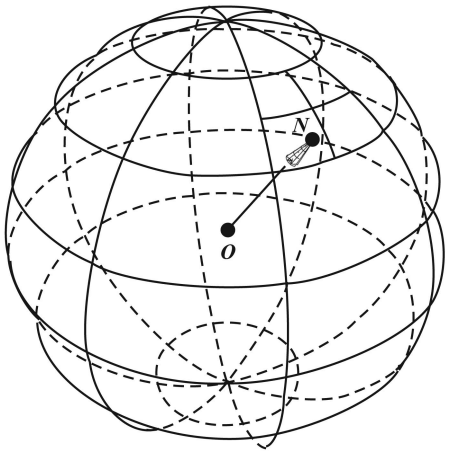


Fig. 9. Hierarchical clustering structure on the Gaussian sphere.

is a good approximation to a circular arc if the Hermite data (that is, the endpoints and their tangents) are sampled from a circle (see details in Section 3.3).

For the end vertex V_0 of the feature polyline, its tangent vector is set to be the tangent to the circle passing through V_0, V_1, and V_2. The tangent vector at the other end vertex V_k is similarly computed. Given V_i, V_{i+1}, and their tangent vectors, their cubic Hermite interpolation curve is then uniquely determined. (The expression is given in Section 3.3.)

3.2 Finding Silhouette Triangles

Silhouette triangles are determined by the visibility status of its vertices, which in turn depends on the angles between the shading normal vectors at the vertices and the viewing direction vector. For a nonfeature mesh vertex V, its shading normal vector is computed as the angle-weighted average of the normal vectors of triangles incident to V, as in [18] (see Fig. 8a).

A feature vertex V has multiple shading normal vectors, each associated with a surface incident to V, since V is at the intersection of multiple smooth surfaces. Specifically, the triangles adjacent to V are divided into several groups, with each group belonging to a smooth surface passing through V (see Fig. 8b for the case of two groups). The angle-weighted average normal vector over the triangles in the same group is assigned as a shading vector to V. Therefore, the vertex V has multiple shading vectors, each contributed by a group of triangles incident to V. When testing whether a triangle incident to a feature vertex V is a silhouette triangle, we use the normal vector associated with the group containing that triangle.

To parse *silhouette triangles*, we compute the visibility status of all mesh vertices by hierarchical clustering based on the normals of the vertices in the case of parallel projection or the normals and positions of the vertices in the case of perspective projection. For parallel projection, we follow the method in [2]. Specifically, let D be the constant viewing direction vector and let N be the normal of the vertex V. The vertex V is visible if $N \cdot D \geq 0$ and invisible otherwise. The normalized normal vectors N of all mesh vertices are mapped to points on the Gaussian sphere, which is subdivided into hierarchical cells along longitudes and latitudes. Each cell is a spherical convex region and has four corner points, and each parent cell has four child cells, as shown in Fig. 9. The cells induce a hierarchical clustering of mesh vertices via their normal vectors.

For a cluster C of mesh vertices, if all the four corners of its containing cell are visible (or invisible), then all the vertices in C are visible (or invisible). If the four corners of the cell have different visibility status, each child of the cell will be checked; this is done recursively until the visibility of each mesh vertex is resolved.

For perspective projection, the basic idea is the same, but a 3D Gaussian sphere in 4D space will be used instead, since the viewing direction is no longer a constant vector. Let $V = (V_x, V_y, V_z)$ be a mesh vertex. Let $N = (N_x, N_y, N_z)$ be the unit normal vector of the vertex V and let $E = (E_x, E_y, E_z)$ be the viewpoint. The plane passing through V and having N as its normal vector has the equation $F(X; V) \equiv N \cdot (X - V) = 0$.

Clearly, the vertex V is visible if and only if the viewpoint E is above the plane $F(X; V) = 0$, that is, $F(E; V) \geq 0$, which is the condition we use for determining the visibility of all mesh vertices V. Note that

$$\begin{aligned} F(E;V) &= E_x \cdot N_x + E_y \cdot N_y + E_z \cdot N_z \\ &\quad - (N_x \cdot V_x + N_y \cdot V_y + N_z \cdot V_z) \\ &\equiv E' \cdot V', \end{aligned}$$

where $E' = (E_x, E_y, E_z, 1)$, and $V' = (N_x, N_y, N_z, -(N_x \cdot V_x + N_y \cdot V_y + N_z \cdot V_z))$ are 4D vectors. Therefore, we normalize V', map them onto the 3D Gaussian sphere S^3 in 4D space, and subdivide S^3 for hierarchical clustering. In this case, each cluster has eight corner points and eight child cells. Then, this hierarchical structure is used for fast visibility determination for all mesh vertices. The visibility checking for a point V' on S^3 is simply done by the sign of the inner product $E' \cdot V'$.

3.3 Computing Silhouette Bridges

By definition, there are two sides labeled $(+, -)$ in a silhouette triangle, such as V_1V_2 and V_1V_3 in the silhouette triangle $\triangle V_1V_2V_3$ in Fig. 6. Below, we use the side V_1V_2 to explain the procedure for computing a silhouette bridge by Hermite interpolation.

First, suppose that V_1V_2 is not a feature edge. Let N_1 and N_2 be estimated normal vectors at V_1 and V_2. To compute

tangent vectors at V_1 and V_2, which are needed by Hermite interpolation, we take the tangent direction $\hat{T}_1$ at the vertex V_1 to be the projection of $\overrightarrow{V_1V_2}$ on the "tangent" plane at V_1, which is the plane passing through V_1 and having N_1 as the normal vector. Then, the vector $\hat{T}_1$ is given by

$$\hat{T}_1 = (V_2 - V_1) - [(V_2 - V_1) \cdot N_1] \cdot N_1. \tag{2}$$

Similarly, the tangent direction $\hat{T}_2$ at V_2 is

$$\hat{T}_2 = (V_2 - V_1) - [(V_2 - V_1) \cdot N_2] \cdot N_2. \tag{3}$$

The tangent vectors T_1 and T_2 we use in Hermite interpolation have the same directions of $\hat{T}_1$ and $\hat{T}_2$. Obviously, different lengths of the end tangent vectors affect the shape of the resulting cubic Hermite interpolation curve. We choose the length of T_1 and T_2 in such a way that if the end data points (that is, V_i and T_i, $i = 1, 2$) are extracted from a circular arc, then the resulting Hermite cubic curve gives a good approximation of the circular arc. It can be shown that this requirement is met if the lengths of T_1 and T_2 are set to be

$$L_1 = \frac{2|V_2 - V_1|}{1 + \cos\theta_1}, \qquad L_2 = \frac{2|V_2 - V_1|}{1 + \cos\theta_2}, \tag{4}$$

where θ_i is the angle between $\overrightarrow{V_1V_2}$ and $\hat{T}_i$, $i = 1, 2$. Here, a circular arc is used as a target shape for approximation because it has constant curvature; hence, it is expected that the interpolating cubic curve will have small curvature variation when the endpoint data does not deviate much from a circular arc.

With all the end data points determined, the silhouette bridge $S_1(u)$ over the side V_1V_2 is given by the cubic Hermite curve:

$$S_1(u) = (2V_1 - 2V_2 + T_1 + T_2)u^3 - (3V_1 - 3V_2 + 2T_1 + T_2)u^2 + T_1u + V_1, \quad u \in [0, 1]. \tag{5}$$

3.4 Computing Silhouette Segments

A *silhouette point* is a point on a silhouette bridge whose surface normal vector is perpendicular to the viewing direction. Here, we assume that the normal vector along the silhouette bridge $S_1(u)$ over V_1V_2 is linearly interpolated from the normal vectors at V_1 and V_2. Thus, we assign

$$\tilde{N}_1(u) = (1 - u) \cdot N_1 + u \cdot N_2, \quad u \in [0, 1], \tag{6}$$

to be the normal vector at the point $S_1(u)$. First, consider the case of parallel projection, with the constant viewing direction vector denoted by D. The silhouette point on the silhouette bridge $S_1(u)$ is $S_1(u_0)$, where the parameter value u_0 is easily obtained as the solution to the linear equation $D \cdot \tilde{N}_1(u) = 0$. The other silhouette point $S_2(v_0)$ on the silhouette bridge $S_2(v)$ can similarly be computed.

For perspective projection, the viewing direction at the point $S_1(u)$ is $D(u) = S_1(u) - E$, which is no longer a constant vector. That means that we would have to solve the quartic equation $D(u) \cdot \tilde{N}_1(u) = 0$ to locate the silhouette point $S_1(u_0)$, since $S_1(u)$ is cubic. To obtain a simple approximate solution, we use

$$\hat{D}(u) = (1 - u) \cdot D_1 + u \cdot D_2, \quad u \in [0, 1], \tag{7}$$

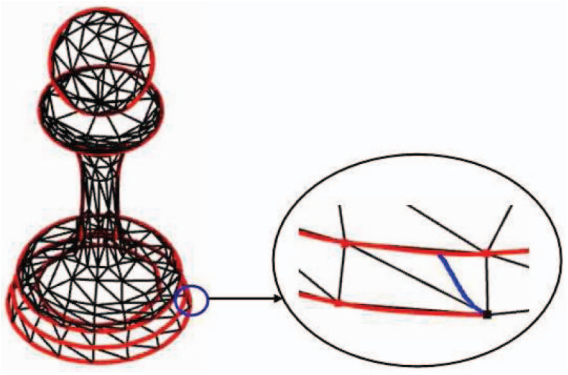


Fig. 10. An example of a silhouette triangle containing a feature edge (the close-up is a side view). The red curves are smooth feature segments, and the blue one is a silhouette segment.

where D_1 and D_2 are viewing directions at V_1 and V_2, to approximate the true viewing direction $D(u)$. This approximation $\hat{D}(u)$ makes sense because it agrees with the true viewing direction $D(u)$ at the endpoints V_1 and V_2 and thus gives the correct visibility status at V_1 and V_2. With this approximation, we just need to solve the quadratic equation $h(u) \equiv \hat{D}(u) \cdot \tilde{N}_1(u) = 0$ to get u_0 so as to determine the silhouette point $S_1(u_0)$. Note that the quadratic equation $h(u) = 0$ has a unique solution in [0, 1], since $h(u)$ has opposite signs at $u = 0$ and $u = 1$.

Once the two silhouette points $S_1(u_0)$ and $S_2(v_0)$ are available, together with their normal vectors $\tilde{N}_1(u_0)$ and $\tilde{N}_2(v_0)$, we compute the silhouette segment using the Hermite interpolation in the same way as in computing silhouette bridges. Thus, we get the silhouette segment $S(t)$ associated with the silhouette triangle $\triangle V_1V_2V_3$ (see Fig. 6).

Special consideration is needed when a feature edge is one of the sides of a silhouette triangle. There are two cases: 1) a feature edge is a side whose endpoints have different visibility status (for example, the side V_1V_2 or V_1V_3 in Fig. 6), and 2) a feature edge is the side whose endpoints have the same visibility status (for example, the side V_2V_3 in Fig. 6). In case 1, the silhouette bridge over the feature edge is simply its corresponding precomputed feature segment. (Fig. 10 illustrates an example of case 1 where a feature edge is a side of a silhouette triangle.)

In case 2, no special treatment is actually needed. In this case, assuming that the feature edge is the side V_2V_3 in Fig. 6, if a changing viewing direction makes the normal vectors at the two endpoints of V_2V_3 get closer and closer to being perpendicular to the viewing direction, then the silhouette points $S_1(u_0)$ and $S_2(v_0)$ on the two silhouette bridges will approach V_2 and V_3, respectively, which ensures that the reconstructed smooth silhouette segment $S(t)$ approaches the feature segment associated with the feature edge V_2V_3. This is important for the visual coherence of silhouette curves with respect to a moving viewpoint when feature segments are involved, and it is achieved by smoothing feature edges using the same curve interpolation scheme as for silhouettes.

3.5 Local Remeshing

In this section, we shall discuss how to remesh silhouette regions and how to adaptively sample silhouette segments

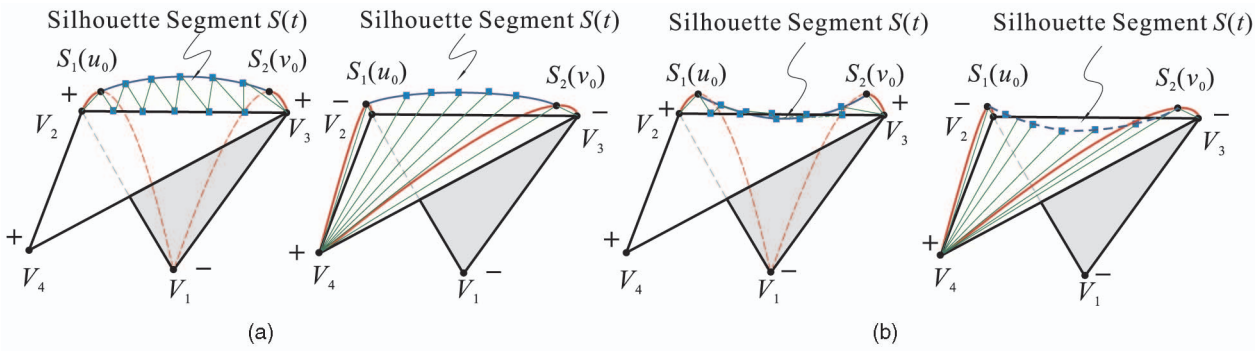


Fig. 11. Local remeshing. (a) (left) The silhouette triangle is $\triangle V_1V_2V_3$, and the silhouette segment $S(t)$ is further back than the edge V_2V_3. To attain local convexity, the sample points on the segment $S(t)$ and the edge V_2V_3 are connected to form local triangulation. (a) (right) The silhouette triangle is $\triangle V_2V_3V_4$, and $S(t)$ is in front of the edge V_2V_3. In this case, again, to attain local convexity, the sample points on the segment $S(t)$ are connected to the vertex V_4. (b) The two cases shown here involve concave silhouette curve segments and are processed similarly; local vertex perturbation is needed in these cases to rectify the blocking problem.

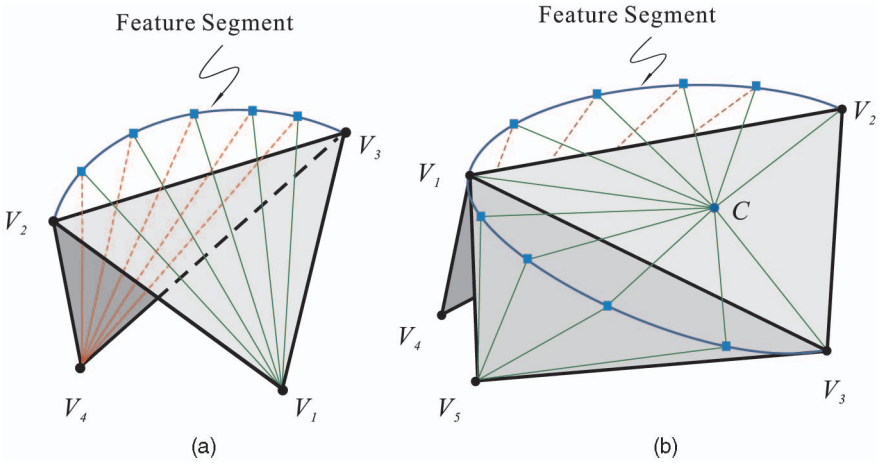


Fig. 12. Remeshing when feature segments are involved. (a) One feature segment: the sample points on the feature segment are connected to the opposite vertex. (b) Two feature segments: the sample points on the two feature segments are connected to the center C of the triangle $\triangle V_1V_2V_3$.

and then address the issue of silhouette blocking when concave silhouette segments are involved.

Remeshing. To render a mesh surface with smooth feature segments and silhouette segments that we have constructed via Hermite interpolation, these curve segments need to be sampled for local remeshing. Fig. 11 shows different cases in which the curve should be remeshed, and Fig. 12 shows how the remeshing should be done when feature curves are involved. Note that when the silhouette curve is concave, it may be blocked by a neighboring triangle of the original mesh. We will address this issue later in this section.

Adaptive sampling. The reconstructed smooth silhouette curve will be rendered as a polygon connecting sample points on the silhouette. To ensure that the silhouette curve appears smooth after sampling while keeping low the number of sample points, we determine the number of sample points on the silhouette segment adaptively based on the curvature and the projected length of the silhouette segment. Since the silhouette segment is intended as an approximation to a circular arc (cf. Section 3.3), using the simple geometry of a circular arc, we derive an estimate on the number of sample points as follows:

Let A and B denote the endpoints of the silhouette segment $S(t)$, $t \in [0,1]$. (Refer to Fig. 13). Let h be the height of the silhouette segment in the projection plane, approximated by the distance from the middle point $S(1/2)$ of the silhouette segment to the middle point of the line segment AB. Let $r = \sin(\alpha/2) = \sqrt{(1-\cos(\alpha))/2}$, where α is the central angle of the arc subtended by the chord AB and so $\cos(\alpha)$ can be approximated by the inner product of the two unit normal vectors at the endpoints A and B. Let e be the allowed tolerance, in pixels. Then, the bound n_s on the number of sample points is given by

$$n_s = \left\lceil \frac{1}{2} \cdot \sqrt{h \cdot r/e} \right\rceil. \tag{8}$$

The interpretation of this bound n_s is that if the number of sample points is at least n_s, then the polygon connecting the sample points approximate the smooth silhouette curve

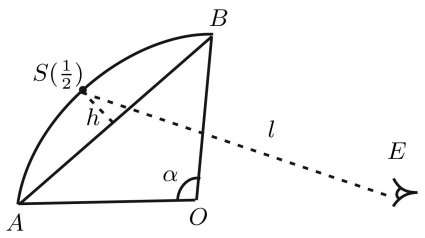


Fig. 13. Adaptive sampling of a silhouette curve segment.

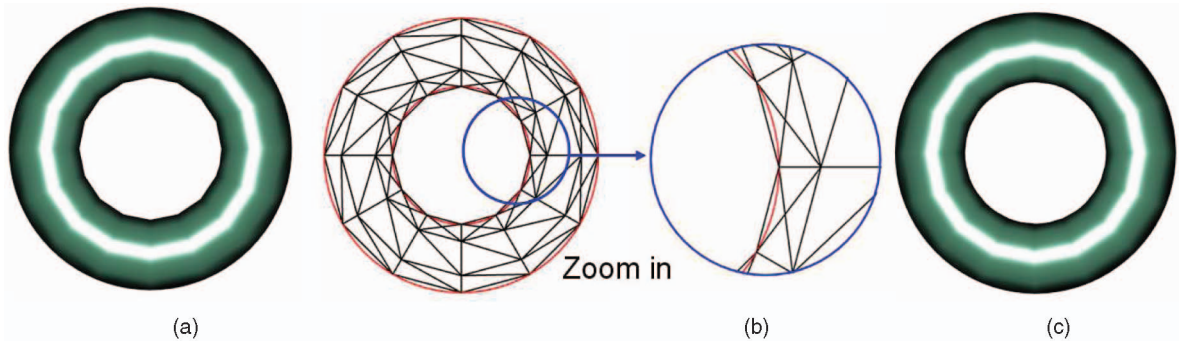


Fig. 14. An example of silhouette smoothing in a saddle-shaped region. (a) The result without fixing by vertex perturbation. (b) The correspondence mesh model of (a) and a close-up view. The red curve is the smoothed silhouette blocked by the original mesh. (c) The result after fixing by vertex perturbation.

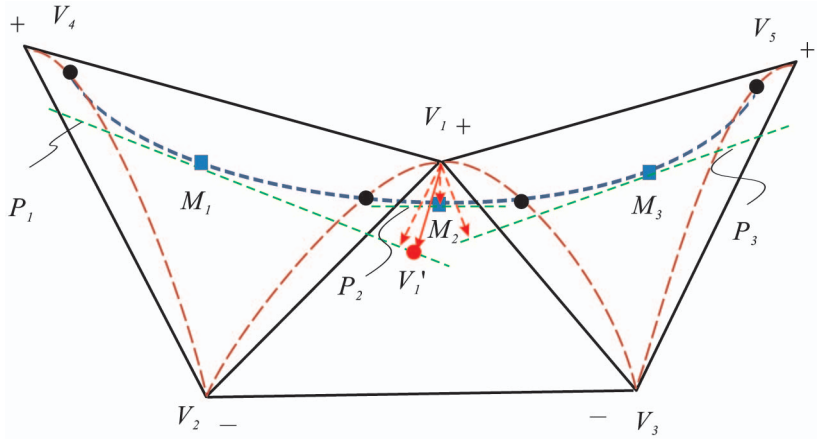


Fig. 15. Vertex perturbation to remove silhouette blocking. The dashed blue curve is the smoothed silhouette, which is blocked by three adjacent triangles. By lowering the vertex V_1 to V_1', the smoothed silhouette curve is displayed properly.

within the error e. Here, the value of e should be in the range of 0.5 to 2 to ensure the smooth appearance of the rendered silhouette.

Local perturbation. In a saddle-shaped region the reconstructed curved silhouette curve may be blocked by its neighboring triangles, and thus, the original polygonal silhouette still persists, as already seen in Fig. 11b. Fig. 14a provides a convincing example with detailed explanation where the inner circle of the torus does not appear to have been smoothed due to this blocking problem. This occurs because the reconstructed silhouette curve moves into the mesh surface and so may be blocked by adjacent mesh triangles, which is an issue that has not been addressed by previous view-dependent local refinement methods (for example, [20] and [4]). We propose below a simple and effective perturbation technique to rectify this problem.

First, we need to detect concave silhouette segments. Let $\tilde{N}_1$ and $\tilde{N}_2$ be the surface normal vectors at two consecutive silhouette points S_1 and S_2, respectively. Denote $F = S_2 - S_1$. Then, the silhouette segment connecting S_1 and S_2 is concave if $\tilde{N}_1 \cdot F > 0$ and $\tilde{N}_2 \cdot F < 0$, that is, the angle between $\tilde{N}_1$ and F is less than $\pi/2$, and the angle between $\tilde{N}_2$ and F is greater than $\pi/2$. After all concave silhouette segments are found, the following perturbation technique will be applied.

The basic idea is to perturb positions of the neighboring mesh vertices of a concave silhouette segment to keep them from blocking the silhouette segment. Let T denote the silhouette triangle containing the concave silhouette segment under consideration. Then, by neighboring mesh vertices, we mean those vertices of the triangle T or vertices that are connected to T through a mesh edge. For example, in Fig. 15, as far as the silhouette triangle $\Delta V_1V_2V_3$ is concerned, all V_i, $i = 1, 2, 3, 4, 5$, are neighboring vertices, so they will be subject to perturbation.

For the simplicity of discussion, we will use the vertex V_1 to explain the perturbation procedure in the case of parallel projection; the idea is similar for the case of a perspective projection. Refer to Fig. 15. Suppose that $S_i(t)$, $i = 1, 2, 3$, $t \in [0, 1]$, are all the concave silhouette segments that involve V_1 as a neighboring point. Take the middle points $M_i = S_i(1/2)$ of the silhouette segments $S_i(t)$. Let N_i denote the interpolated normal vector at M_i. Then, each silhouette segment $S_i(t)$ is associated with a plane $\mathcal{P}_i$ that passes through M_i and has N_i as its normal vectors. It can be shown that the plane $\mathcal{P}_i$ contains the viewing direction. The upper side of the plane $\mathcal{P}_i$ is defined by the inequality $N_i \cdot (X - M_i) > 0$. Clearly, the silhouette segment $S_i(t)$, $t \in [0, 1]$, lies entirely above the plane $\mathcal{P}_i$.

Now, we shall determine a displacement vector for V_1 such that after perturbation, V_1 will be below each plane $\mathcal{P}_i$, $i = 1, 2, 3$. The direction R of this displacement vector is the average of all the normal vectors N_i, that is, $R = N_1 + N_2 + N_3$, whose normalized vector is denoted by $\hat{R} = R/\|R\|$. It follows that the length of this displacement vector is given by

$$\ell = \max_{i=1,2,3} \left\{ -\frac{(V_1 - M_i) \cdot N_i}{\hat{R} \cdot N_i} \right\}.$$

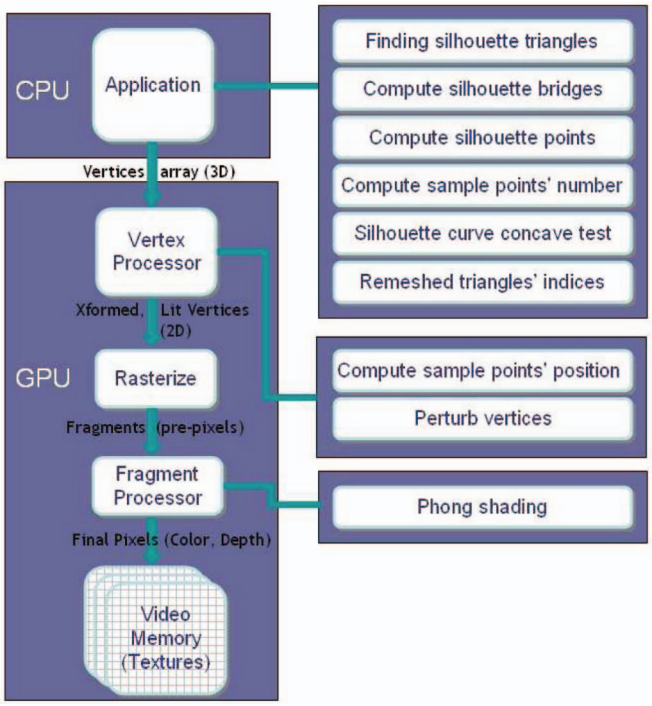


Fig. 16. The graphics pipeline.

Hence, V_1 is displaced to V_1' by

$$V_1' = V_1 + \ell \hat{R}.$$

Similarly, we perturb all other neighboring vertices of concave silhouette segments. Since for each silhouette segment $S_i(t)$, all vertices of its neighboring triangles are below the plane $\mathcal{P}_i$, these triangles will not block $S_i(t)$ with respect to the viewpoint, which is on the plane $\mathcal{P}_i$.

The above simple scheme for blocking removal has proven very effective. Fig. 14 shows a coarse mesh with a close-up view to reveal the blocking phenomenon. Fig. 14c shows the unblocked smooth silhouette after applying the above vertex perturbation. The example shown in Fig. 22 also demonstrates the effectiveness of this perturbation scheme.

Above, only those mesh triangles within a small neighborhood of concave silhouette segments are considered for perturbation to avoid silhouette blocking. Although it can be imagined that triangles little beyond this neighborhood may also block the concave silhouette segments, we do not consider this extension in order to minimize implementation complexity. Moreover, such potential blocking by relatively far neighboring triangles must occur very rarely, since they have not been visually discernable in our extensive experiments.

4 Speedup with GPU

To speed up the smooth processing and the rendering of the refined mesh, we use the GPU, as well as the CPU, for the computation described in Section 3. We balance the loads of the GPU and CPU as shown in Fig. 16 to parallelize the computation as much as possible.

After the CPU computes the silhouette segments and completes remeshing, the GPU is fed with mesh vertices accompanied by remeshing information. Vertex perturbation and the locations of sample points on silhouette segments are computed by the GPU. After filtering the vertices by vertex processors, the positions of all the vertices are transformed to the homogenous clip space by the vertex program of the GPU, and then, all the triangles are rendered in Phong shading by the fragment processors of the GPU.

5 Experimental Results

In this section, we shall present experimental results of testing our method and compare our method with the methods in [20] and [19] in terms of rendering quality and efficiency.

Figs. 1, 17, 18, 19, 22, and 23 show examples generated by our method. The fine meshes shown there are used only as a reference for comparison to the remeshed coarse meshes with reconstructed smooth silhouettes. For the textured mesh in Fig. 19, the linear interpolation of texture coordinates is used to assign texture coordinates to the sample points on silhouette curves. Fig. 22 shows how the silhouette blocking problem in a saddle-shaped region (cf. Section 3.3) is fixed by our vertex perturbation technique. Fig. 23 provides an example that our method also works well for meshes with nonuniformly sampled data points. The real-time rendering of these models is demonstrated in the submitted video.

Figs. 20 and 21 show two mesh surfaces with prominent feature edges. Our method constructs satisfactory smooth feature curves from the polygonal feature edges. Although a similar treatment to feature curve smoothing has been studied before [4], we have shown here how it can be integrated with our silhouette smoothing technique in a

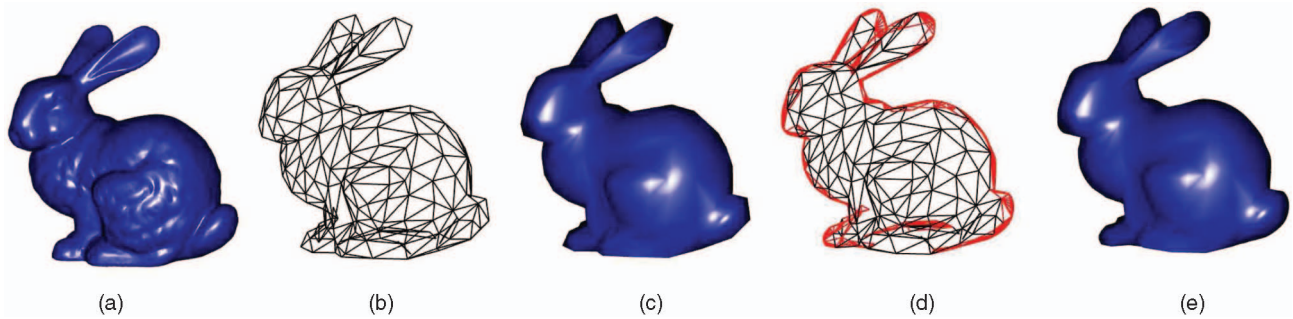


Fig. 17. Bunny. (a) Shaded fine mesh (30,000 triangles). (b) Coarse mesh (500 triangles). (c) Shaded coarse mesh. (d) Remeshing by our method (873 triangles). (e) Phong shading of the mesh in (d).

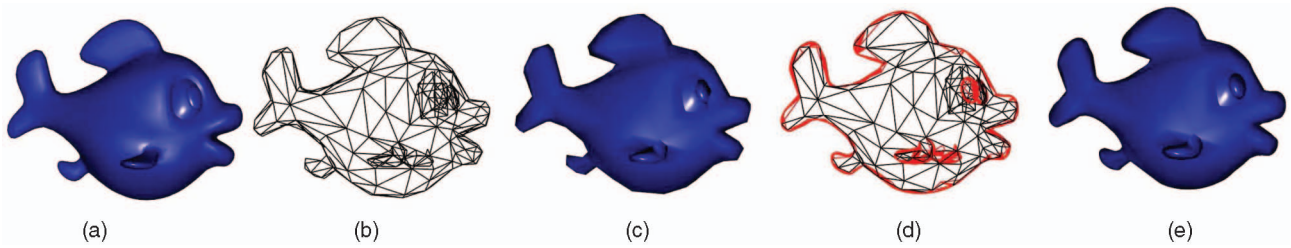


Fig. 18. Fish. (a) Shaded fine mesh (5,000 triangles). (b) Coarse mesh (500 triangles). (c) Shaded coarse mesh. (d) Remeshing by our method (1,018 triangles). (e) Phong shading of the mesh in (d).

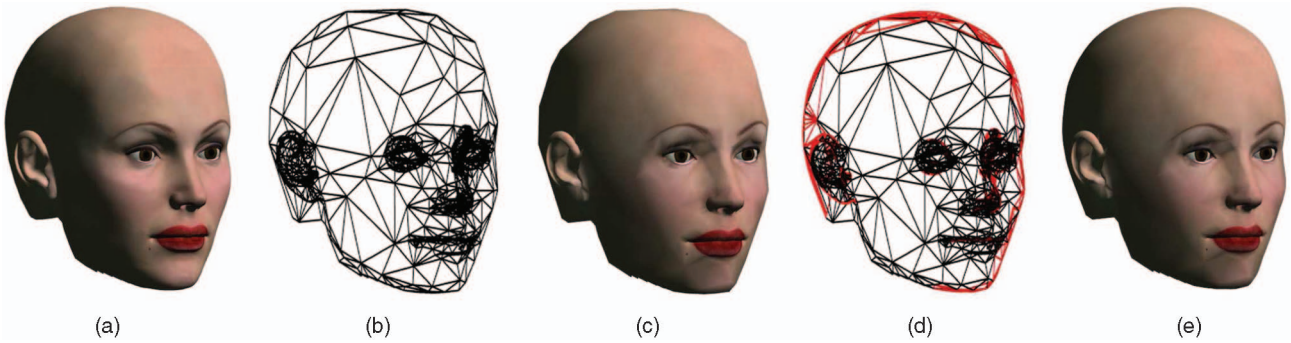


Fig. 19. Texture head. (a) Shaded fine mesh (43,151 triangles). (b) Coarse mesh (4,778 triangles). (c) Shaded coarse mesh. (d) Remeshing by our method (6,858 triangles). (e) Phong shading of the mesh in (d).

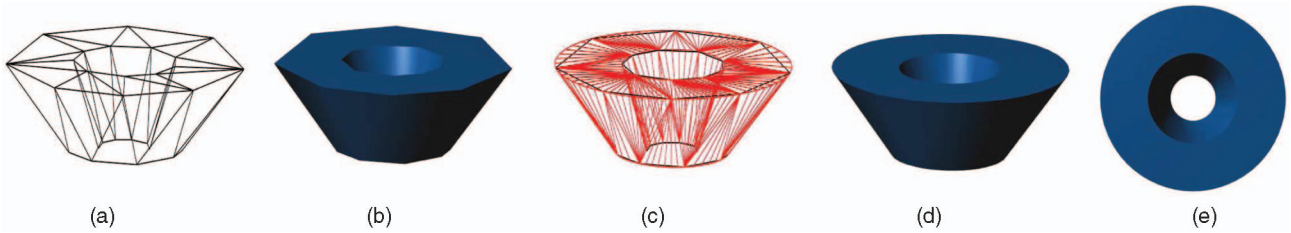


Fig. 20. Cone mesh. (a) Coarse mesh (70 triangles). (b) Shaded coarse model. (c) Remeshing by our method (576 triangles). (d) Phong shading of the mesh in (c). (e) Another view.

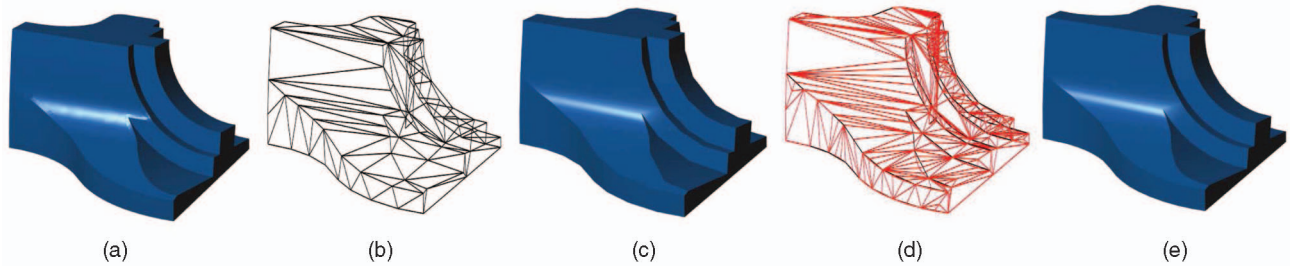


Fig. 21. Fandisk. (a) Fine mesh (1,000 triangles). (b) Coarse mesh (128 triangles). (c) Shaded coarse mesh. (d) Remeshing by our method (1,899 triangles). (e) Phong shading of the mesh in (d).

unified framework (for example, see the mesh model of a pawn in Fig. 10).

Fig. 22 compares our method with the method in [20]. The result by the method in [20] is not as good, because the interpolated silhouette curve used there corresponds to some mesh edges and always passes through mesh vertices, leading to visual artifacts. In contrast, the method of the present paper generates a more faithful smooth silhouette that is visually coherent with a moving viewpoint or a moving object, as can be seen in the accompanying video.

Compared with the PN-triangle method in [19], besides memory saving due to local remeshing, our method supports smooth rendering of silhouette with arbitrary "zoom-in," an operation used frequently in applications. Fig. 24b shows a refined mesh after three levels of subdivision by the PN-triangle method. The silhouette by the PN-triangle method looks smooth in the current view, but its polygonal appearance is revealed when zooming in for a close-up view. To make silhouettes smooth, the whole model would need to be subdivided again, and the number of triangle faces would increase significantly, since the PN-triangle method operates in a viewpoint-independent manner, insensitive to the changing viewing distance and the changing location of silhouettes.

In contrast, the number of sample points is determined adaptively in our method according to the curvature and the projected height of silhouette segments. Therefore, our method ensures that the silhouette curve always looks smooth via simple runtime adaptive local remeshing, when the model is scaled larger during zoom-in (see Fig. 24c).

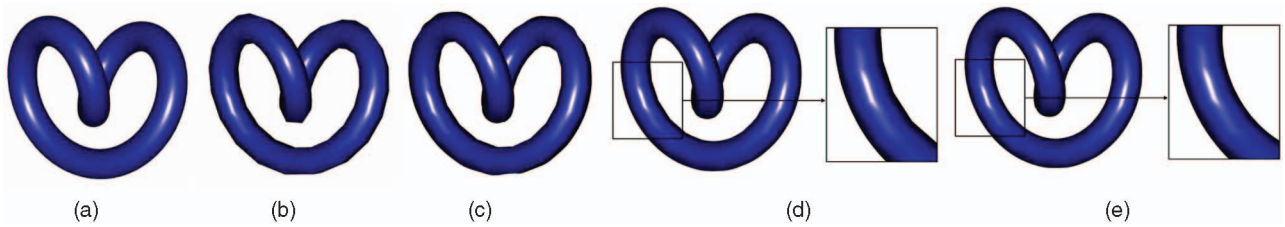


Fig. 22. Knot. (a) Shaded fine mesh (15,000 triangles). (b) Shaded coarse mesh (480 triangles). (c) Result by the method in [20]. (d) Shaded refined mesh by our method without local perturbation (843 triangles). (e) Shaded refined mesh by our method with local perturbation.

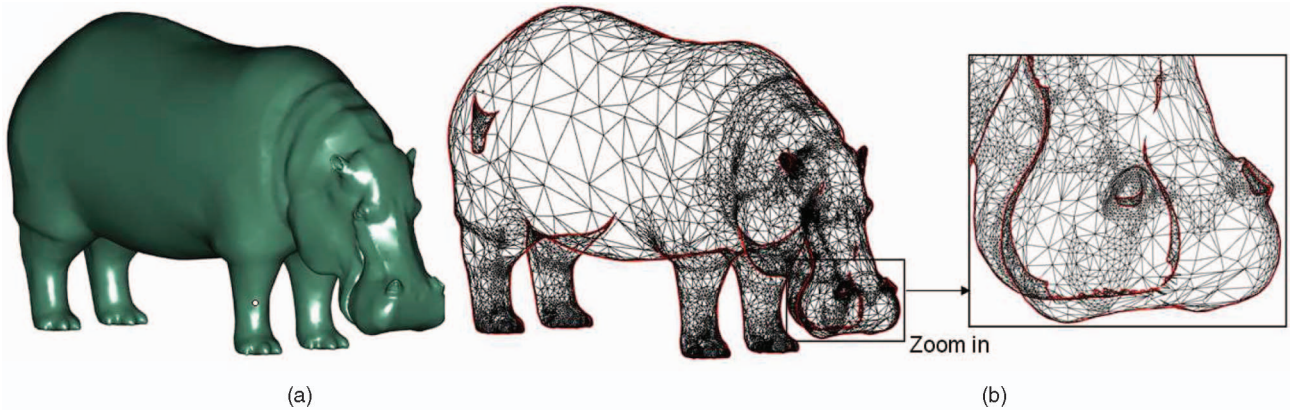


Fig. 23. Hippo. An example of smoothing the silhouettes of a nonuniformly sampled model. (a) Shaded refined model. (b) The refined mesh model by our method and a close-up view.

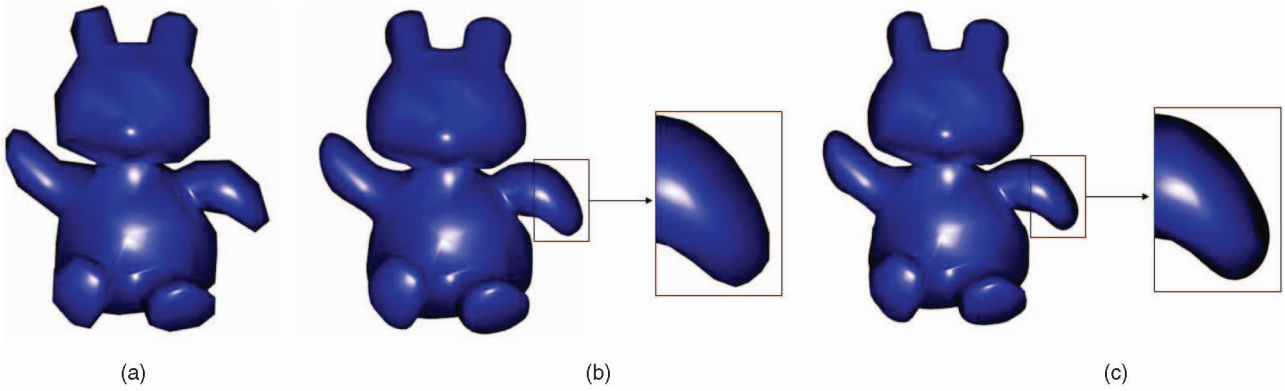


Fig. 24. Bear. (a) A coarse mesh (500 triangles). (b) The result by the PN-triangle method and a zoom-in view, showing nonsmoothness. (c) The result by our method and the zoom-in view of the same part.

As shown in Fig. 24, although the PN-triangle method uses global patch reconstruction, the quality of the interior shading is the same with the shaded mesh refined by our method or the shaded coarse mesh. That is because the Phong shading model, which is used in our experiments, is based on normal interpolation, and the normal computed by the Phong shading model at interior points of a triangle of the coarse mesh is almost the same as the normal produced by the PN-triangle method at the points. Hence, the interior shading quality has been improved little by the PN-triangle method despite of the increased number of triangles it uses for rendering.

We tested our silhouette smoothing method on a PC workstation with one 3.0-GHz CPU, 512 Mbytes of memory, and an Nvidia's GeForce 7800 GT graphics card. The window size was set to 1,280 $\times$ 800. The testing objects covered areas around 40-60 percent of the window.

Table 1 compares the frame rates when applying the method in [20], the PN-triangle method [19], and our method. The column "Direct Drawing" is the control, which shows the frame rates when drawing objects without silhouette smoothing. All the methods have used vertex arrays to obtain the best frame rates possible. For the PN-triangle method, the frame rates depend on the subdivision level. In this test, the subdivision level varies from 2 to 3 to make the model's silhouettes look smooth to give comparable visual quality. Since unlike the PN-triangle method, the number of triangles rendered by our method in each frame varies with the viewing parameters, only the typical counts of triangles for our method are listed in Table 1 for reference.

Since the three methods are designed for enhancing visual quality of rendered coarse meshes, in the comparisons, we use only coarse meshes with no more than 600 triangles except the last one, the "hippo" model. Note that the rendering of such simple models using a

TABLE 1
Performance Comparisons of Our Method and the Other Methods

Mesh	Tri.count			Phong shading frame rates (fps)				
	Original	PN method (LOD)	Our method	Direct Drawing	Wang's.* method	PN** method	Our method Ortho	Our method Perspective
Sphere	200	3200 (3)	332	573	60	115	352	347
Knot	480	7680 (3)	884	605	56	53	152	147
Bunny	500	8000 (3)	800	442	53	50	75	68
Fish	500	4500 (2)	886	620	54	77	114	114
Bear	500	8000 (3)	893	633	54	52	117	112
Head	500	8000 (3)	904	448	52	54	78	75
Venus	500	4500 (2)	862	655	56	76	155	153
Cat	500	8000 (3)	842	372	53	52	74	74
Cow	500	8000 (3)	983	560	52	51	82	80
Torus	576	5184 (2)	960	592	57	67	172	167
Hippo(render once)	43288	389592 (2)	51842	462	43	60	99	96

*Wang's method refers to the one in [20]. **The PN-triangle method refers to the one in [19].

modern GPU is very fast. The frame rate for rendering a single model of that size is usually in the range of thousands of *frames per second* (fps) that is easily subject to noises caused by timing inaccuracy, difference in GPU design, and display driver tweaks. Therefore, 60 copies of the same models were drawn to make the rendering frame rates stay at the normal range of less than 300 fps, except for the "hippo" model, which was drawn only once.

As shown in Table 1, our method is typically two to three times faster than the method in [20] for different models. Comparing with the PN-triangle method in [19], our method also shows a 30 percent to more than 200 percent speedup for all tested models. From our experiments, it has been observed that the part of extracting silhouette triangles and the part of local perturbation take only a small part of the overall computation and rendering time (less than 4 percent for each part), whereas the parts on computing the smoothed silhouettes and sampling points on them take the major portion of the total time. Furthermore, with our scheme, we have found that the tasks for the CPU and those for the GPU take about an equal amount of time. It is, however, possible that there is a better scheme that is more suitable for the GPU computation so as to further speed up the method.

6 Conclusions

We have presented a method for silhouette smoothing of coarse meshes through 3D curve interpolation and local remeshing based on the notion of *silhouette triangles*. Both smooth silhouette curves and feature curves are computed in a unified framework. We have demonstrated the effectiveness of the method using a number of examples. This method is a promising alternative to LOD-based view-dependent methods, especially when a fine mesh or an LOD model is not available.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (60473103 and 60473127) and the National Basic Research Program of China (2006CB303102).

References

[1] D.I. Azuma, D.N. Wood, B. Curless, T. Duchamp, D.H. Salesin, and W. Stuetzle, "View-Dependent Refinement of Multiresolution Meshes with Subdivision Connectivity," *Proc. Second Int'l Conf. Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa (AFRIGRAPH '03),* pp. 69-78, 2003.

[2] F. Benichou and G. Elber, "Output Sensitive Extraction of Silhouettes from Polygonal Geometry," *Proc. Seventh Pacific Conf. Computer Graphics and Applications (PG '99),* pp. 60-69, 1999.

[3] T. Boubekeur, P. Reuter, and C. Schlick, "Scalar Tagged PN Triangles," *Proc. Eurographics '05,* pp. 17-20, 2005.

[4] C. Dyken and M. Reimers, "Real-Time Linear Silhouette Enhancement," *Proc. Math. Methods for Curves and Surfaces,* pp. 135-143, July 2004.

[5] J.D. Foley, A. van Dam, S. Feiner, and J. Hughes, *Computer Graphics: Principles and Practice.* Addison-Wesley, 1990.

[6] P. Heckbert and M. Garland, "Survey of Polygonal Surface Simplification Algorithms," *ACM SIGGRAPH '97 Course Notes 25,* 1997.

[7] A. Hertzmann and D. Zorin, "Illustrating Smooth Surfaces," *Proc. ACM SIGGRAPH '00,* pp. 517-526, 2000.

[8] H. Hoppe, "View-Dependent Refinement of Progressive Meshes," *Proc. ACM SIGGRAPH '97,* pp. 189-198, 1997.

[9] H. Hoppe, "Smooth View-Dependent Level-of-Detail Control and Its Application to Terrain Rendering," *Proc. IEEE Conf. Visualization (VIS '98),* pp. 35-42, 1998.

[10] D. Kirsanov, P.V. Sander, and S.J. Gortler, "Simple Silhouettes for Complex Surfaces," *Proc. Eurographics Symp. Geometry Processing (SGP '03),* pp. 102-106, 2003.

[11] J.J. Koenderink, "What Does the Occluding Contour Tell Us about Solid Shape," *Perception,* vol. 13, pp. 321-330, 1984.

[12] S. Kumar, D. Manocha, B. Garrett, and M. Lin, "Hierarchical Back-Face Culling," *Proc. Seventh Eurographics Workshop Rendering,* pp. 231-240, 1996.

[13] D.P. Luebke and C. Erikson, "View-Dependent Simplification of Arbitrary Polygonal Environments," *Proc. ACM SIGGRAPH '97,* pp. 199-208, 1997.

[14] M. Olson and H. Zhang, "Silhouette Extraction in Hough Space," *Proc. Eurographics '06,* pp. 273-282, 2006.

[15] R. Pajarola, "FastMesh: Efficient View-Dependent Meshing," *Proc. Ninth Pacific Conf. Computer Graphics and Applications (PG '01),* pp. 22-30, 2001.

[16] M. Pop, C. Duncan, G. Barequet, M. Goodrich, W. Huang, and S. Kumar, "Efficient Perspective-Accurate Silhouette Computation and Applications," *Proc. 17th Ann. Symp. Computational Geometry (SCG '01),* pp. 60-68, 2001.

[17] P.V. Sander, X. Gu, S.J. Gortler, H. Hoppe, and J. Snyder, "Silhouette Clipping," *Proc. ACM SIGGRAPH '00,* pp. 327-334, 2000.

[18] G. Thürmer and C.A. Wüthrich, "Computing Vertex Normals from Polygonal Facets," *J. Graphics Tools,* vol. 3, no. 1, pp. 43-46, 1998.

[19] A. Vlachos, J. Peters, C. Boyd, and J. Mitchell, "Curved PN Triangles," *Proc. Symp. Interactive 3D Graphics (I3D '01)*, pp. 159-166, 2001.
[20] B. Wang, W. Wang, J. Wu, and J. Sun, "Silhouette Smoothing by Boundary Curve Interpolation," *Proc. Eighth Int'l Conf. Computer-Aided-Design and Computer Graphics (CAD/Graphics '03)*, pp. 197-202, Oct. 2003.
[21] J. Xia and A. Varshney, "Dynamic View-Dependent Simplification for Polygonal Models," *Proc. IEEE Conf. Visualization (VIS '96)*, pp. 327-334, 1996.
[22] H. Xu, M.X. Nguyen, X. Yuan, and B. Chen, "Interactive Silhouette Rendering for Point-Based Models," *Proc. Eurographics Symp. Point-Based Graphics*, pp. 13-18, 2004.

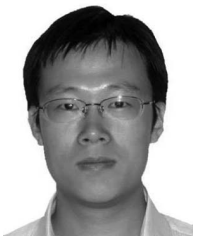
Lu Wang received the BSc degree from the Department of Computer Science, Shandong Normal University, in 2003. She is currently pursuing the PhD degree in the School of Computer Science and Technology, Shandong University, Jinan, P.R. China. Her research interests include computer graphics, real-time rendering, nonphotorealistic rendering, and geometric modeling.

Changhe Tu received the BSc, MEng, and PhD degrees from Shandong University, Jinan, P.R. China, in 1990, 1993, and 2003, respectively. He is now a professor in the School of Computer Science and Technology, Shandong University. His research interests include computer graphics, geometric modeling and processing, nonphotorealistic rendering, and virtual reality.

Wenping Wang received the BSc and MEng degrees in computer science from Shandong University in 1983 and 1986, respectively, and the PhD degree in computer science from the University of Alberta in 1992. He is an associate professor of computer science at the University of Hong Kong (HKU). His research covers computer graphics, geometric computing, and visualization. He has published more than 100 papers in these fields. He is an associate editor of the Springer journal *Computer Aided Geometric Design* and has been the program chair of several international conferences, including Geometric Modeling and Processing (GMP '00), the Pacific Conference Computer Graphics and Applications (PG '00 and PG '03), the ACM Symposium Virtual Reality Software and Technology (VRST '01), and the ACM Symposium on Physical and Solid Modeling (SPM '06). He received the Teaching Excellence Award of the Department of Computer Science at HKU in 2006 and the HKU Research Output Prize in 2007.

Xiangxu Meng received the BSc and MEng degrees from the Department of Computer Science, Shandong University, Jinan, P.R. China in 1982 and 1985, respectively, and the PhD degree from the Institute of Computing Technology, Chinese Academy of Sciences, in 1998. He is a professor in the School of Computer Science and Technology, Shandong University. His current research interests include human-computer interaction, virtual reality, computer graphics, CAD/CAM/CIMS, grid computing, visualization, and scientific computing.

Bin Chan received the BEng, MPhil, and PhD degrees from the University of Hong Kong in 1995, 1998, and 2005, respectively. He is now a research associate in the Department of Computer Science, University of Hong Kong. His research interests include real-time rendering, virtual reality, visualization, and global illumination.



Dongming Yan received the BSc and master's degrees in computer science from Tsinghua University, P.R. China, in 2002 and 2005, respectively. He is a PhD student in the Department of Computer Science, University of Hong Kong. His current research interests include geometric modeling, computer-aided design, and real-time rendering.