International Journal of CAD/CAM Vol. 6, No. 1, pp. 139~148 (2006)

INTERNATIONAL JOURNAL OF CAD/CAM
www.ijcc.org

Inscribed Approximation based Adaptive Tessellation of Catmull-Clark Subdivision Surfaces

Shuhua Lai and Fuhua (Frank) Cheng*

Graphics & Geometric Modeling Lab. Department of Computer Science, University of Kentucky, Lexington, Kentucky 40506-0046

Abstract – Catmull-Clark subdivision scheme provides a powerful method for building smooth and complex surfaces. But the number of faces in the uniformly refined meshes increases exponentially with respect to subdivision depth. Adaptive tessellation reduces the number of faces needed to yield a smooth approximation to the limit surface and, consequently, makes the rendering process more efficient. In this paper, we present a new adaptive tessellation method for general Catmull-Clark subdivision surfaces. Different from previous control mesh refinement based approaches, which generate approximate meshes that usually do not interpolate the limit surface, the new method is based on direct evaluation of the limit surface to generate an inscribed polyhedron of the limit surface. With explicit evaluation of general Catmull-Clark subdivision surfaces becoming available, the new adaptive tessellation method can precisely measure error for every point of the limit surface. Hence, it has complete control of the accuracy of the tessellation result. Cracks are avoided by using a recursive color marking process to ensure that adjacent patches or subpatches use the same limit surface points in the construction of the shared boundary. The new method performs limit surface evaluation only at points that are needed for the final rendering process. Therefore it is very fast and memory efficient. The new method is presented for the general Catmull-Clark subdivision scheme. But it can be used for any subdivision scheme that has an explicit evaluation method for its limit surface.

KeyWords: **subdivision, Catmull-Clark surfaces, adaptive tessellation, surface evaluation**

1. Introduction

Subdivision surfaces have become popular recently in graphical modeling and animation because of their capability in modeling/representing complex shape of arbitrary topology [2], their relatively high visual quality, and their stability and efficiency in numerical computation. Subdivision surfaces can model/represent complex shape of arbitrary topology because there is no limit on the shape and topology of the control mesh of a subdivision surface.

With the parametrization technique for subdivision surfaces becoming available [4] and with the fact that non-uniform B-spline and NURBS surfaces are special cases of subdivision surfaces becoming known [16], we now know that subdivision surfaces cover both *parametric forms* and *discrete forms*. Parametric forms are good for design and representation, discrete forms are good for machining and tessellation [1]. Hence, we have a representation scheme that is good for all graphics and CAD/CAM applications.

Subdivision based evaluation process of a subdivision surface relies on performing repeated subdivision of the control mesh until the refined mesh is close enough to the limit surface (within some given tolerance). It is then possible to push the control points (mesh vertices) to their limit positions. But the number of faces in the uniformly refined meshes increases exponentially with the recursive steps of subdivision. See Fig. 5(a) for an example where the control mesh of a Gargoyle is uniformly subdivided only twice and yet the resulting mesh is already quite dense. Hence, a good method for reducing the number of faces in the refined mesh while keeping the precision of the approximation is necessary. For instance, in Figs. 5(b), 5(c), and 5(d), the same model is adaptively subdivided 4, 3 and 2 times, respectively. The resulting meshes have a higher or similar precision while the number of facets in the resulting meshes is much less than the uniform case. Such a method is important for both rendering and finite-element mesh generation. The criterion for rendering, however, is different from the criterion for finite-element mesh generation. In the first case, the number of sides of the mesh faces could be different while, in the second case, the mesh faces are either all triangles or all quadrilaterals. Fig. 5(e) shows a triangulated result of Fig. 5(d).

Research work for reducing the number of faces in a mesh has been done in several directions. *Mesh simplification* [8] is the most popular one over the past decade. It aims at removing some of the overly sampled vertices in a mesh and produces approximate meshes with various levels of detail. Another main method for reducing the number of faces in a mesh, called *adaptive tessellation*, is to apply adaptive or local refinement schemes to areas specified by a user or determined by an application. The resulting mesh should be crack-free and have the same limit surface as the uniformly refined mesh.

*Corresponding author:
Tel: +1-859-257-6760
Fax: +-1-859-323-1971
E-mail: cheng@cs.uky.edu

There are two possible approaches for adaptive tessellation of subdivision surfaces. One is a *mesh refinement based* approach. It approximates the limit surface by adaptively refining the control mesh of the surface. The resulting mesh usually does not interpolate points of the limit surface. The other one is a *surface evaluation based* approach. This approach approximates the limit surface by generating an inscribing polyhedron of the limit surface, with vertices of the polyhedron taken (evaluated) adaptively from the limit surface. The mesh refinement based approach needs a subdivision scheme, such as the Catmull-Clark method or the Doo-Sabin method, to refine the input mesh. Most methods proposed in the literature for adaptive tessellation of subdivision surfaces belong to this category. The second approach needs a parametrization/evaluation method for the limit surface. With the availability of direct evaluation methods of subdivision surfaces recently [4-7], the second approach could be more appealing for adaptive tessellation of subdivision surface because of its simplicity in nature. Currently there is only one paper published in this category [11]. This paper works for parametrization that reproduces linear functions [19].

In this paper we will present a surface evaluation based approach for adaptive tessellation of subdivision surfaces. Our method is different from [11] in that our method works with any parametrization method and has a precise error estimate. The new approach is presented for the general Catmull-Clark subdivision surfaces [2], but it can be easily extended to work for any subdivision surface that has an exact evaluation method for its limit surface.

2. Previous Work

2.1. Catmull-Clark Subdivision Surfaces

Given a control mesh, a *Catmull-Clark subdivision surface* (CCSS) is generated by iteratively refining (subdividing) the control mesh [2] to form new control meshes. The subdividing process consists of defining new vertices (*face points*, *edge points* and *vertex points*) and connecting the new vertices to form new edges and faces of a new control mesh. A CCSS is the limit surface of a sequence of refined control meshes. The limit surface is called a *subdivision surface* because the mesh refining process is a generalization of the uniform B-spline surface *subdivision technique*. The *valence* of a mesh vertex is the number of mesh edges adjacent to the vertex. A mesh vertex is called an *extra-ordinary vertex* if its valence is different from four. A mesh face with an extra-ordinary vertex is called an *extra-ordinary face*. The *valance* of an extra-ordinary face is the valence of its extra-ordinary vertex. Given an extra-ordinary face, if the valence of its extra-ordinary vertex is n, then the surface patch corresponding to this extra-ordinary face is influenced by $2n+8$ control vertices. Recent work [4, 5, 6, 7] shows that any point in the limit surface of a CCSS can be exactly and directly evaluated from its $2n+8$ control points. Hence control mesh subdivision is not absolutely necessary for the rendering of a CCSS.

2.2. Adaptive Tessellation

A number of adaptive tessellation methods for subdivision surfaces have been proposed [3,9,10,11, 14,15]. Most of them are *mesh refinement based*, i.e., approximating the limit surface by adaptively refining the control mesh. This approach requires the assignment of a subdivision depth to each region of the surface first. In [3], a subdivision depth is calculated for each patch of the given Catmull-Clark surface with respect to a given error tolerance ε. In [9], a subdivision depth is estimated for each vertex of the given Catmull-Clark surface by considering factors such as curvature, visibility, membership to the silhouette, and projected size of the patch. The approach used in [3] is error controllable. An error controllable approach for Loop surface is proposed in [11], which calculates a subdivision depth for each patch of a Loop surface by estimating the distance between two bounding linear functions for each component of the 3D representation.

Several other adaptive tessellation schemes have been presented as well [15,14,10]. In [10], two methods of adaptive tessellation for triangular meshes are proposed. The adaptive tessellation process for each patch is based on angles between its normal and normals of adjacent faces. A set of new error metrics tailored to the particular needs of surfaces with sharp creases is introduced in [14].

In addition to various adaptive tessellation schemes, there are also applications of these techniques. D. Rose *et al.* used adaptive tessellation method to render terrain [18] and K. Muller et al. combined ray tracing with adaptive subdivision surfaces to generate some realistic scenes [13]. Adaptive tessellation is such an important technique that an API has been designed for its general usage [17]. Actually hardware implementation of this technique has been reported recently as well [12].

A problem with the mesh-refinement-based, adaptive tessellation techniques is the so called gap-prevention requirement. Because the number of new vertices generated on each boundary of the control mesh depends on the subdivision depth, gaps (or, cracks) could occur between the control meshes of adjacent patches if these patches are assigned different subdivision depths. Hence, each mesh-refinement-based adaptive tessellation method needs some special mechanism to eliminate gaps. This is usually done by performing additional subdivision or splitting steps on the patch with lower subdivision depth. As a result, many unnecessary polygons are generated in the tessellation process. In this paper, we will adaptively tessellate a subdivision surface by taking points from the limit surface to form an inscribing polyhedron of the limit surface, instead of refining the control mesh.

Our method simplifies the process of gap detecting and elimination. It does not need to perform extra or unnecessary evaluations either.

2.3. Evaluation of a CCSS Patch

Several approaches [4,5,6,7] have been presented for exact evaluation of an extraordinary patch at any parameter point (u,v). In this paper, we will follow the parametrization technique presented in [7], because this method is numerically stable, employs less eigen basis functions, and can be used for the evaluation of 3D position and normal vector of any point in the limit surface exactly and explicitly. Some most related results of [7] are briefly summarized below.

Every point in the parameter space of a regular or extra-ordinary patch can be exactly and explicitly evaluated as follows:

$$S(u,v)=W^T K^m \sum_{j=0}^{n+5} \lambda_j^{m-1} M_{b,j} G \tag{1}$$

where n is the valance of the extraordinary patch, W is a vector containing the *16* B-spline power basis functions. $\lambda_j (0 \le j \le n+5)$ are eigenvalues of the Catmull-Clark subdivision matrix and their values can be found in [7]. m and b can be exactly evaluated from u and v [7]. K and $M_{b,j}$ are constant matrices and their values can be found in [7]. G is the vector of control points (See [7] for their labeling).

One can compute the derivatives of $S(u,v)$ to any order simply by differentiating $W(u,v)$ in Eq. (1) accordingly. For example,

$$\frac{\partial}{\partial u} S(u,v)=\left(\frac{\partial W}{\partial u}\right)^T K^m \sum_{j=0}^{n+5} \lambda_j^{m-1} M_{b,j} G \tag{2}$$

With the explicit expression of $S(u,v)$ and its partial derivatives, one can easily get the limit point of an extraordinary vertex in a general Catmull Clark subdivision surface:

$$S(0,0)=[1,0,0,0,\Lambda,0]\cdot M_{b,n+1}\cdot G \tag{3}$$

and the first derivatives:

$$D_u=[0,1,0,0,\Lambda,0]\cdot M_{b,2}\cdot G$$

$$D_u=[0,0,1,0,\Lambda,0]\cdot M_{b,2}\cdot G$$

where D_u and D_v are the direction vectors of $\frac{\partial S(0,0)}{\partial u}$ and $\frac{\partial S(0,0)}{\partial v}$, respectively. The normal vector at (0,0) is the cross product of D_u and D_v.

3. Basic Idea

3.1. Inscribed Approximation

One way to approximate a curve (surface) is to use its control polygon (mesh) as the approximating polyline (polyhedron). For instance, in Fig. 1(a), at the top are a cubic Bezier curve and its control polygon. For a better approximation, we can refine the control polygon using midpoint subdivision. The solid polyline at the bottom of Fig. 3(a) is the approximating control polygon after one refinement. This method relies on performing iterative refinement of the control polygon or control mesh to approximate the limit curve or surface. Because this method approximates the limit shape from control polygon or control mesh "outside" the limit shape, we call this method *circumscribed approximation*.

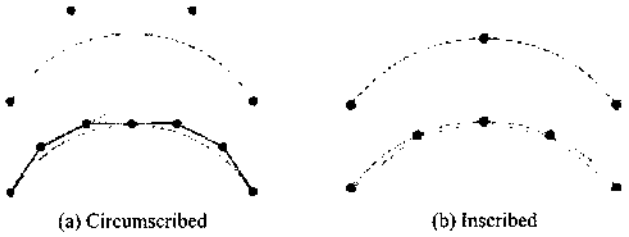


Fig. 1. Inscribed and Circumscribed Approximation.

Another possible method is *inscribed approximation*. Instead of approximating the limit curve (surface) by performing subdivision on its control polygon (mesh), one can approximate the limit curve (surface) by inscribed polygons (polyhedra) whose vertices are taken from the limit curve (surface) directly. The easiest approach to get vertices of the inscribed polygons (polyhedra) is to perform uniform midpoint subdivision on the parameter space and use the evaluated vertices of the resulting subsegments (subpatches) as vertices of the inscribed polylines (polyhedra). For instance, in Fig. 3(b), at the top are a cubic Bezier curve and its approximating polygon with vertices evaluated at parameter points 0, 1/2 and 1. Similarly, the solid polygon at the bottom of Fig. 3(b) is an approximating polygon with vertices evaluated at five parameter points.

Because inscribed approximation uses points directly located on the limit curve or surface, in most cases, it has faster convergent rate than the circumscribed approximation. As one can see clearly from Fig. 3 that the inscribing polygon at the bottom of Fig. 3(b) is closer to the limit curve than the circumscribing polygon shown at the bottom of Fig. 3(a) even though the inscribing polygon actually has less segments than the circumscribing polygon.

However, the problem with both approaches is that, with uniform subdivision, no matter it is performed on the control mesh or the parameter space, one would get unnecessarily small and dense polygons for surface patches that are already flat enough and, consequently, slow down the rendering process. To speed up the rendering process, a flat surface patch should not be tessellated as densely as a surface patch with big curvature. The adaptive tessellation process of a surface patch should be performed based on the flatness of the patch. This leads to our adaptive inscribed approximation.

3.2. Adaptive Inscribed Approximation

For a patch of $S(u,v)$ defined on $u_1 \le u \le u_2$ and $v_1 \le v \le v_2$, we try to approximate it with the quadrilateral formed by its four vertices $V_1 = S(u_1,v_1)$, $V_2 = S(u_2,v_1)$, $V_3 = S(u_2,v_2)$ and $V_4 = S(u_1,v_2)$. If the distance (to be defined below) between the patch and its corresponding quadrilateral is small enough (to be defined below), then the patch is considered *flat* enough and will be (for now) replaced with the corresponding quadrilateral in the tessellation process. Otherwise, we perform a midpoint subdivision on the parameter space by setting $u_{12} = (u_1 + u_2)/2$ and $v_{12} = (v_1 + v_2)/2$ to get four subpatches: $S([u_1, u_{12}] \times [v_1, v_{12}])$, $S([u_{12}, u_2] \times [v_1, v_{12}])$, $S([u_{12}, u_2] \times [v_{12}, v_2])$, $S([u_1, u_{12}] \times [v_{12}, v_2])$ and repeat the *flatness* testing process on each of the subpatches. The process is recursively repeated until the distance between all the subpatches and their corresponding quadrilaterals are small enough. The vertices of the resulting subpatches are then used as vertices of the inscribed polyhedron of the limit surface. For instance, if the four rectangles in Fig. 2(a) are the parameter spaces of four adjacent patches of $S(u,v)$, and if the rectangles shown in Fig. 2(b) are the parameter spaces of the resulting subpatches when the above flatness testing process stops, then the limit surface will be evaluated at the points marked with small solid circles to form vertices of the inscribing polyhedron of the limit surface.

In the above flatness testing process, to measure the difference between a patch (or subpatch) and its corresponding quadrilateral, we need to parametrize the quadrilateral as well. The quadrilateral can be parametrized as follows:

$$Q(u,v) = \frac{v_2 - v}{v_2 - v_1}\left(\frac{u_2 - u}{u_2 - u_1}V_1 + \frac{u - u_1}{u_2 - u_1}V_2\right) + \frac{v - v_1}{v_2 - v_1}\left(\frac{u_2 - u}{u_2 - u_1}V_4 + \frac{u - u_1}{u_2 - u_1}V_3\right) \tag{4}$$

where $u_1 \le u \le u_2$, $v_1 \le v \le v_2$. The *difference* between the patch (or subpatch) and the corresponding quadrilateral at (u,v) is defined as

$$d(u,v) = \|Q(u,v) - S(u,v)\|^2 = (Q(u,v) - S(u,v)) \cdot (Q(u,v) - S(u,v))^T \tag{5}$$

where $\|\bullet\|$ is the second norm and A^T is the transpose of A. The *distance* between the patch (or subpatch) and the corresponding quadrilateral is the maximum of all the differences:

$$D = \max\{\sqrt{d(u,v)} \mid (u,v) \in [u_1,u_2] \times [v_1,v_2]\}$$

To measure the distance between a patch (or subpatch) and the corresponding quadrilateral, we only need to measure the norms of all local minima and maxima of $d(u,v)$. Note that $Q(u,v)$ and $S(u,v)$ are both C^1-continuous, and $d(V_1)$, $d(V_2)$, $d(V_3)$ and $d(V_4)$ are equal to 0. Therefore, by Mean Value Theorem, the local minima and maxima must lie either inside $[u_1,u_2] \times [v_1,v_2]$ or on the four boundary curves. In other words, they must satisfy at least one of the following three conditions:

$$\begin{cases} \dfrac{\partial d(u,v)}{\partial u} = 0 \\ v = v_1 \ \text{ or } \ v = v_2 \\ u_1 \le u \le u_2 \end{cases} \quad \begin{cases} \dfrac{\partial d(u,v)}{\partial v} = 0 \\ u = u_1 \ \text{ or } \ u = u_2 \\ v_1 \le v \le v_2 \end{cases} \quad \begin{cases} \dfrac{\partial d(u,v)}{\partial u} = 0 \\ \dfrac{\partial d(u,v)}{\partial v} = 0 \\ (u,v) \in (u_1,u_2) \times (v_1,v_2) \end{cases} \tag{6}$$

For a patch (or subpatch) that is not adjacent to an extraordinary point (i.e., $(u_1,v_1) \ != (0,0)$), m is fixed and known [7]. Hence Eq. (6) can be solved explicitly. With the valid solutions, we can find the difference for each of them using Eq. (5). Suppose the one with the biggest difference is $(\hat{u},\hat{v})$. Then $(\hat{u},\hat{v})$ is also the point with the biggest distance between the patch (or subpatch) and its corresponding quadrilateral. We consider the patch (or subpatch) to be *flat* enough if

$$D = \sqrt{d(\hat{u},\hat{v})} \le \varepsilon \tag{7}$$

where ε is a given error tolerance. In such a case, the patch (or subpatch) is replaced with the corresponding quadrilateral in the tessellation process. If a patch (or subpatch) is not flat enough yet, i.e., if Eq. (7) does not hold, we perform a midpoint subdivision on the patch (or subpatch) to get four new subpatches and repeat the flatness testing process for each of the new subpatches. This process is recursively repeated until all the subpatches satisfy Eq. (7).

For a patch (or subpatch) that is adjacent to an extraordinary point (i.e. $(u_1,v_1) = (0,0)$ in Eq. (6)), m is not fixed and m tends to infinity (see Fig. 3). As a result, Eq. (6) can not be solved explicitly. One way to resolve this problem is to use nonlinear numerical method to solve these equations. But numerical approach cannot guarantee the error is less than ε everywhere. For precise error control, a better choice is needed. In the following, an alternative method is given for that purpose.

Eq. (3) shows that $S(u,v)$ and $Q(u,v)$ both converge to $S(0,0)$ when $(u,v) \to (0,0)$. Hence, for any given error tolerance ε, there exists an integer m_ε such that if $m \ge m_\varepsilon$, then the distance between $S(u,v)$ and $S(0,0)$ is smaller than $\varepsilon/2$ for any $(u,v) \in [0,1/2^m] \times [0,1/2^m]$, and so is the distance between $Q(u,v)$ and $S(0,0)$. Consequently,

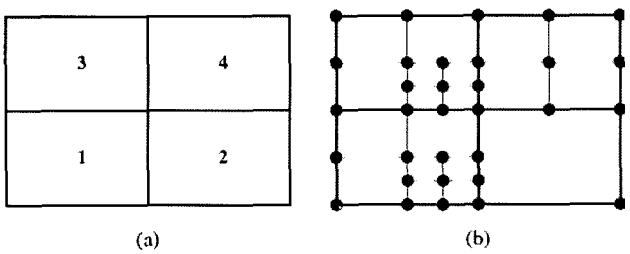


Fig. 2. Basic idea of the construction of an inscribed polyhedron.

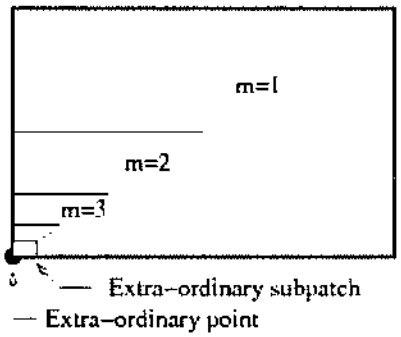


Fig. 3. Partitioning of the unit square [7].

when $(u,v) \in [0,1/2^m] \times [0,1/2^m]$, the distance between $\boldsymbol{S}(u,v)$ and $\boldsymbol{Q}(u,v)$ is smaller than ε. The value of m_ε, in most of the cases, is a relatively small number and can be explicitly calculated. In next subsection, we will show how to calculate m_ε.

For other regions of the unit square with $[\log_{1/2}u_2] \le m < m_\varepsilon$ (see Fig. 3), eq. (6) can be used directly to find the difference between $\boldsymbol{S}(u,v)$ and $\boldsymbol{Q}(u,v)$ for any fixed $m \in \min([\log_{1/2}u_2], m_\varepsilon)$. Therefore, by combining all these differences, we have the distance between the given extra-ordinary patch (or subpatch) and the corresponding quadrilateral. If this distance is smaller than ε, we consider the given extra-ordinary patch (or subpatch) to be flat, and use the corresponding quadrilateral to replace the extra-ordinary patch (or subpatch) in the tessellation process. Otherwise, repeatedly subdivide the patch (or subpatch) and perform flatness testing on the resulting subpatches until all the subpatches satisfy Eq. (7).

3.3 Calculating m_ε

For a given $\varepsilon > 0$, an integer k_ε will first be computed so that if k is bigger than k_ε, then the subpatch of $\boldsymbol{S}(u,v)$ with $0 \le u, v \le 1/2^k$ is contained in a sphere with center $\boldsymbol{S}(0,0)$ and diameter ε (called an ε-sphere). A subpatch is contained in an ε-sphere if all points of the subpatch are $\varepsilon/2$ away from $\boldsymbol{S}(0,0)$.

To find such k_ε, we need a few properties from [7]. Recall that an extra-ordinary patch $\boldsymbol{S}(u,v)$ can be expressed as $S(u,v) = \sum_{j=0}^{n+5} \Phi_{b,j}(u,v) \cdot G$ where $\Phi_{b,j}$ are *eigen basis functions* defined in [7] and G is the vector of control points of $\boldsymbol{S}$. The eigen basis functions satisfy the *scaling relation* [4, 7], i.e., $\Phi_{b,j}(u/2^k, v/2^k) = \lambda_j^k \Phi_{b,j}(u,v)$ for any positive integer k, where λ_j are eigen values of the Catmull-Clark subdivision matrix [7]. The eigen values are indexed so that $1 = \lambda_{n+1} > \lambda_2 \ge \lambda_i > 0$, where $0 \le i \le n+5$ and $i \ne n+1$. Also recall that $\Phi_{b,j}(0,0) = 0$ when $j \ne n+1$, and $\Phi_{b,n+1}(u,v)$ is a constant vector, its value is independent of (u,v) [7]. Hence, $(\Phi_{b,n+1}(u,v) - \Phi_{b,n+1}(u',v')) \cdot G_r = 0$ for any (u,v) and (u',v') where $r \in \{x,y,z\}$ and $\boldsymbol{G}_r$ is the x-, y- or z-component of $\boldsymbol{G}$.

Hence for any $1/2 \le u \le 1$ or $1/2 \le v \le 1$, and for any k we have

$$|S_r(u/2^k, v/2^k) - S_r(0,0)| = \left|\sum_{j=0}^{n+5} (\lambda_j^k \Phi_{b,j}(u,v) - \Phi_{b,j}(0,0)) \cdot G_r\right|$$
$$\le \sum_{j \ne n+1} \lambda_j^k |\Phi_{b,j}(u,v) \cdot G_r| < \lambda_2^k \sum_{j \ne n+1} |\Phi_{b,j}(u,v) \cdot G_r|$$

Similarly, the three conditions in Eqs. (6) can be used to find the maxima of $\Phi_{b,j}(u,v) \cdot G_r$ for any j. Note that because here $(u,v) \notin [0,1/2] \times [0,1/2]$, the corresponding m is equal to 1 (See figure 3). Hence we can easily find the maximum in its domain $\{(u,v) | 1/2 \le u \le 1$ or $1/2 \le v \le 1\}$. Let the maximum of $\Phi_{b,j}(u,v) \cdot G_r$ be F_{rj} and $F_r = \Sigma_{j \ne n+1} F_{rj}$. Then for any $k > 0$ we have $|S_r(u/2^k, v2^k) - S_r(0,0)| \le \lambda_2^k F_r$. Therefore if $(\lambda_2^k F_x)^2 + (\lambda_2^k F_y)^2 + (\lambda_2^k F_z)^2 \le (\varepsilon/2)^2$, we have $\|S(u/2^k, v/2^k) - S(0,0)\| \le \varepsilon/2$. If we define k_ε as follows: $k_\varepsilon = \left\lceil \log_{\lambda_2} \frac{\varepsilon}{2\sqrt{F_x^2 + F_y^2 + F_z^2}} \right\rceil$, then it is easy to see that when $k \ge k_\varepsilon$, the subpatch $\boldsymbol{S}(u,v)$ with $(u,v) \notin [0,1/2^k] \times [0,1/2^k]$ is inside an ε-sphere whose center is $\boldsymbol{S}(0,0)$.

In addition, $\boldsymbol{S}(0,0)$ is a fixed point and has an explicit expression for any patch (see eq. 3), and $\boldsymbol{Q}(u,v)$ also has an explicit parametrization (See eq. (4)). Hence, similarly, by using the method of Eqs. (6), it is easy to find an integer k_ε, such that for any given $\varepsilon > 0$, when $k \ge k_\varepsilon$, we have $\|\boldsymbol{Q}(u,v) - \boldsymbol{S}(0,0)\| \le \varepsilon/2$, where $(u,v) \in [0,1/2^k] \times [0,1/2^k]$. Once we have k_ε and k_ε, simply set m_ε as the maximum of k_ε and k_ε. With this m_ε, it is easy to see that when $m \ge m_\varepsilon$, we have $\|\boldsymbol{S}(u,v) - \boldsymbol{Q}(u,v)\| \le \varepsilon$, where $(u,v) \in [0,1/2^k] \times [0,1/2^k]$

4. Crack Elimination

Due to the fact that adjacent patches might be approximated by quadrilaterals corresponding to subpatches from different levels of the midpoint subdivision process, cracks could occur between adjacent patches. For instance, in Fig. 4, the left patch $\boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{A}_5\boldsymbol{A}_6$ is approximated by one quadrilateral but the right patch is approximated by 7 quadrilaterals. Consider the boundary shared by the left patch and the right patch. On the left side, that boundary is a line segment defined by two vertices : $\boldsymbol{A}_2$ and $\boldsymbol{A}_5$. But on the right side, the boundary is a polyline defined by four vertices : $\boldsymbol{A}_2$, $\boldsymbol{C}_4$, $\boldsymbol{B}_4$, and $\boldsymbol{A}_5$. They would not coincide unless $\boldsymbol{C}_4$ and $\boldsymbol{B}_4$ lie on the line segment defined by $\boldsymbol{A}_2$ and $\boldsymbol{A}_5$. But that usually is not the case. Hence, cracks would appear between the left patch and the right patch.

Fortunately Cracks can be eliminated simply by replacing each boundary of a patch or subpatch with the one that contains all the evaluated points for that boundary. For example, in Fig. 4, all the dashed lines should be replaced with the corresponding polylines. In particular, boundary $\boldsymbol{A}_2\boldsymbol{A}_5$ of patch $\boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{A}_5\boldsymbol{A}_6$ should be replaced with the polyline $\boldsymbol{A}_2\boldsymbol{C}_4\boldsymbol{B}_4\boldsymbol{A}_5$. As a result, polygon $\boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{A}_5\boldsymbol{A}_6$ is replaced with polygon $\boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{C}_4\boldsymbol{B}_4\boldsymbol{A}_5\boldsymbol{A}_6$ in the

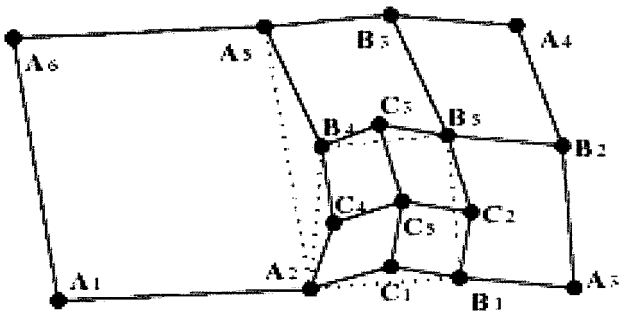


Fig. 4. Crack elimination.

tessellation process. For rendering purpose this is fine because graphics systems like OpenGL can handle polygons with non-co-planar vertices and polygons with any number of sides. However, it should be pointed out that through a simple zigzag technique, triangulation of those polygons is actually a simple and very fast process.

A potential problem with this process is the new polygons generated by the crack elimination algorithm might not satisfy the flatness requirement. To ensure the flatness requirement is satisfied everywhere when the above crack elimination method is used, we need to change the test condition in Eq. (7) to the following one:

$$\sqrt{d(\bar{u},\bar{v})}+\sqrt{d(\hat{u},\hat{v})}\le\varepsilon \tag{8}$$

where $(\hat{u},\hat{v})$ and $(\bar{u},\bar{v})$ are solutions of Eq. (6) and they satisfy the following conditions:

- Among all the solutions of Eq. (6) that are located on one side of $\boldsymbol{Q}(u,v)$, i.e. solutions that satisfy $\boldsymbol{Q}(u,v)\ge 0$, $d(\bar{u},\bar{v})$ is the biggest. If there does not exist any solution such that $\boldsymbol{Q}(u,v)-\boldsymbol{S}(u,v))\cdot((V_1-V_3)\times(V_2-V_4))\ge 0$, then $d(\hat{u},\hat{v})$ is set to 0;
- Among all the solutions of Eq. (6) that are located on the other side of $\boldsymbol{Q}(u,v)$, i.e. solutions that satisfy $\boldsymbol{Q}(u,v)<0$, $d(\hat{u},\hat{v})$ is the biggest. If there does not exist any solution such that $\boldsymbol{Q}(u,v)-\boldsymbol{S}(u,v))\cdot((V_1-V_3)\times(V_2-V_4))<0$, then $d(\bar{u},\bar{v})$ is set to 0.

From the definition of $(\hat{u},\hat{v})$ and $(\bar{u},\bar{v})$, we can see that satisfying Eq. (8) means that the patch being tested is located between two quadrilaterals that are ε away.

Note that all the evaluated points lie on the limit surface. Hence, for instance, in Fig. 4, points $\boldsymbol{A}_2$, $\boldsymbol{C}_4$, $\boldsymbol{B}_4$ and $\boldsymbol{A}_5$ of patch $\boldsymbol{A}_2\boldsymbol{A}_3\boldsymbol{A}_4\boldsymbol{A}_5$ are also points of patch $\boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{A}_5\boldsymbol{A}_6$. With the new test condition in Eq. (8), we know that a patch or subpatch is flat enough if it is located between two quadrilaterals that are ε away. Because boundary points $\boldsymbol{A}_2$, $\boldsymbol{C}_4$, $\boldsymbol{B}_4$ and $\boldsymbol{A}_5$ are on the limit surface, they must be located between two quadrilaterals that are ε away. So is the polygon $\boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{C}_4\boldsymbol{B}_4\boldsymbol{A}_5\boldsymbol{A}_6$. Now the patch (or subpatch) and its approximating polygon are both located inside two quadrilaterals that are ε away. Hence the overall error between the patch (or subpatch) and its approximating polygon is guaranteed to be smaller than ε.

In previous methods for adaptive tessellation of subdivision surfaces [3,9,10,14], the most difficult part is crack prevention. Yet in our method, this part becomes the simplest part to handle and implement. The resulting surface is error controllable and guaranteed to be crack free.

5. Degree of Flatness

Just like numerical errors have two different settings, the flatness of a patch, which can be viewed as a numerical error from the approximation point of view, has two different aspects as well, depending on if the flatness is considered in the absolute sense or relative sense. The flatness of a patch is called the *absolute flatness* (***AF***) if the patch is not transformed in any way. In that case, the value of ε in Eqs. (7) and (8) is set to whatever precision the flatness of the patch is supposed to meet. ***AF*** should be considered for operations that work on physical size of an object such as machining or prototyping.

For operations that do not work on the physical size of an object, such as the rendering process, we need a flatness that does not depends on the physical size of a patch. Such a flatness must be Affine transformation invariant and be a constant for any transformed version of the patch. Such a flatness is called the *relative flatness* of the patch. More specifically, if $\boldsymbol{Q}$ is the corresponding quadrilateral of patch $\boldsymbol{S}$, the relative flatness (***RF***) of $\boldsymbol{S}$ with respect to $\boldsymbol{Q}$ is defined as follows:

$$RF=\frac{d}{\max\{D_1,D_2\}}$$

where d is the maximal distance from $\boldsymbol{S}$ to $\boldsymbol{Q}$, and D_1, D_2 are lengths of the diagonal lines of $\boldsymbol{Q}$. It is easy to see that ***RF*** defined this way is Affine transformation invariant. Note that when D_1 and D_2 are fixed, smaller ***RF*** means smaller d. Hence, ***RF*** indeed measures the flatness of a patch. The difference between ***RF*** and ***AF*** is that ***RF*** measures the flatness of a patch in a global sense while ***AF*** measures flatness of a patch in a local sense. Therefore, ***RF*** is more suitable for operations that have data sets of various sizes but with a constant size display area such as the rendering process. Using ***RF*** is also good for adaptive tessellation process because it has the advantage of keeping the number of polygons low in the tessellation process.

6. Algorithms of Adaptive Tessellation

In this section, we discuss the important steps of the adaptive tessellation process and present the corresponding algorithms.

6.1. Global Index ID

Currently, all the subdivision surface parametrization and evaluation techniques are patch based [4,6,7].

Hence, no matter which method is used in the adaptive tessellation process, a patch cannot see vertices evaluated by other patches from its own (local) structure even though the vertices are on its own boundary. For example, in Fig. 4, vertices C_4 and B_4 are on the shared boundary of patches $A_1A_2A_5A_6$ and $A_2A_3A_4A_5$. But patch $A_1A_2A_5A_6$ can not see these vertices from its own structure because these vertices are not evaluated by this patch. To make adjacent patches visible to each other and to make subsequent crack elimination work easier, a *global index ID* is assigned to each evaluated vertex such that

- all the evaluated vertices with the same 3D position have the same index ***ID***;
- the index ***ID***'s are sorted in v and then in u, i.e., if $(u_i,v_i)(u_j,v_j)$, then ID_i ID_j, unless ID_i or ID_j has been used in previous patch evaluation.

With a global index ***ID***, it is easy to do crack prevention even with a patch based approach. Actually, subsequent processing can all be done with a patch based approach and still performed efficiently. For example, in Fig. 4, patch $A_1A_2A_5A_6$ can see both C_4 and B_4 even though they are not evaluated by this patch. In the subsequent tessellating process, the patch simply output all the marked vertices on its boundary that it can see to form a polygon for the tessellation purpose, i.e., $A_1A_2C_4B_4A_5A_6$.

6.2. Adaptive Marking

The purpose of *adaptive marking* is to mark those points in uv space where the limit surface should be evaluated. With the help of the global index ***ID***, this step can be done on an individual patch basis. Initially, all (u,v) points are marked white. If surface evaluation should be performed at a point and the resulting vertex is needed in the tessellation process, then that point is marked in black. This process can be easily implemented as a recursive function. The pseudo code for this step is given below.

AdaptiveMarking(P, u_1, u_2, v_1, v_2)
1. ***Evaluate***(P, u_1, u_2, v_1, v_2);
2. ***AssignGlobalID***(P, u_1, u_2, v_1, v_2);
3. if (***FlatEnough***(P, u_1, u_2, v_1, v_2)) ***MarkBlack***(P, u_1, u_2, v_1, v_2);
4. else $u_{12} = (u_1 + u_2)/2$; $v_{12} = (v_1 + v_2)/2$;
5. ***AdaptiveMarking***(P, u_1, u_{12}, v_1, v_{12});
6. ***AdaptiveMarking***(P, u_{12}, u_2, v_1, v_{12});
7. ***AdaptiveMarking***(P, u_{12}, u_2, v_{12}, v_2);
8. ***AdaptiveMarking***(P, u_1, u_{12}, v_{12}, v_2);

This routine adaptively marks points in the parameter space of patch P. Function 'Evaluate' evaluates limit surface at the four corners of patch or subpatch P defined on $[u_1,u_2] \times [v_1,v_2]$. Function 'FlatEnough' uses the method given in section 3 and Eq. (7) to tell if a patch or subpatch is flat enough. Function 'MarkBlack' marks the four corners of patch or subpatch P defined on $[u_1,u_2] \times [v_1,v_2]$ in black. All the marked corner points will be used in the tessellation process.

6.3. Adaptive Tessellation of a Single Patch

The purpose of this step is to tessellate the limit surface with as few polygons as possible, while preventing the occurrence of any cracks. Note that the limit surface will be evaluated only at the points marked in black, and the resulting vertices are the only vertices that will be used in the tessellation process. To avoid cracks, each marked points must be tessellated properly. Hence special care must be taken on adjacent patches or subpatches. With the help of adaptive marking, this process can easily be implemented as a recursive function as well. A pseudo code for this step is given below.

AdaptiveTessellation (P, u_1, u_2, v_1, v_2)
1. if (***NoMarkedPointInside***(P, u_1, u_2, v_1, v_2)) ***TessellatePolygon***(P, u_1, u_2, v_1, v_2);
2. else $u_{12} = (u_1 + u_2)/2$; $v_{12} = (v_1 + v_2)/2$;
3. ***AdaptiveTessellation*** (P, u_1, u_{12}, v_1, v_{12});
4. ***AdaptiveTessellation*** (P, u_{12}, u_2, v_1, v_{12});
5. ***AdaptiveTessellation*** (P, u_{12}, u_2, v_{12}, v_2);
6. ***AdaptiveTessellation*** (P, u_1, u_{12}, v_{12}, v_2);

This routine adaptively tessellates marked points in patch or subpatch P. Function 'NoMarkedPointInside' tests if none of the points inside $[u_1,u_2] \times [v_1,v_2]$, excluding the boundary points, are marked. If all the interior points are in white (i.e. not marked), it returns TRUE. Function 'TessellatePolygon' is defined as follows.

TessellatePolygon(P, u_1, u_2, v_1, v_2)
1. ***Begin***(TessellationModel);
2. Output all the marked points between:
3. $(u_1,v_1) \rightarrow (u_2,v_1)$;
4. $(u_2,v_1) \rightarrow (u_2,v_2)$;
5. $(u_2,v_2) \rightarrow (u_1,v_2)$;
6. $(u_1,v_2) \rightarrow (u_1,v_1)$;
7. ***End***();

6.4. Adaptive Tessellation of a CCSS

The overall algorithm for tessellating a general CCSS is given below. The algorithm takes the control mesh of the surface as input.

CCSSAdaptiveTessellation(Mesh M)
1. for each face P in M
2. ***AdaptiveMarking***(P, u_1, u_2, v_1, v_2);
3. for each face P in M
4. ***AdaptiveTessellation***(P, u_1, u_2, v_1, v_2);

7. Implementation and Test Results

The proposed approach has been implemented in *C++* using *OpenGL* as the supporting graphics system on the *Windows* platform. Quite a few examples have been tested with the method described here. All of these examples have extra-ordinary points in the given meshes. Some of the tested results are shown in Figs.

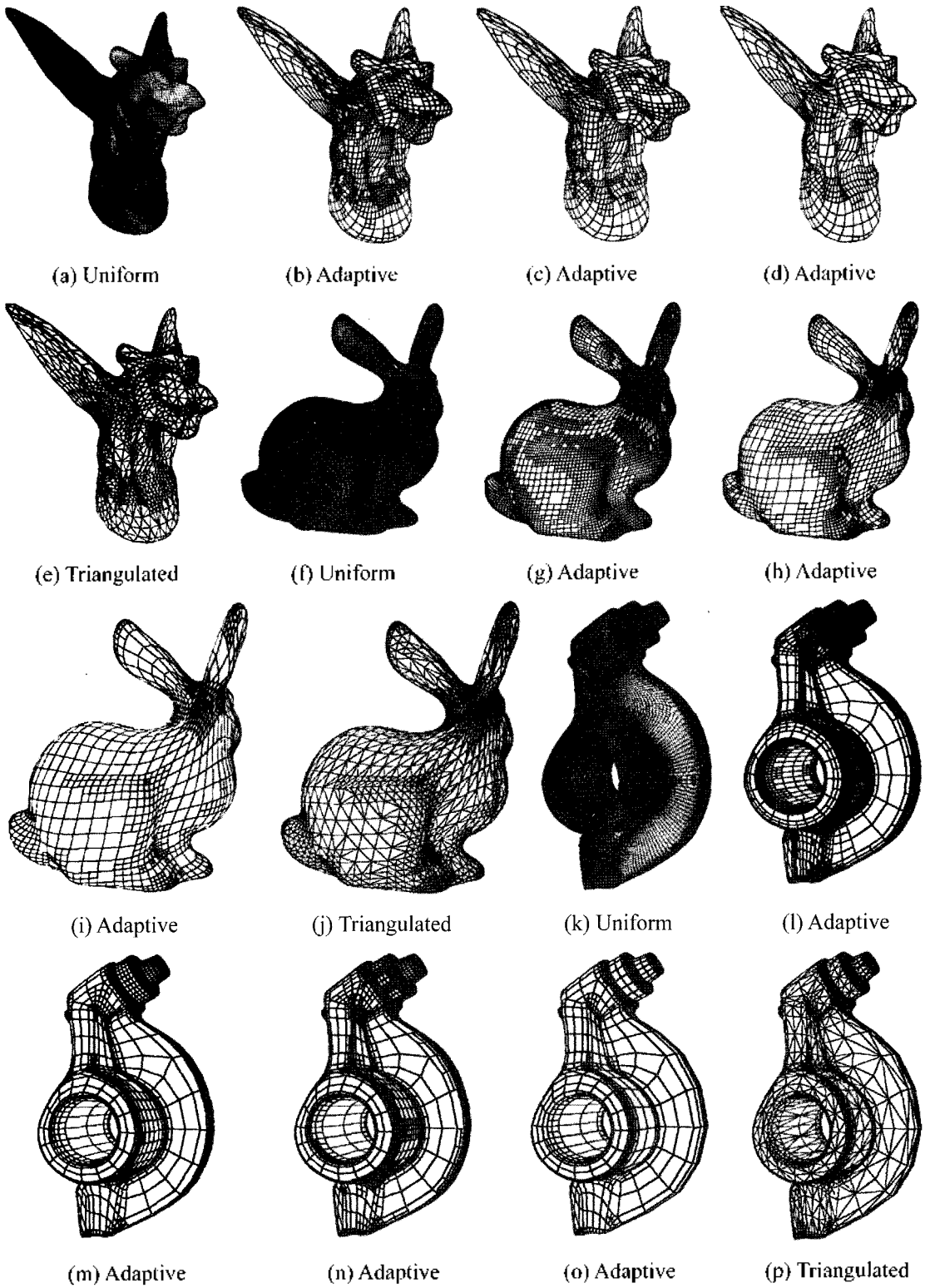


Fig. 5. Adaptive tessellation of surfaces with arbitrary topology.

5. For instance, Figs. 5(f) is generated using uniform subdivision, while Figs. 5(g), 5(h), 5(i) are tessellated with the adaptive technique presented in this paper, and Fig. 5(j) is the triangulated result of Fig. 5(i). Also Fig. 5(e) and Fig. 5(p) are the triangulated results of Fig. 5(d) and Fig. 5(o), respectively. From Fig. 5 we can see that all the adaptively tessellated CCSS's indeed significantly reduce the number of faces in the resulting tessellation while satisfying the given error requirement.

From our experiments, we also see that triangulated tessellations usually increases the number of polygons by at lease 2 times. Hence triangulation will slow down

the rendering process while it does not improve accuracy. From the view point of rendering, triangulation is not really necessary. But for some special applications, such as Finite Element Analysis, triangulation is indispensable. As mentioned above, performing triangulation on the resulting mesh of our adaptive tessellation process is straightforward and fast.

The proposed adaptive tessellation method is good for models that have large flat or nearly flat regions in its limit surface and would save significant amount of time in the final rendering process. One main disadvantage of all the current adaptive tessellation methods (including the method proposed here) is that they only eliminate polygons inside a single patch. They do not take the whole surface into consideration. For instance, all the flat sides of the rocker arm model in Fig. 5 are already flat enough, yet a lot of polygons are still generated there.

8. Summary

A surface-evaluation-based adaptive tessellation method for general Catmull-Clark subdivision surfaces is presented. The new method only evaluates those limit surface points that are needed in the final rendering process. On the other hand, while previous methods use a significant amount of effort to prevent the occurrence of cracks between adjacent patches, it takes almost no effort for the new method to eliminate cracks in the resulting inscribing polyhedron of the limit surface. Hence the new method is both computation efficient and memory efficient.

Acknowledgements

Research work of the authors is supported by NSF under grants DMS-0310645 and DMI-0422126. Data sets for Fig. 5 except the rocker arm are downloaded from the web site: http://research.microsoft.com/~hoppe/.

References

[1] Austin, S.P., Jerard, R.B. and Drysdale, R.L. (1997), "Comparison of discretization algorithms for NURBS surfaces with application to numerically controlled machining," *Computer Aided Design*, **29**(1), 71-83.

[2] Catmull, E. and Clark, J. (1978), "Recursively generated B-spline surfaces on arbitrary topological meshes," *Computer-Aided Design*, **10**(6), 350-355.

[3] Yong, J. and Cheng, F. (2005), "Adaptive Subdivision of Catmull-Clark Subdivision Surfaces," *Computer-Aided Design & Applications*, **2**(1-4), 253-261.

[4] Stam, J. (1998), "Exact Evaluation of Catmull-Clark Subdivision Surfaces at Arbitrary Parameter Values," *Proceedings of SIGGRAPH*, 395-404.

[5] Stam, J. (1999), "Evaluation of Loop Subdivision Surfaces," *SIGGRAPH'99 Course Notes*.

[6] Zorin, D. and Kristjansson, D. (2002), "Evaluation of Piecewise Smooth Subdivision Surfaces," *The Visual Computer*, **18**(5/6), 299-315.

[7] Lai, S. and Cheng, F. (2006), "Parametrization of General Catmull Clark Subdivision Surfaces and its Application," *Computer Aided Design & Applications*, **3**, 1-4.

[8] Garland, M. and Heckber, P. (1997), "Surface simplification using quadric error metrics," *Proceedings of SIGGRAPH*, 209-216.

[9] Settgast, V., Muller, K., Fuetzig, C., *et al.* (2004), "Adaptive Tesselation of Subdivision Surfaces," *Computers & Graphics*, pp.73-78.

[10] Amresh, A., Farin, G. and Razdan, A. (2002), "Adaptive Subdivision Schemes for Triangular Meshes," *Hierarchical and Geometric Methods in Scientific Visualization*, Springer-Verlag, pp.319-327.

[11] Wu, X. and Peters, J. (2005), "An Accurate Error Measure for Adaptive Subdivision Surfaces," *Shape Modeling International*.

[12] M. Boo, M. Amor, *et al.* (2001), "Hardware Support for Adaptive Subdivision Surface Rendering," *Proc. of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pp.33-40.

[13] Muller, K., Techmann, T. and Fellner, D. (2003), "Adaptive Ray Tracing of Subdivision Surfaces," *Computer Graphics Forum*, **22**(3), Sept.

[14] Smith, J. and Sequin, C., "Vertex-Centered Adaptive Subdivision," www.cs.berkeley.edu/~jordans/pubs/vertexcentered.pdf.

[15] Isenberg, T., Hartmann, K. and Konig, H. (2003), "Interest Value Driven Adaptive Subdivision," *Simulation und Visualisierung*, March 6-7, Magdeburg, Germany.

[16] Sederberg, T.W., Zheng, J., Sewell, D. and Sabin, M. (1998), "Non-uniform recursive subdivision surfaces," *Proceedings of SIGGRAPH*, 19-24.

[17] Sovakar, A. and Kobbelt, L. (2004), "API Design for adaptive subdivision schemes," 67-72, *Computers & Graphics*, **28**(1), Feb.

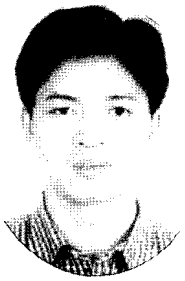
[18] Rose, D., Kada, M. and Ertl, T. (2001), "On-the-Fly Adaptive Subdivision Terrain," *Proceedings of the Vision Modeling and Visualization Conference*, Stuttgart, Germany, pp. 87-92, Nov.

[19] Wu, X. and Peters, J. (2004), "Interference detection for subdivision surfaces," *Computer Graphics Forum, Eurographics*, **23**(3), 577-585.

FUHUA (FRANK) CHENG is Professor of Computer Science and Director of the Graphics & Geometric Modeling Lab at the University of Kentucky. He holds a PhD from the Ohio State University, 1982. His research interests include computer aided geometric modeling, computer graphics, parallel computing in geometric modeling and computer graphics, approximation theory, and collaborative CAD.

SHUHUA LAI currently is a Ph.D student in the Department of Computer Science at the University of Kentucky. He received a BS in Applied Mathematics and Computer Applications from the East China Normal University, and an ME in Computer Science and Engineering from the Shanghai Jiaotong University. His research interests include computer graphics and modelling.

FUHUA (FRANK) CHENG



SHUHUA LAI