Adaptive Rendering 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

A new adaptive rendering method for Catmull-Clark subdivision surfaces is presented. The new method is based on direct evaluation of the limit surface to generate an inscribed polyhedron of the limit surface. The new method can precisely measure error for every point of the limit surface. Hence, it has complete control of the accuracy of the rendering 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 both computation and memory efficient.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modelling - curve, surface, solid and object representations;

Keywords: subdivision, Catmull-Clark surfaces, adaptive rendering, surface evaluation

1 Introduction

There are two possible approaches for the adaptive tessellation of a subdivision surface. One is a *mesh-refinement-based* (MRB) approach. It approximates the limit surface by adaptively refining the control mesh of the surface. The resulting mesh usually does not interpolate the limit surface. The other one is a *surface-evaluation-based* (SEB) approach. This approach approximates the limit surface by generating an inscribed polyhedron of the limit surface, with vertices of the polyhedron taken (evaluated) adaptively from the limit surface. The MRB 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 [2, 3, 4, 6], 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 [9]. This paper works parametrization of Loop subdivision scheme that reproduces linear functions [16]. Nothing has been done for parametrization of Catmull-Clark subdivision scheme [2, 6] yet.

In this paper we will present an SEB approach for adaptive tessellation of Catmull-Clark subdivision surfaces. The new method can precisely measure error for every point of the limit surface. Hence, it has complete control of the accuracy of the rendering 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 both computation and memory efficient.

The remaining part of the paper is arranged as follows. A brief review of previous works related to this one is given in Section 2. A description of the basic idea of our adaptive rendering technique is given in Section 3. The issue of crack elimination is discussed in Section 4. Algorithms of our technique are presented in Seciton 5. Test results are shown in Section 6. The concluding remarks are given in Section 7.

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 [1] 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

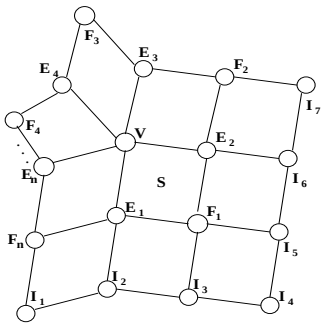


Figure 1: Control vertices of an extra-ordinary patch and their labeling.

faces of a new control mesh. A CCSS is the limit surface of the 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. Vertex **V** in Figure 1 is an extra-ordinary vertex of valence five. A mesh face with an extra-ordinary vertex is called an *extra-ordinary face*. [1] 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. The control vertices shown in Figure 1 are the ones that influence the patch marked with an "**S**". Recent work [2, 3, 4, 6] 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 [5, 7, 8, 9, 12, 13]. 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 [5], a subdivision depth is calculated for each patch of the given Catmull-Clark surface with respect to a given error tolerance ϵ. In [7], 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 [5] is error controllable. An error controllable approach for Loop surface is proposed in [9], 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 [8, 13, 12]. In [8], 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 [12].

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 [15] and K. Müller et al. combined ray tracing with adaptive subdivision surfaces to generate realistic scenes [11]. Adaptive tessellation is such an important technique that an API has been designed for its general usage [14]. Actually hardware implementation of this technique has been reported recently as well [10].

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 inscribed 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 [2, 3, 4, 6] 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 [6]. This technique is numerically stable, employs less eigen basis functions, and can be used to evaluate both position and normal of any point in the limit surface exactly

[1] Here, without loss of generality, we assume each patch has at most one extra-ordinary vertex

and explicitly. Some related results of [6] are summarized below.

The parametrization/evaluation approach of [6] is presented for general Catmull-Clark subdivision surface. That is, the new *vertex point* $\mathbf{V}'$ of $\mathbf{V}$ after one subdivision is computed as follows:

$$\mathbf{V}' = \alpha_n \mathbf{V} + \beta_n \sum_{i=1}^{n} \mathbf{E}_i + \gamma_n \sum_{i=1}^{n} \mathbf{F}_i$$

where α_n, β_n and γ_n are positive numbers and $\alpha_n + \beta_n + \gamma_n = 1$. In a general Catmull-Clark subdivision surface, new *face points* and *edge points* are computed the same way as in an ordinary Catmull-Clark subdivision surface [1]. The parametrization/evaluation approach of [6] is based on an $\Omega - partition$ of the parameter space [2, 6]. After a detoured subdivision path and some specific transforms [6], every point in the parameter space of a patch can be explicitly and precisely evaluated as follows.

$$\mathbf{S}(u,v) = W^T \mathrm{K}^m \sum_{j=0}^{n+5} \lambda_j^{m-1}\ \mathrm{M}_{b,j}\ G \qquad (1)$$

where n is the valance of the extraordinary patch, [2] W is a vector containing the 16 B-spline power basis functions:

$$\begin{aligned} W^T(u,v) \quad = \quad & [1, u, v, u^2, uv, v^2, u^3, u^2v, uv^2, v^3, \\ & u^3v, u^2v^2, uv^3, u^3v^2, u^2v^3, u^3v^3] \ , \end{aligned}$$

with $0 \le u, v \le 1$, K is a diagonal matrix:

$$\mathrm{K} = Diag(1, 2, 2, 4, 4, 4, 8, 8, 8, 8, 16, 16, 16, 32, 32, 64),$$

and m and b are defined as follows:

$$m(u,v) = min\{\lceil log_{\frac{1}{2}} u \rceil, \lceil log_{\frac{1}{2}} v \rceil\} \ ,$$

$$b(u,v) = \begin{cases} 1, & if \ \ 2^m u \ge 1 \ \ and \ \ 2^m v < 1 \\ 2, & if \ \ 2^m u \ge 1 \ \ and \ \ 2^m v \ge 1 \\ 3, & if \ \ 2^m u < 1 \ \ and \ \ 2^m v \ge 1 \ , \end{cases}$$

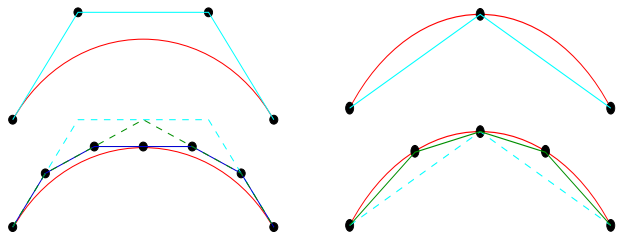
λ_j, $0 \le j \le n+5$, are eigenvalues of the Catmull-Clark subdivision metrix and $\mathrm{M}_{b,j}$, $1 \le b \le 3$, $0 \le j \le n+5$, are matrices of dimension $16 \times (2n+8)$. λ_j and $\mathrm{M}_{b,j}$ are independent of (u,v) and their exact expressions are given in [6]. G is the vector of control points (See Fig. 1 for their labeling):

$$G = [\mathbf{V}, \mathbf{E}_1, \cdots, \mathbf{E}_n, \mathbf{F}_1, \cdots, \mathbf{F}_n, \mathbf{I}_1, \cdots, \mathbf{I}_7]$$

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

$$\frac{\partial}{\partial u}\mathbf{S}(u,v) = (\frac{\partial W}{\partial u})^T\ \mathrm{K}^m \sum_{j=0}^{n+5} \lambda_j^{m-1}\ \mathrm{M}_{b,j}\ G. \qquad (2)$$

[2] Eq. (1) works for regular patches as well, i.e., when $n = 4$.



(a) Circumscribed (b) Inscribed

Figure 2: Inscribed and Circumscribed Approximation.

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

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

and the first derivatives:

$$\begin{aligned} \mathbf{D}_u(0,0) &= [0, 1, 0, 0, \cdots, 0] \cdot \mathrm{M}_{b,2} \cdot G \\ \mathbf{D}_v(0,0) &= [0, 0, 1, 0, \cdots, 0] \cdot \mathrm{M}_{b,2} \cdot G \end{aligned}$$

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

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 Figure 2(a), at the top are a cubic Bézier 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. 2(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" (more or less) the limit shape, we call this method *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 Figure 2(b), at the top are a cubic Bézier curve and its approximating polygon with vertices evaluated at parameter points 0, 1/2 and 1. Similarily, the solid polygon at the bottom of Figure 2(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 appromimation. As one can see from Fig. 2 that the inscribed polygon at the bottom of Fig. 2(b) is closer to the limit curve than the circumscribed polygon shown at the bottom of Fig. 2(a) even though the inscribed polygon actually has less segments than the circumscribed 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 $\mathbf{S}(u,v)$ 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 $\mathbf{V}_1 = \mathbf{S}(u_1, v_1)$, $\mathbf{V}_2 = \mathbf{S}(u_2, v_1)$, $\mathbf{V}_3 = \mathbf{S}(u_2, v_2)$ and $\mathbf{V}_4 = \mathbf{S}(u_1, v_2)$. If the distance (to be defined below) between the patch and its corresponding quadrilateral is small enough, then the patch is considered *flat* enough and will be (for now) replaced with the corresponding quadrilateral in the rendering process. Otherwise, we perform a midpoint subdivision on the parameter space by setting

$$u_{12} = \frac{u_1 + u_2}{2} \quad \text{and} \quad v_{12} = \frac{v_1 + v_2}{2}$$

to get four subpatches: $[u_1, u_{12}] \times [v_1, v_{12}]$, $[u_{12}, u_2] \times [v_1, v_{12}]$, $[u_{12}, u_2] \times [v_{12}, v_2]$, $[u_1, u_{12}] \times [v_{12}, v_2]$, 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 Figure 3(a) are the parameter spaces of four adjacent patches of $\mathbf{S}(u,v)$, and if the rectangles shown in Figure 3(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 inscribed polyhedron of the limit surface.

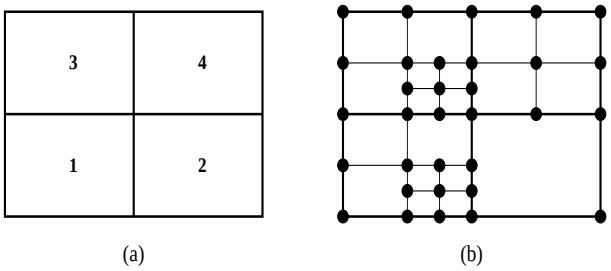


Figure 3: Basic idea of the construction of an inscribed polyhedron.

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 using a simple bilinear interpolation, as follows:

$$\begin{aligned} \mathbf{Q}(u,v) &= \tfrac{v_2-v}{v_2-v_1}\left(\tfrac{u_2-u}{u_2-u_1}\mathbf{V}_1 + \tfrac{u-u_1}{u_2-u_1}\mathbf{V}_2\right) \\ &\quad + \tfrac{v-v_1}{v_2-v_1}\left(\tfrac{u_2-u}{u_2-u_1}\mathbf{V}_4 + \tfrac{u-u_1}{u_2-u_1}\mathbf{V}_3\right) \end{aligned} \tag{4}$$

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

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

where $\| \cdot \|$ is the second norm and $\mathbf{A}^T$ is the transpose of $\mathbf{A}$. 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 $\mathbf{Q}(u,v)$ and $\mathbf{S}(u,v)$ are both C^1-continuous, and $d(\mathbf{V}_1)$, $d(\mathbf{V}_2)$, $d(\mathbf{V}_3)$ and $d(\mathbf{V}_4)$

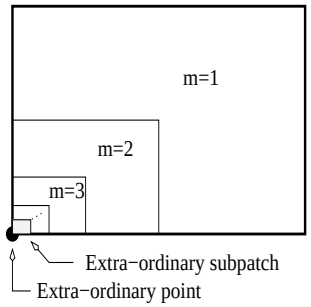


Figure 4: Partitioning of the unit square.

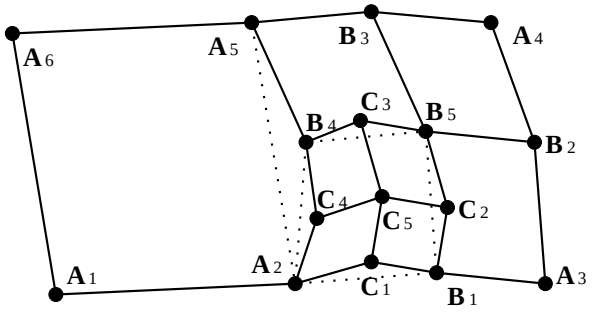


Figure 5: Crack prevention.

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:

$$\begin{cases} \frac{\partial d(u,v)}{\partial u} = 0 \\ v = v_1 \text{ or } v = v_2 \\ u_1 \le u \le u_2 \end{cases}$$

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

$$\begin{cases} \frac{\partial d(u,v)}{\partial u} = 0 \\ \frac{\partial d(u,v)}{\partial v} = 0 \\ (u,v) \in (u_1, u_2) \times (v_1, v_2) \end{cases}$$

For a patch (or subpatch) that is not adjacent to an extraordinary point (i.e., $(u_1, v_1) \neq (0, 0)$), m is fixed and known ($m(u,v) = min\{\lceil log_{\frac{1}{2}} u \rceil, \lceil log_{\frac{1}{2}} v \rceil\}$). 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. The patch (or subpatch) is said to be *flat* enough if

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

where ϵ is a given error tolerance. In such a case, the patch (or subpatch) is replaced with the corresponding quadrilateral in the rendering 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 ∞ (see Figure 4). 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 $\mathbf{S}(u,v)$ and $\mathbf{Q}(u,v)$ both converge to $\mathbf{S}(0,0)$. Hence, for any given error tolerance ϵ, there exists an integer m_ϵ such that if $m \ge m_\epsilon$, then the difference between $\mathbf{S}(u,v)$ and $\mathbf{S}(0,0)$ is smaller than $\epsilon/2$ for any $(u,v) \in [0, 1/2^m] \times [0, 1/2^m]$, and so is the difference between $\mathbf{Q}(u,v)$ and $\mathbf{S}(0,0)$. Consequently, when $(u,v) \in [0, 1/2^m] \times [0, 1/2^m]$, the difference between $\mathbf{S}(u,v)$ and $\mathbf{Q}(u,v)$ is smaller than ϵ. The value of m_ϵ, in most of the cases, is smaller than 12. For other regions of the unit square with $\lceil \log_{\frac{1}{2}} u_2 \rceil \le m < m_\epsilon$, eq. (6) can be used to find the difference between $\mathbf{S}(u,v)$ and $\mathbf{Q}(u,v)$ (see Figure 4). 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 rendering process. Otherwise, repeatedly subdivide the patch (or subpatch) and perform flatness testing on the resulting subpatches until all the subpatches satisfy Eq. (7).

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 Figure 3, patch 2 is approximated by one quadrilateral but patch 4 is approximated by 4 quadrilaterals and patch 1 is approximated by 7 quadrilaterals. Consider the boundary shared by patch 1 and patch 2. On the patch 2 side, that boundary is a line segment defined by two vertices : $\mathbf{S}(\mathbf{D})$ and $\mathbf{S}(\mathbf{G})$. But on the patch 1 side, the boundary is a polyline defined by four vertices : $\mathbf{S}(\mathbf{D})$, $\mathbf{S}(\mathbf{E})$, $\mathbf{S}(\mathbf{F})$, and $\mathbf{S}(\mathbf{G})$. They would not coincide unless $\mathbf{S}(\mathbf{E})$ and $\mathbf{S}(\mathbf{F})$ lie on the line segment defined by $\mathbf{S}(\mathbf{D})$ and $\mathbf{S}(\mathbf{G})$. But that usually is not the case. Hence, cracks would appear between patch 1 and patch 2.

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 Figure 5, all the dashed lines should be replaced with the corresponding polylines. In particular, boundary $\mathbf{A}_2\mathbf{A}_5$ of patch $\mathbf{A}_1\mathbf{A}_2\mathbf{A}_5\mathbf{A}_6$ should be replaced with the polyline $\mathbf{A}_2\mathbf{C}_4\mathbf{B}_4\mathbf{A}_5$. As a result, polygon $\mathbf{A}_1\mathbf{A}_2\mathbf{A}_5\mathbf{A}_6$ is replaced with polygon $\mathbf{A}_1\mathbf{A}_2\mathbf{C}_4\mathbf{B}_4\mathbf{A}_5\mathbf{A}_6$ in the rendering 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 vertices. The points shown in Figure 5 are points of the limit surface, not points in the parameter space of the limit surface.

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:

$$\sqrt{d\ (\ \bar{u},\ \bar{v})\ } + \sqrt{d\ (\ \hat{u},\ \hat{v})\ } \leq \epsilon \qquad (8)$$

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 $Q(u, v)$, i.e. solutions that satisfy $Q(u, v) \geq 0$, $d(\hat{u}, \hat{v})$ is the biggest.
- Among all the solutions of Eq. (6) that are located on the other side of $Q(u, v)$, i.e. solutions that satisfy $Q(u, v) < 0$, $d(\bar{u}, \bar{v})$ is the biggest.

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, in Fig. 5, points $\mathbf{A}_2, \mathbf{C}_4, \mathbf{B}_4$ and $\mathbf{A}_5$ of patch $\mathbf{A}_2\mathbf{A}_3\mathbf{A}_4\mathbf{A}_5$ are also points of patch $\mathbf{A}_1\mathbf{A}_2\mathbf{A}_5\mathbf{A}_6$. 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 points $\mathbf{A}_2, \mathbf{C}_4, \mathbf{B}_4$ and $\mathbf{A}_5$ are on the limit surface, they are located between two quadrilaterals that are ϵ away. So is the polygon $\mathbf{A}_1\mathbf{A}_2\mathbf{C}_4\mathbf{B}_4\mathbf{A}_5\mathbf{A}_6$. 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 [7, 5, 8, 12], the most difficult part is crack prevention. Yet in our method, this part is the simplest part to handle and implement. The resulting surface is error controllable and guaranteed to be crack free.

5 Algorithms

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

5.1 Global Index ID

All currently available subdivision surface parametrization and evaluation techniques are patch based [2, 4, 6]. Hence, no matter which method is used in the adaptive tessellation process, a patch, from its own (local) structure, cannot see vertices generated by adjacent patches, even the vertices are generated on a common boundary. For example, in Figure 5, vertices $\mathbf{C}_4$ and $\mathbf{B}_4$ are on the shared boundary of patches $\mathbf{A}_1\mathbf{A}_2\mathbf{A}_5\mathbf{A}_6$ and $\mathbf{A}_2\mathbf{A}_3\mathbf{A}_4\mathbf{A}_5$. But patch $\mathbf{A}_1\mathbf{A}_2\mathbf{A}_5\mathbf{A}_6$ cannot see these vertices from its own structure because these vertices are not generated by this patch. To make activities of adjacent patches visible to each other and, consequently, make crack detection unnecessary, one should assign a *global index ID* to each evaluated vertex so that

- all 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 $(u_i, v_i) \geq (u_j, v_j)$, then $ID_i \geq ID_j$, unless ID_i or ID_j has been used in previous patch evaluation.

With a global index ID, crack prevention is not a problem 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 Figure 5, patch $\mathbf{A}_1\mathbf{A}_2\mathbf{A}_5\mathbf{A}_6$ can see both $\mathbf{C}_4$ and $\mathbf{B}_4$ even though they are not evaluated by this patch. In the subsequent rendering process, the patch simply

output all the marked vertices (to be defined below) on its boundary that it can see to form a polygon for the rendering purpose, i.e., $\mathbf{A}_1\mathbf{A}_2\mathbf{C}_4\mathbf{B}_4\mathbf{A}_5\mathbf{A}_6$.

5.2 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 rendering process, then that point is marked in black. This process can be easily implemented as a recursive function. A pseudo code for this step is given below.

```
AdaptiveMarking(P, u1, u2, v1, v2)
1.   Evaluate(P, u1, u2, v1, v2),
2.   AssignGlobalID(P, u1, u2, v1, v2),
3.   if (FlatEnough(P, u1, u2, v1, v2))
4.       MarkBlack(P, u1, u2, v1, v2)
5.   else
6.       u12 = (u1 + u2)/2
7.       v12 = (v1 + v2)/2
8.       AdaptiveMarking(P, u1, u12, v1, v12)
9.       AdaptiveMarking(P, u12, u2, v1, v12)
10.      AdaptiveMarking(P, u12, u2, v12, v2)
11.      AdaptiveMarking(P, u1, u12, v12, v2)
```

This routine adaptively marks points in the parameter space of patch $\mathbf{P}$. Function 'Evaluate' evaluates limit surface at the four corners of patch or subpatch $\mathbf{P}$ 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 $\mathbf{P}$ defined on $[u_1, u_2] \times [v_1, v_2]$ in black. All the marked corner points will be used in the rendering process.

5.3 Adaptive Rendering a Single Patch

The purpose of this step is to render 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 rendering process. To avoid cracks, each marked points must be rendered 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.

```
AdaptiveRendering(P, u1, u2, v1, v2)
1.   if (NoMarkedPointInside(P, u1, u2, v1, v2))
2.       RenderPolygon(P, u1, u2, v1, v2)
3.   else
4.       u12 = (u1 + u2)/2
5.       v12 = (v1 + v2)/2
6.       AdaptiveRendering(P, u1, u12, v1, v12)
7.       AdaptiveRendering(P, u12, u2, v1, v12)
8.       AdaptiveRendering(P, u12, u2, v12, v2)
9.       AdaptiveRendering(P, u1, u12, v12, v2)
```

This routine adaptively renders marked points in patch or subpatch $\mathbf{P}$. 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 'RenderPolygon' is defined as follows.

```
RenderPolygon(P, u1, u2, v1, v2)
1.   glBegin(RenderModel)
2.       Output all the marked points between
3.           (u1, v1) → (u2, v1)
4.           (u2, v1) → (u2, v2)
5.           (u2, v2) → (u1, v2)
6.           (u1, v2) → (u1, v1)
7.   glEnd()
```

6 Test Results

The proposed approach has been implemented in *C++* using *OpenGL* as the supporting graphics system on the Windows platform. Some of the tested results are shown in Figure 6. We also summarize those tested results in Table 1. The column underneath A|U in Table 1 indicates the type of tessellation technique (Adaptive or Uniform) used in the rendering process. The term *A/U ratio* means the ratio of number of polygons in an adaptively tessellated CCSS to its counter part in a uniformly tessellated CCSS with the same accuracy. From Table 1 we can see that all the adaptively tessellated CCSS's have relatively low A/U ratios. The error in the last column is absolute error. We can easily see that, for the same model, the smaller the error, the lower the A/U ratio. For example, Fig. 6(g) has lower A/U ratio than Fig. 6(h) and Fig. 6(i). The same holds for Fig. 6(l) and Fig. 6(m). An interesting fact is that Fig. 6(f) uses many more polygons than Fig. 6(g) does, while the former is less accurate than the latter. This shows the presented adaptive tessellation method is capable of providing a higher accuracy with less polygons. However, for different models, comparing their absolute errors might not make practical

sense because absolute error is not Affine transformation invariant.

Table 1: Extra information of Fig. 6

Figure	A|U	polygons	A/U Ratio	Error
Fig. 6(a)	U	20480	100.00%	0.1
Fig. 6(b)	A	4688	22.89%	0.1
Fig. 6(c)	A	8017	8.26%	0.015
Fig. 6(f)	U	6912	100.00%	0.105
Fig. 6(g)	A	2068	7.48%	0.050
Fig. 6(h)	A	1432	20.72%	0.105
Fig. 6(i)	A	412	23.84%	0.250
Fig. 6(k)	U	22656	100.00%	1.0
Fig. 6(l)	A	3048	13.45%	1.0
Fig. 6(m)	A	2238	39.51%	1.5
Fig. 6(o)	A	6654	28.15%	0.0003
Fig. 6(p)	U	17088	100.00%	0.02
Fig. 6(q)	A	11544	4.22%	0.01

7 Summary

An adaptive rendering method based on inscribed approximation 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, and it takes almost no effort for the new method to eliminate cracks in the resulting inscribed polyhedron of the limit surface. Hence the new mthod is both computation efficient and memory efficient.

Currently, all the methods for adaptive rendering work on a patch by patch basis. One of our future works is to take the whole surface into consideration, so that not only the inner-patch redundancy, but the inter-patch redundancy, can be eliminated as well.

Acknowledgement. Data set for Fig. 6(o) was downloaded from the following web site
http://graphics.cs.uiuc.edu/~garland/research/quadrics.html.
The package *qslim* was used to reduce the number of faces in this model to 500.

References

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

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

[3] Stam J, Evaluation of Loop Subdivision Surfaces, *SIGGRAPH'99 Course Notes*, 1999.

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

[5] Fuhua (Frank) Cheng and Junhai Yong, Adaptive Subdivision of Catmull-Clark Subdivision Surfaces, *Computer-Aided Design & Applications* 2,1-4, 2005.

[6] Shuhua Lai and Fuhua (Frank) Cheng, Parametrization of General Catmull Clark Subdivision Surfaces and its Application, submitted. www.cs.uky.edu/~cheng/PUBL/para.pdf.

[7] V. Settgast, K. Müller, Christoph Fünfzig et.al., Adaptive Tesselation of Subdivision Surfaces, In *Computers & Graphics*, 2004, pp:73-78.

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

[9] Xiaobin Wu and Jörg Peters, An Accurate Error Measure for Adaptive Subdivision Surfaces, In *Shape Modeling International*, 2005

[10] M. Bo, M. Amor, M. Doggett, et.al., Hardware Support for Adaptive Subdivision Surface Rendering, In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware* 2001, pp:33-40.

[11] K. Müller, T. Techmann and D. Fellner, Adaptive Ray Tracing of Subdivision Surfaces *Computer Graphics Forum* Vol 22, Issue 3 (Sept 2003).

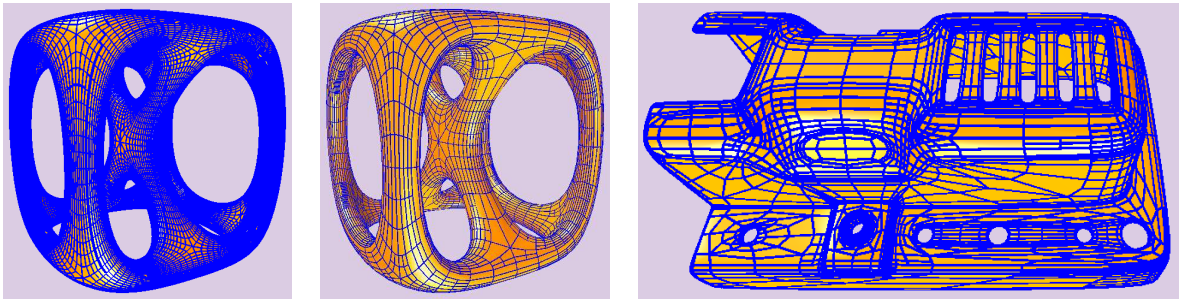
[12] Jordan Smith and Carlo Séquin, Vertex-Centered Adaptive Subdivision, www.cs.berkeley.edu/~jordans/pubs/vertexcentered.pdf.

[13] T. Isenberg, K. Hartmann and H. König. Interest Value Driven Adaptive Subdivision, In *Simulation und Visualisierung*, March 6-7, 2003, Magdeburg, Germany.

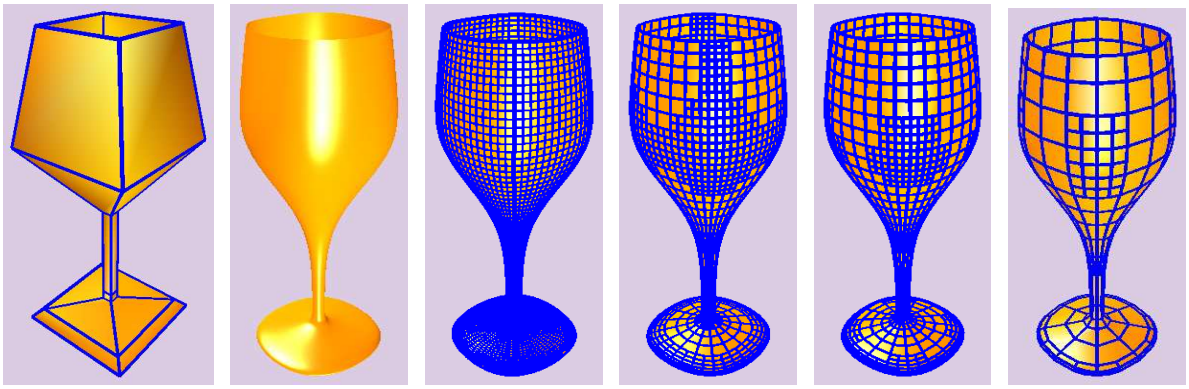
[14] A. Sovakar and Leif Kobbelt, API Design for adaptive subdivision schemes. 67-72, *Computers & Graphics*, Vol. 28, No. 1, Feb. 2004.

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

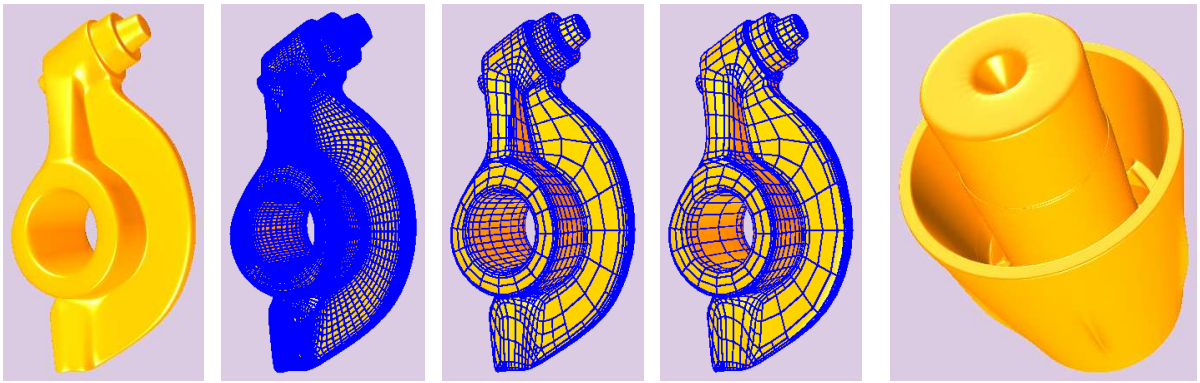
[16] X. Wu and J. Peters, Interference detection for subdivision surfaces, *Computer Graphics Forum, Eurographics* 23(3):577585, 2004.



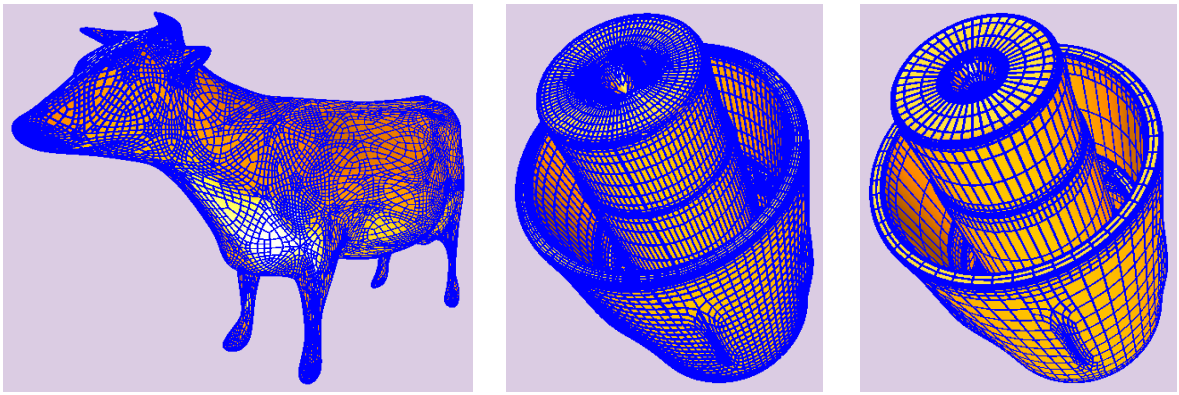
(a) Uniform Evaluation (b) Adaptive Evaluation (c) Adaptive Evaluation



(d) Mesh (e) Limit Surface (f) Uniform (g) Adaptive (h) Adaptive (i) Adaptive



(j) Limit Surface (k) Uniform (l) Adaptive (m) Adaptive (n) Limit Surface



(o) Adaptive Evaluation (p) Uniform Evaluation (q) Adaptive Evaluation

Figure 6: Adaptive rendering of surfaces with arbitrary topology.