Near-Optimum Adaptive Tessellation of General Catmull-Clark Subdivision Surfaces

Shuhua Lai and Fuhua (Frank) Cheng (University of Kentucky)

Graphics & Geometric Modeling Lab, Department of Computer Science,
University of Kentucky, Lexington, Kentucky 40506-0046,
{slai2|cheng}@cs.uky.edu

Abstract. A new adaptive tessellation method for general Catmull-Clark subdivision surfaces is presented. Development of the new method is based on the observation that *optimum adaptive tessellation* for rendering purpose is a *recursive error evaluation* and *globalization* process. The adaptive tessellation process is done by generating an inscribing polyhedron to approximate the limit surface for each individual patch. The inscribing polyhedron is generated through an adaptive subdivision on the patch's parameter space driven by a recursive error evaluation process. This approach generates less faces in the resulting approximating mesh while meeting the given precision requirement. The crack problem is avoided through globalization of new vertices generated in the adaptive subdivision process of the parameter space. No crack-detection or crack-elimination is needed in the adaptive tessellation process. Therefore, no mesh element splitting to eliminate cracks is necessary. The new adaptive tessellation method can precisely measure the error for every point of the limit surface. Hence, it has complete control of the accuracy of the tessellation result.

1 Introduction

Catmull-Clark subdivision scheme provides a powerful method for building smooth and complex surfaces. Given a control mesh, a *Catmull-Clark subdivision surface* (CCSS) is generated by iteratively refining (subdividing) the control mesh to form new and finer control meshes [3]. 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 [3]. 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.

2 Previous Work

A number of adaptive tessellation methods for subdivision surfaces have been proposed [9, 1, 13, 10, 4, 14]. 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. Several other adaptive tessellation schemes have been presented as well [1, 10, 4]. A common problem of these schemes is that they all have to develop complicated process to prevent the occurance of cracks.

In addition to various adaptive tessellation schemes, there are also applications of these techniques. In [8] adaptive tessellation method is used to render terrain and in [7] adaptive tessellation is combined with ray tracing techniques to generate some realistic scenes. Adaptive tessellation is so useful that an API has been designed for its general usage [11]. Actually hardware implementation of this technique has been reported recently as well [2].

A problem with mesh-refinement-based, adaptive tessellation techniques is the possible *over-tessellation problem*. Each region, such as a patch, where a subdivision depth is assigned is uniformly subdivided to the level specified by the subdivision depth. Since the subdivision depth is computed based on the largest possible curvature of the region, parts of the region which do not carry such a large curvature will be unnecessarily subdivided.

Another problem is the so called *crack-prevention requirement*. Because the number of new vertices generated on the boundary of a region depends on the subdivision depth, *cracks* would occur between adjacent regions if these regions are assigned different subdivision depths. Hence, such an adaptive tessellation method needs special mechanism to eliminate cracks. This is usually done by performing additional subdivision or splitting steps on the region with lower subdivision depth. As a result, many unnecessary mesh elements are generated.

3 Basic Idea

Given the control mesh of a CCSS and an error tolerance, ϵ, the goal is to generate an approximating polyhedron mesh close enough to the limit surface $\mathbf{S}(u, v)$, i.e., within the error tolerance of $\mathbf{S}(u, v)$, but with as few mesh faces as possible, so that the rendering process of $\mathbf{S}(u, v)$ can be performed efficiently. An approximating polyhedron mesh with the least number of mesh faces is called an *optimum approximating polyhedron mesh*.

Our first goal is to avoid the possible *over-tessellation problem*. It is easy to see that, to achieve such a goal, tessellation process within each patch should also be performed based on the flatness of each local region. This can be accomplished by doing *adaptive subdivision* on the parameter space of each patch that is driven by a *recursive error evaluation process*. Contrary to the *mesh refinement based approaches* which generate approximating polyhedra from 'outside' the limit surface that usually do not interpolate the limit surface, the approximating polyhedron generated by this approach is an *inscribing polyhedron* whose vertices interpolate the limit surface.

For a patch of $\mathbf{S}(u, v)$ defined on $[u_1, u_2] \times [v_1, v_2]$, we approximate it with the *base 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* (error) (to be defined below) between the patch and its base quadrilateral is small than ϵ, the patch is consid-

ered *flat* enough and is replaced with the base 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: $[u_1, u_{12}] \times [v_1, v_{12}]$, $[u_{12}, u_2] \times [v_1, v_{12}]$, $[u_{12}, u_2] \times [v_{12}, v_2]$, $[u_1, u_{12}] \times [v_{12}, v_2]$, and repeat the *flatness testing* (error evaluation) process on each of the subpatches. The process is recursively repeated until the distances (errors) between all the subpatches and their corresponding base quadrilaterals are smaller than ϵ. The vertices of the resulting subpatches are then used as vertices of the inscribing polyhedron that approximates the limit surface. For example, if the four rectangles in Figure 1(a) are the parameter spaces of four adjacent patches of $\mathbf{S}(u, v)$, and if the rectangles shown in Figure 1(b) are the parameter spaces of the resulting subpatches when the above *recursive flatness testing (error evaluation) process* stops, then the limit surface will be *evaluated* at the points marked with small solid circles to form vertices of an inscribing approximating polyhedron of the limit surface.

This is a simple and straightforward process, but the result could be very significant. Note that each face in the inscribed approximating polyhedron for a patch is built with the expectation that it is just close enough to the limit surface but with the maximum possible size. Therefore, if the recursive error evaluation process can indeed provide precise error estimate, then the approximating polyhedron mesh generated by this process is *optimum* or *near-optimum* (in case some faces from different sides of a common boundary of two patches can be merged into a bigger face with the same error size). To ensure that the approximating polyhedron mesh is precisely constructed, we must also be able to precisely evaluate a CCSS at any given parameter point. With parametrization of CCSS becoming available [12, 5], this is always possible now.

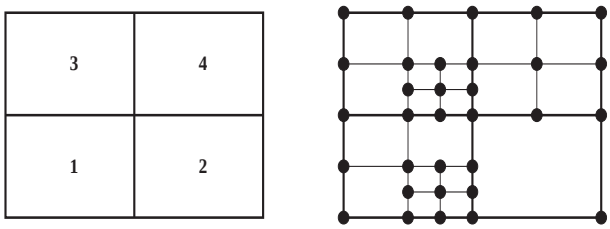


Fig. 1. Adaptive subdivision on parameter spaces of patches.

Our second goal is to avoid the *crack prevention requirement*. Due to the fact that adjacent patches are usually approximated by base quadrilaterals from different levels of the midpoint subdivision process, cracks could occur between adjacent patches. For instance, in Figure 2, the left patch is approximated by one base quadrilateral but the right patch is approximated by 7 base quadrilaterals. Consider the boundary shared by the left patch and the right patch. On the left side, that boundary is approximated by a line segment defined by two vertices, $\mathbf{A}_2$ and $\mathbf{A}_5$. But on the right side, the boundary is approximated by a polyline

defined by four vertices, $\mathbf{A}_2$, $\mathbf{C}_4$, $\mathbf{B}_4$, and $\mathbf{A}_5$. They do not coincide unless $\mathbf{C}_4$ and $\mathbf{B}_4$ lie on the line segment defined by $\mathbf{A}_2$ and $\mathbf{A}_5$. But this usually is not the case. Hence, a crack would appear between the left patch and the right patch.

Fortunately Cracks can be removed simply by replacing edges of the base quadrilaterals with appropriate polylines in the tessellation process. Namely, each edge of a base quadrilateral should be replaced with a polyline defined with all the new vertices computed for that edge of the corresponding patch or subpatch. For example, in Figure 2, all the dashed lines should be replaced with the corresponding polylines. In particular, edge $\mathbf{A}_2\mathbf{A}_5$ of the base quadrilateral $\mathbf{A}_1\mathbf{A}_2\mathbf{A}_5\mathbf{A}_6$ should be replaced with the polyline $\mathbf{A}_2\mathbf{C}_4\mathbf{B}_4\mathbf{A}_5$. As a result, the left patch is approximated by the polygon $\mathbf{A}_1\mathbf{A}_2\mathbf{C}_4\mathbf{B}_4\mathbf{A}_5\mathbf{A}_6$, instead of the base quadrilateral $\mathbf{A}_1\mathbf{A}_2\mathbf{A}_5\mathbf{A}_6$, in the tessellation process. For rendering purpose this is fine because graphics systems like OpenGL can handle polygons with any number of vertices and the vertices do not have to be co-planar. Note that, with the above approach, there is no need to perform *crack detection* at all because the resulting approximating polyhedron contains no cracks. Besides, since this process does not increase the number of faces in an approximating polyhedron, the resulting approximating polyhedron is *optimum* or *near-optimum* for the entire CCSS. For convenience of subsequent reference, the process of replacing

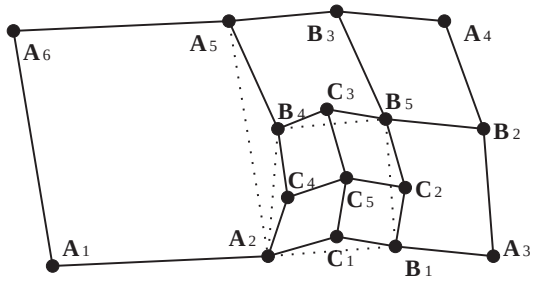


Fig. 2. Cracks between adjacent patches (subpatches).

edges of base quadrilaterals with new polylines is called a *base quadrilateral replacement process*.

Note that in previous methods for adaptive tessellation of subdivision surfaces [14, 9, 1, 10], the most difficult part is crack prevention. With the above approach, this part becomes the simplest part to handle and implement.

4 Flatness Testing (Error Evaluation)

In the *flatness testing process*, to measure the difference between a patch (or subpatch) and its *base quadrilateral*, we need to parametrize the base quadrilateral as well. The base quadrilateral can be parametrized with a simple bilinear interpolation:

$$\mathbf{Q}(u,v) = \frac{v_2-v}{v_2-v_1}\left(\frac{u_2-u}{u_2-u_1}\mathbf{V}_1 + \frac{u-u_1}{u_2-u_1}\mathbf{V}_2\right) + \frac{v-v_1}{v_2-v_1}\left(\frac{u_2-u}{u_2-u_1}\mathbf{V}_4 + \frac{u-u_1}{u_2-u_1}\mathbf{V}_3\right) \quad (1)$$

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

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

The *distance* between the patch (or subpatch) and the base 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 base 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 $d(\mathbf{V}_4)$ 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:

$$\begin{cases} \frac{\partial d(u,v)}{\partial u} = 0 \\ v = v_1 \text{ or } v = v_2 \\ u_1 \le u \le u_2 \end{cases} \quad \begin{cases} \frac{\partial d(u,v)}{\partial v} = 0 \\ u = u_1 \text{ or } u = u_2 \\ v_1 \le v \le v_2 \end{cases} \quad \begin{cases} \frac{\partial d(u,v)}{\partial u} \;=\; 0 \\ \frac{\partial d(u,v)}{\partial v} \;=\; 0 \\ (u,v) \in (u_1,u_2) \times (v_1,v_2) \end{cases} \tag{3}$$

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. (3) can be solved explicitly. With the valid solutions, we can find the difference for each of them using Eq. (2). 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 the corresponding base quadrilateral. The patch (or subpatch) is considered *flat* enough if

$$\sqrt{d\ (\ \hat{u},\ \hat{v})\ } \quad \le \quad \epsilon \tag{4}$$

where ϵ is a given error tolerance. In such a case, the patch (or subpatch) is replaced with the corresponding base quadrilateral in the tessellation process.

For a patch that is adjacent to an extraordinary point (i.e. $(u_1, v_1) = (0, 0)$ in Eq. (3)), m is not fixed and m tends to ∞. As a result, Eq. (3) 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.

$\mathbf{S}(u, v)$ and $\mathbf{Q}(u, v)$ both converge to $\mathbf{S}(0, 0)$ when $(u, v) \to (0, 0)$. Hence, for any given error tolerance ϵ, there exists an integer m_ϵ such that if $m \ge m_\epsilon$, then the distance 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 distance 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 distance 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 a relatively small number and can be explicitly calculated [6].

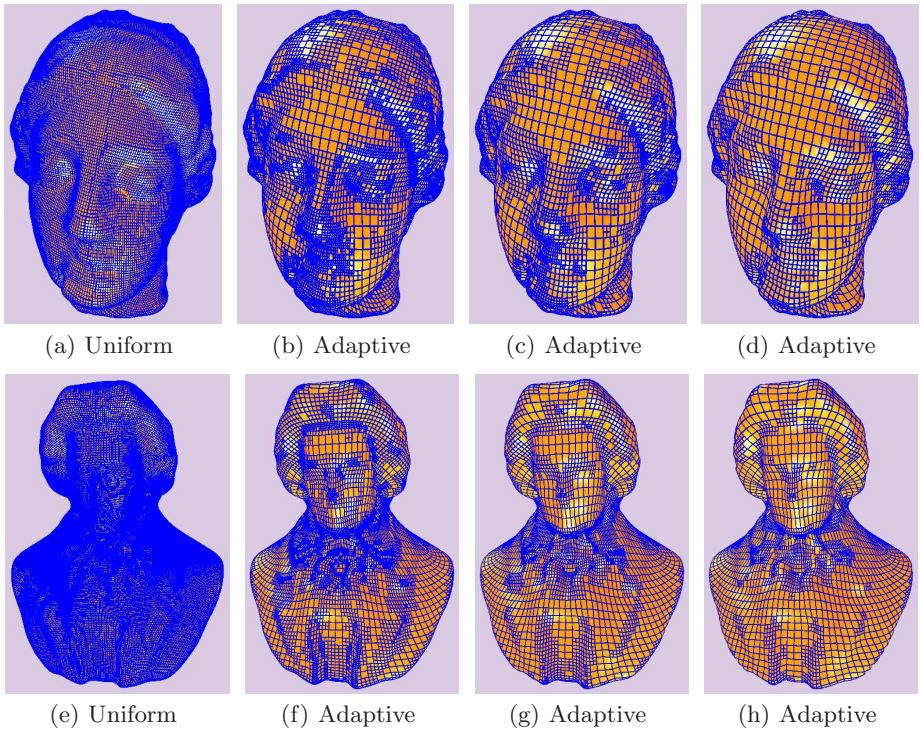
For other regions of the unit square with $\lceil \log_{\frac{1}{2}} u_2 \rceil \leq m < m_\epsilon$, eq. (3) can be used directly to find the difference between $\mathbf{S}(u, v)$ and $\mathbf{Q}(u, v)$ for any fixed $m \in (\lceil \log_{\frac{1}{2}} u_2 \rceil, m_\epsilon)$. Therefore, by combining all these differences, we have the distance between the given extra-ordinary patch (or subpatch) and the corresponding base quadrilateral. If this distance is smaller than ϵ, we consider the given extra-ordinary patch (or subpatch) to be flat, and use the base 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. (4).

5 Making Patches Visible to Each Other

Currently, all the subdivision surface parametrization and evaluation techniques are patch based [12, 5]. Hence, no matter which method is used in the 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 Figure 2, vertices $\mathbf{C}_4$ and $\mathbf{B}_4$ are on the shared boundary of the left and the right patches. But the left patch can not see these vertices from its own structure because these vertices are not evaluated by this patch. So, the key here is to make adjacent patches visible to each other so that new vertices computed by one patch for the shared boundary can be accessed by the other patch. We call this process a *globalization process*.

To make adjacent patches visible to each other and to make subsequent *base quadrilateral replacement process* easier, one should assign a *global index ID* to each evaluated vertex so that all the evaluated vertices with the same 3D position have the same index ID. Global indexing allows subsequent processing to be performed on individual patches but still with a global visibility. We also need a step called *adaptive marking* to facilitate the *base quadrilateral replacement process*. 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. A pseudo code for this step is given below.

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



(a) Uniform (b) Adaptive (c) Adaptive (d) Adaptive

(e) Uniform (f) Adaptive (g) Adaptive (h) Adaptive

Fig. 3. Adaptive tessellation of surfaces with arbitrary topology.

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 'AssignGlobeID' assignes global index ID to the four corners of **P**. Function 'FlatEnough' uses the method given in Section 4 and Eq. (4) 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 Implementation and 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 Fig. 3. All these testing models have some extra-ordinary points in the input meshes. For the Venus model, uniform subdivision (Fig. 3(a)) generates 65536 polygons, while with a similar or higher accuracy, adaptive tessellation only requires 29830, 21841, and 9763 polygons for Fig. 3(b), 3(c) and 3(d) respectively. For the Beethoven model, uniform subdivision (Fig. 3(e)) generates 65536 polygons, while with a similar or higher accuracy, adaptive tessellation

only requires 20893, 15622, and 7741 polygons for Fig. 3(f), 3(g) and 3(h), respectively. Hence the proposed method indeed significantly reduces the number of faces in the resulting tessellation while satisfying the given error requirement.

7 Summary

A new adaptive tessellation method for general CCSSs is presented. The method is developed for rendering purpose and is based on the observation that *optimum adaptive tessellation* for rendering purpose is a *recursive error evaluation* and *globalization* process for indivisual patches. For a CCSS with multi-patches, the result of our work can be improved by running a post-processor to see if some faces from different sides of a patch boundary can be merged into a bigger face with the same error size. However, since the improvement is not significant and the computation is costly, it might not be worth the effort to do so.

Acknowledgement: Research work of the authors is supported by NSF under grants DMS-0310645 and DMI-0422126. Data sets for Fig. 3 are downloaded from the web site: http://research.microsoft.com/∼hoppe .

References

1. Amresh A, Farin G, Razdan A, Adaptive Subdivision Schemes for Triangular Meshes, *Hierarchical and Geometric Methods in Scientific Visualization*, 2002.
2. M. Boo, M. Amor, M. Doggett, et.al., Hardware Support for Adaptive Subdivision Surface Rendering, *SIGGRAPH workshop on Graphics hardware* 2001, pp:33-40.
3. Catmull E, Clark J. Recursively generated B-spline surfaces on arbitrary topological meshes, *Computer-Aided Design*, 1978, 10(6):350-355.
4. Isenberg T, Hartmann K, König H, Interest Value Driven Adaptive Subdivision, In *Simulation und Visualisierung*, March 6-7, 2003, Magdeburg, Germany.
5. Lai S, Cheng F, Parametrization of General Catmull Clark Subdivision Surfaces and its Application, *Computer Aided Design & Applications 3*, 1-4, 2006.
6. Lai S, Cheng F, Near-Optimum Adaptive Tessellation of General Catmull-Clark Subdivision Surfaces (complete version), www.cs.uky.edu/∼cheng/PUBL/adaptive.pdf.
7. Müller K, Techmann T, Fellner D, Adaptive Ray Tracing of Subdivision Surfaces *Computer Graphics Forum* Vol 22, Issue 3 (Sept 2003).
8. Rose D, Kada M, Ertl T, On-the-Fly Adaptive Subdivision Terrain. In *Proceedings of the Vision Modeling and Visualization Conference*, pp: 87-92, Nov. 2001.
9. Settgast V, Müller K, Fünfzig C, et.al., Adaptive Tesselation of Subdivision Surfaces, In *Computers & Graphics*, 2004, pp:73-78.
10. Smith J, Séquin C, Vertex-Centered Adaptive Subdivision, www.cs.berkeley.edu/∼jordans/pubs/vertexcentered.pdf.
11. Sovakar A, Kobbelt L, API Design for adaptive subdivision schemes. 67-72, *Computers & Graphics*, 28, 1, Feb. 2004.
12. Stam J, Exact Evaluation of Catmull-Clark Subdivision Surfaces at Arbitrary Parameter Values, *Proceedings of SIGGRAPH* 1998:395-404.
13. Wu X, Peters J, An Accurate Error Measure for Adaptive Subdivision Surfaces, In *Shape Modeling International*, 2005.
14. Yong J, Cheng F, Adaptive Subdivision of Catmull-Clark Subdivision Surfaces, *Computer-Aided Design & Applications* 2(1-4):253-261, 2005.