An Accurate Error Measure for Adaptive Subdivision Surfaces

Xiaobin Wu
University of Florida
xwu@cise.ufl.edu

Jörg Peters
University of Florida
jorg@cise.ufl.edu

Abstract

A tight estimate on the maximum distance between a subdivision surface and its linear approximation is introduced to guide adaptive subdivision with guaranteed accuracy.

1. Introduction

Subdivision provides a simple and powerful method for modeling free-form surfaces: given a polygonal input mesh, a sequence of refinements generates an ever denser mesh with a generically smooth limit surface. For modern graphics applications, input meshes can consist of thousands of faces. If, at each step of refinement, every mesh facet is split into a fixed number of new faces, the number of facets increases exponentially and the complexity of the mesh quickly exceeds the memory and processing limitations. The obvious answer, adaptive refinement, requires a good bound on how well planar triangles approximate the limit surface. Loose bounds waste resources and overly aggressive approximations can miss surface features. Due to the procedural definition of the subdivision surface, adaptive refinement is more tricky than for standard spline surfaces.

In this paper, we leverage a new bound on the maximum distance between the limit surface and its linear approximation. The bound can be computed locally and efficiently, and yields a tight estimate with an error converging to zero under subdivision. Figure 1 shows adaptive meshes for a given threshold e and Figure 2 compares the new bound to conventional ones.

1.1. Previous work

A number of error estimates have been used for adaptive subdivision: sampling the distance between mesh node and its limit [8], oriented bounding boxes (directional distance between the limit surface and the interpolation of its three corners) [6, 5], axis aligned bounding boxes (to detect self intersection) [3]. Also various planarity tests have been used to guide adaptivity. Müller et al. used the angle between the limit normal of a vertex and the normals of its adjoining faces [9, 10], Zorin et al. measure planarity at corners and edge midpoints of the next subdivision level [15], Xu and Kondo [14] compute the angle difference between the normal of a face and that of all its neighboring faces.

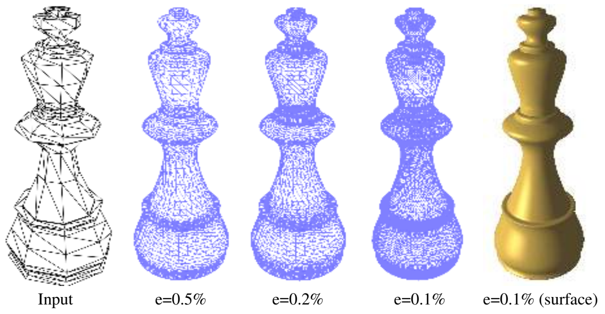


Figure 1. Adaptively subdivided chess piece. (from left to right): input mesh; mesh with maximal error e below 0.5%, 0.2%, 0.1% and the resulting surface for e=0.1%.

A comparison of the surface max-norm bounds and our new bound is shown in Figure 2. Note that sampling and planarity tests do not yield explicit guaranteed maximal bounds between the mesh-polyhedron and the limit surface. Such an error guarantee, however, is crucial for manufacturing applications.

1.2. Overview

Our algorithm is as follows. For each facet of the input mesh, for each coordinate x, y, z, say the x-component, we compute two linear functions p^x and m^x such that $p^x(u,v) \leq x(u,v) \leq m^x(u,v)$. The x-error is $e^x = p^x - m^x$ and the maximum error $\|(e^x, e^y, e^z)\|$ between the subdivision patch and the facet guides the refinement.

In Section 2, we give the details of each step above except for the important detail of parametrization discussed in Section 3. The implementation and the data structures are summarized in Section 4.

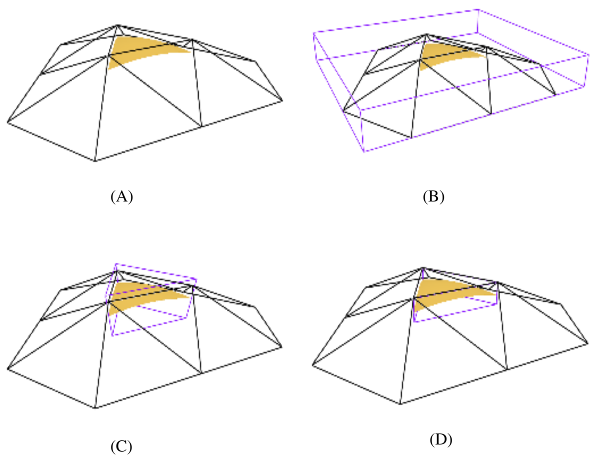


Figure 2. (A) A subdivision patch (shaded) with its local control mesh. (B) Axis-aligned bounding box (dotted) (C) Kobbelt's estimate (dotted) (D) Our estimate (dotted)

2. Bounding Subdivision

This section describes the central contribution of the paper: efficiently determining the maximum distance between for subdivision mesh, viewed as a polyhedron, and the limit surface.

We take advantage of *locality*. Just like NURBS patches, a mesh node (control point) only locally influences the surface shape. Conversely, any point on the surface also depends only on a small submesh. We will *focus on Loop*'s subdivision scheme [7] – it is not difficult to apply the ideas to other subdivision schemes. Loop subdivision is based on a triangle mesh. Each triangle, together with all its direct neighbor triangles, defines a curved triangular surface patch as depicted in 2 (A). Our goal is to approximate the maximum error between the patch and the triangle.

The difficulty is a two-fold. First, it is hard to compute the Hausdorff distance. Second, the limit surface is defined by recursive procedure and lacks a closed form representation. The idea is to parametrize the patch, bound each of the component functions: $x(u,v), y(u,v), z(u,v)$ and combine the component bounds to an overall bound.

We can defer the detailed discussion of the (u,v) parametrization until the next section. For now, we assume that we have a description of $x(u,v), y(u,v), z(u,v)$. In particular, after at most one (local) subdivision, we may assume that at most one of the three vertices of the *center* triangle (corresponding to the patch) has a valence $\mathsf{n} \neq 6$.

2.1. Bounding the Distance

To measure the maximum error between $x(u,v)$ and its linear approximation, we decompose $x(u,v)$ into a linear combination of $\mathsf{n}+6$ functions $b_i(u,v)$

$$x(u,v) = \sum_{i=0}^{\mathsf{n}+5} x_i b_i(u,v).$$

Each $b_i(u,v)$ is the limit, under Loop subdivision, of a local control net $(\mathbf{v}_j, x_j) \in R^3$ with $\mathbf{v}_j = (u_j, v_j)$ abscissae laid out and indexed as in Figure 3 and all $x_j = 0$ except for $x_j = 1$ if $j = i$. The values of (u_j, v_j) we chose and two alternatives are discussed in Section 3.

By linear precision of Loop subdivision, the maximum distance between the lower bound m^x and the upper bound p^x is zero for linear functions. Therefore we can remove a linear function ℓ from the above equation without changing the maximum error:

$$x(u,v) = \ell(u,v) + \sum_{i=3}^{\mathsf{n}+5} d_i b_i(u,v),$$

where ℓ interpolates $(\mathbf{v}_0, x_0), (\mathbf{v}_1, x_1), (\mathbf{v}_2, x_2)$ and d_i is the vertical difference between ℓ and $(\mathbf{v}_i, x_i)$. By interval arithmetic, we obtain an upper and a lower bound of $x(u,v)$ as the follows:

$$\begin{aligned} p^x &:= \ell + \textstyle\sum_{i=3}^{\mathsf{n}+5} \max\{d^i, 0\}\, b_i^+ + \min\{d_i, 0\}\, b_i^-, \\ m^x &:= \ell + \textstyle\sum_{i=3}^{\mathsf{n}+5} \max\{d^i, 0\}\, b_i^- + \min\{d_i, 0\}\, b_i^+. \end{aligned}$$

The error $e(x)$ is bounded by the maximum difference

$$\begin{aligned} e(x) \quad := max\{p^x - m^x\} \quad &= \sum_{i=3}^{\mathsf{n}+5} |d_i| max\{b_i^+ - b_i^-\} \\ &= \sum_{i=3}^{\mathsf{n}+5} |d_i| e(b_i). \end{aligned}$$

In order to efficiently compute $e(x)$, we tabulate the error bound $E_i := e(b_i)$ for each basis function b_i (the linear bound is defined by three scalar coefficients), as well as the barycentric coordinates (s_i, t_i) of $\mathbf{v}_i, i = 3..n+5$ with respect to the vertices $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2$ of the central triangle, i.e.

$$\mathbf{v}_i = (s_i \mathbf{v}_0 + t_i \mathbf{v}_1 + (1 - s_i - t_i)\mathbf{v}_2).$$

Then the distance d_i can be computed efficiently as

$$d_i = x_i - (s_i x_0 + t_i x_1 + (1 - s_i - t_i) x_2).$$

The following short algorithm sums up our process of computing the maximum error. Here input `cf` is a one dimensional array, containing one component of the $n+6$ control points. Variables s, t, E are pre-tabulated 1-dimensional floating point arrays of the values s_i, t_i and E_i described above.

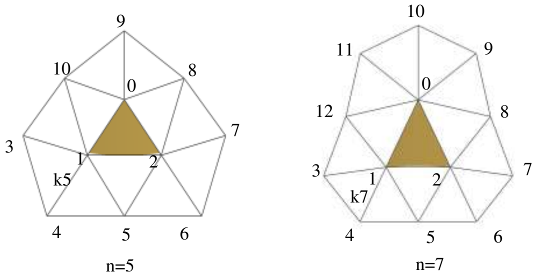


Figure 3. Indices of the vertices $\mathbf{v}_j$ of the (u, v) domain mesh for $n = 5$ and $n = 7$; $\mathbf{v}_0$ is the vertex with valence n. $\mathbf{v}_1$ and $\mathbf{v}_2$ are the other two vertices of the center triangle and have 6 neighbors.

```
float patchError(int n, float* cf) {
  float d, error=0;
  for i from 3 to n+5
    d := cf[i] - (s[i]*cf[0]+t[i]*cf[1]+
      (1-s[i]-t[i])*cf[2]);
    error += E[i]*abs(d);
  end
  return error;
}
```

For each patch, we call this function three times to get e^x, e^y and e^z. If $\|(e^x, e^y, e^z)\| < \epsilon$, the patch is rendered using a single triangle. Otherwise, we subdivide this patch into four subpatches and continue with the subpatches.

3. Parametrization

In this section, we discuss and compare three (u, v) parametrizations of the surface patches, i.e. the layout of the mesh points $\mathbf{v}_i$ defining the parameters of each component. The extra-ordinary node with valence $n \neq 6$ is $\mathbf{v}_0$. The comparison in Table 1 establishes that the carefully constructed, exact parametrization is slightly more efficient than the uniform parametrization and dramatically better than the binary parametrization.

3.1. Exact Parameterization

The parametrization suggested in [13] reproduces linear functions and is defined by the following construction (cf. Figure 3):
1. Set $\mathbf{v}_0 = 0$, the origin of the (u, v) plane.
2. The direct neighbors $\mathbf{v}_i$ of $\mathbf{v}_0$ form a regular unit $n-$gon.
3. Extend the edge $\mathbf{v}_0\mathbf{v}_1$ and $\mathbf{v}_0\mathbf{v}_2$ by k_n to get $\mathbf{v}_4$ and $\mathbf{v}_6$.
4. $\mathbf{v}_5$ is the average of $\mathbf{v}_4$ and $\mathbf{v}_6$.
5. $\mathbf{v}_3, \mathbf{v}_7$ are the reflection of $\mathbf{v}_5$, across $\mathbf{v}_0\mathbf{v}_4$ and $\mathbf{v}_0\mathbf{v}_6$, respectively.

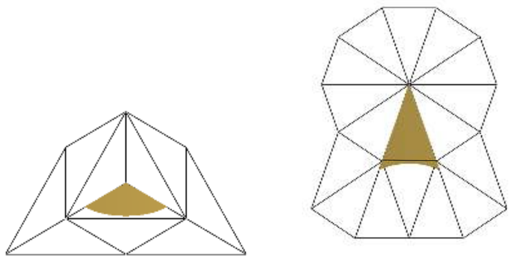


Figure 4. The domains of uniform parametrization for n=3 and n=10.

k_n is defined by the following formula where $c := \cos \frac{\pi}{n}$:

$$k_n := \begin{cases} -4(c^2-2)/(1+2c^2)-1 & \text{if} \quad n \geq 6, \\ -6(2c^2-7)/(15+2c^2)-1 & \text{if} \quad n < 6, \end{cases}$$

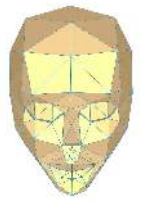
We can now obtain any patch $x(u, v)$ by subdividing the control net with vertices $\mathbf{v}_i, x_i$. The shaded areas in Figure 3), which depend on the valence of $\mathbf{v}_0$, is the limit of the $\mathbf{v}_i$-mesh under Loop subdivision. Note that The limit of the mesh with vertices $\mathbf{v}_j$ are the shaded areas in Figure 3. Note that these areas fit exactly into the center triangle.

3.2. Uniform Parametrization

With the same construction as above, but k_n=1 for all n we obtain the uniform parametrization. The bottom boundary of the domain will either, for $n < 6$, pull back from the boundary of the center triangle or, for $n > 6$, push out of the triangle (see Figure 4) and therefore *requires careful extrapolation to safeguard the bound.*

3.3. Binary Parametrization

The parametrization proposed in [11, 12] associates the vertices of the mesh under subdivision with a binarily refined grid. While this allows deducing the number of subdivision steps from the (u, v) position, it *does not yield linear precision*! This means, we cannot pull out the linear term ℓ from the expansion of $x(u, v)$ and cancel it when we compute the distance between m^x and p^x. Therefore, as for axis-aligned bounding boxes, the relative position of the object in space influences the local error and the error estimates can be dramatically worse as illustrated in Table 1. The vertex based method [8] by measuring the distance between the vertices and their limit positions is also listed in the Table 1 as "v-error".



model	Head (200 triangles)		
error	0.5%	0.2%	0.1%
exact	4,766	11,279	26,231
uniform	4,856	11,417	26,492
binary	291,782	n/a	n/a
v-error	5,966	12,524	28,628

model	Venus (1418 triangles)		
error	0.5%	0.2%	0.1%
exact	6,764	19,295	36,851
uniform	7,088	20,330	38,198
binary	140,879	n/a	n/a
v-error	8,255	22,559	39,602

Table 1. Numbers of resulting triangles for different error measurements. Rows "exact", "uniform", "binary" indicate three different parametrizations. Row "v-error" is for the method based on measuring the distance between the vertices and their limit points. 'n/a' indicates out of memory.

4. Adaptive Subdivision

We support the adaptive subdivision process by a forest of balanced quad trees. Balancing according to [1], assures that neighboring faces differ only by one step of subdivision. The quad trees of this *balanced adaptive subdivision*, one tree per original facet, are linked so that we can directly access the neighbors for any given face at any level and without ascending the tree to the common parent (herein lies the difference to [15]).

The data structure has the following entries (cf. Figure 5).

- Each internal node has four pointers pointing to its children. The order and orientation of the children is shown in Figure 5 *left*.
- Each face, either internal or leaf, stores its three neighboring half-edges, as shown in Figure 5 *right*.
- Each half-edge is stored as pair $[f, i]$, where f is the face and and i is an integer from 0 to 2 indicating the index of the edge.
- Each face has an integer indicating the subdivision level.

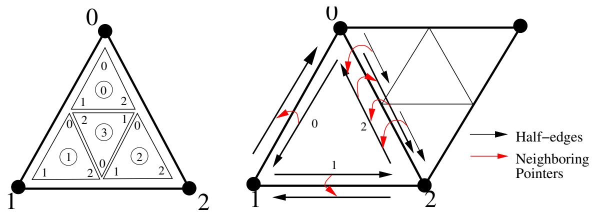


Figure 5. Data structure. (left) The indices of the parent face and its 4 children. (right) Halfedge pointers for fast neighboring access.

Now, for any leaf face f, we can access its neighbors via the halfedge $[n, i]$ in constant time. If the face n is a leaf, then it is the neighbor of f on side i. Otherwise, its children i and ($i + 1$ mod 3) are the two neighbors to f on side i. Due to tree balancing, they must be leaf nodes. In C code, our data structure is as follows:

```
struct patch{
  // subdivision level
  int level;

  // children
  struct patch* children[4];

  // neighbors
  struct patch* neighbors[3];
  // neighboring edges
  int    neighidxs[3];
}
```

The field entries can be filled by a depth first traversal of the quadtree forest. The time to build the data structure is therefore $O(N)$ where N is number of leaf faces.

4.1. Gap Prevention

When two neighboring faces have different subdivision levels, a gap appears between them as illustrated in Figure 6 *left*. Rendering such a mesh results in irritating visual artifacts. Moreover, gaps spell serious trouble for the mesh processing and finite element computations.

We follow the standard recipe for removing the gaps by splitting the patch on the coarser side as illustrated in Figure 6 *right*. Such a process is done top to down recursively for each pair of original neighboring faces. The time complexity of this step is also $O(N)$ where N is number of leaf faces.

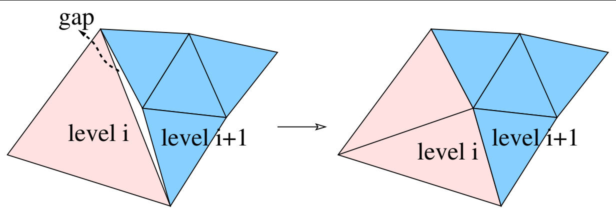


Figure 6. (left) A gap between adjacent faces at different levels of subdivision. (right) The gap is removed by splitting the coarse triangle.

5. Extensions

5.1. View dependent adaptation

So far our focus was on true error control in 3 dimensions. For rendering, we can project our error vectors (e_x, e_y, e_z) into the view plane and compute the projected size as adaptive refinement criterion.

5.2. Creases and Boundaries

Subdivision surfaces become a lot more exciting with Pixar's semi-sharp creases ([2, 4]): a few steps of smoothing in only one direction followed by smooth subdivision. A simple way to avoid special bounds for the different crease configurations is to locally perform the uni-directional subdivision steps regardless of the error and then apply the regular bounds. Alternatively, since we only use upper and lower bounds, we need not generate new tables for every combination of two subdivision rules: if one rule results consistently in higher values than the other, say a uni-directional rule and a generic subdivision rule, we bound the upper function above and the lower function below to enclose the range of combinations. The same technique applies to boundaries that follow a spline curve.

6. Results

In Figure 7, the 500-facet deer model is subdivided to meet error bounds of 0.5% and 0.1%, respectively. The total times to subdivide the model, including the error estimation, are 172ms and 906ms, respectively. The number of triangles generated are 4834 and 25980.

In Figure 8, we raytrace the subdivided mesh, on a P4 2.8G PC with 1G RAM. The total time to raytrace the input cat model of 671 triangles is 8 seconds. The time to adaptively subdivide the model to within a 0.2% bound and raytrace is 9 seconds. The total time to uniformly subdivide the model to the same bound and raytrace is 12 seconds.

7. Conclusion and Future Work

We presented a tight estimate on the maximum distance between a subdivision surface and its linear approximation and applied it to adaptive subdivision. The computation is efficient and the algorithm is easy to implement.

We are currently applying the approach to Catmull-Clark surfaces and develop a similar tight bound for patch normals to help silhouette and self-interference detection.

Acknowledgements:

We thank Jianhua Fan for her help to implement uniform Loop subdivision in Pov-Ray. Pov-Ray is a free ray-tracing package downloadable from http://www.povray.org. The original deer and cat models are achieved from 3D search engine at: http://3d-search.iti.gr/3DSearch. We used the *qslim* package that to simplify the deer model to 500 faces.

A. Convergence of the error estimate

We need to show that $e(x) \to 0$ under subdivision where

$$e(x) := \sum_{i=3}^{n+5} |d_i| e(b_i).$$

Since the terms $e(b_i)$ are constant, it suffices to prove $d_i \to 0$ where

$$d_i := s_t(x_i - x_0) + t_i(x_i - x_1) + (1 - s_i - t_i)(x_i - x_2).$$

This follows from $(x_i - x_j) \to 0$ under subdivision.

References

[1] M. Berg, M. Kreveld, O. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Heidelberg, New York, 1997.

[2] T. DeRose, M. Kass, and T. Truong. Subdivision surfaces in character animation. In M. Cohen, editor, *Siggraph 98*, pages 85–94, 1998.

[3] E. Grinspun and P. Schröder. Normal bounds for subdivision-surface interference detection. In T. Ertl, K. Joy, and A. Varshney, editors, *Proc Visualization*, pages 333–340. IEEE, 2001.

[4] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle. Piecewise smooth surface reconstruction. *Computer Graphics*, 28(Annual Conference Series):295–302, July 1994.

[5] L. Kobbelt. Tight bounding volumes for subdivision surfaces. In B. Werner, editor, *Pacific-Graphics'98*, pages 17–26. IEEE, 1998.

[6] L. Kobbelt, K. Daubert, and H.-P. Seidel. Ray tracing of subdivision surfaces. In *Rendering Techniques '98 (Proceedings of the Eurographics Workshop)*, pages 69–80. Springer-Verlag, 1998.

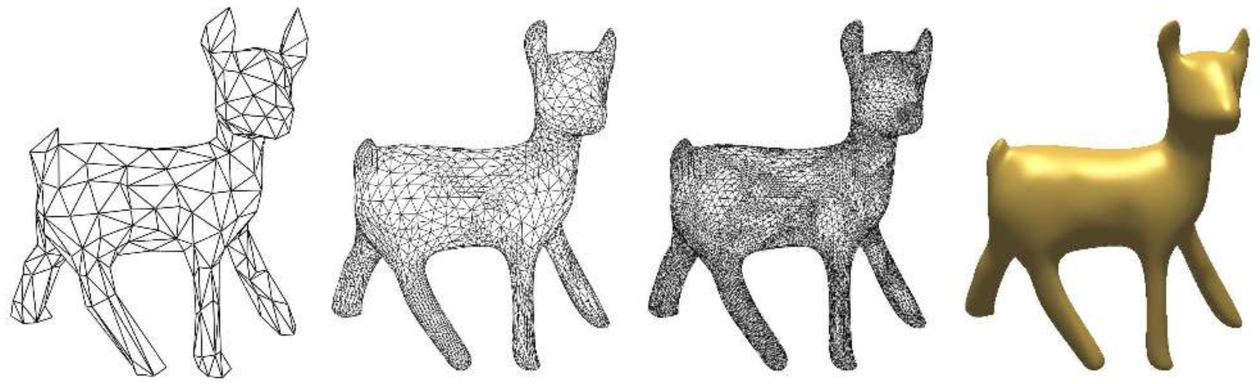


Figure 7. (from left to right) input; e=0.5%; e=0.1%; and the surface with e=0.1%.

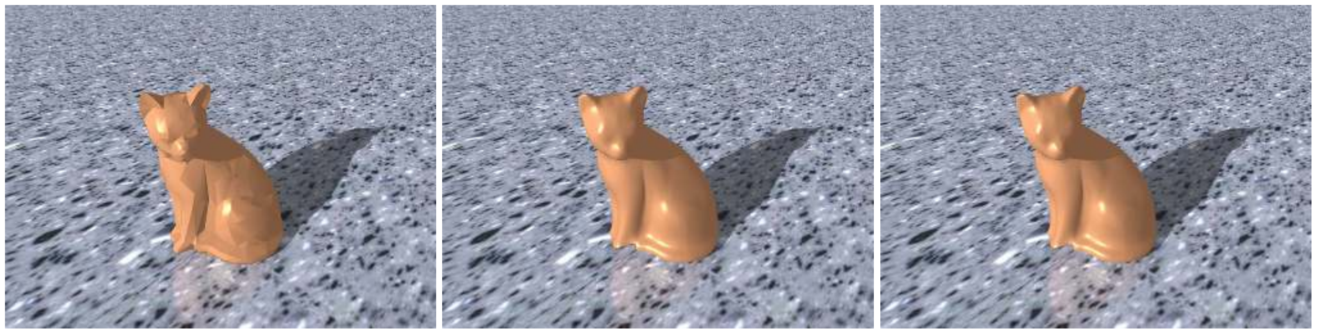


Figure 8. Ray-traced images at 800x600 resolution. (left) Input mesh (671 triangles). Ray-tracing time: 8s. (middle) Adaptive subdivision with e=0.2%. Resulting number of triangles = 9662 and maximum subdivision level: 4. Ray-tracing time: 9s. (right) uniformly subdivided 4 times. Resulting number of triangles = 171776. Ray-tracing time: 12s.

[7] C. T. Loop. Smooth subdivision surfaces based on triangles, 1987. Master's Thesis, Department of Mathematics, University of Utah.

[8] H. Müller and R. Jaeschke. Adaptive subdivision curves and surfaces. In F.-E. Wolter and N. M. Patrikalakis, editors, *Proceedings of the Conference on Computer Graphics International 1998 (CGI-98)*, pages 48–58, Los Alamitos, California, June 22–26 1998. IEEE Computer Society.

[9] K. Müller and S. Havemann. Subdivision surface tesselation on the fly using a versatile mesh data structure. In M. Gross and F. R. A. Hopgood, editors, *Computer Graphics Forum (Eurographics 2000)*, volume 19(3), 2000.

[10] V. Settgast, K. Müller, F. Fuenfzig, and D. Fellner. Adaptive tesselation of subdivision surfaces. *Computers and Graphics*, 28(1):73–78, Feb. 2004.

[11] J. Stam. Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. In M. Cohen, editor, *SIGGRAPH 98 Proceedings*, pages 395–404. Addison Wesley, 1998.

[12] J. Stam. Evaluation of loop subdivision surfaces, Aug. 27 1999.

[13] X. Wu and J. Peters. Interference detection for subdivision surfaces. *Computer Graphics Forum, Eurographics 2004*, 23(3):577–585, 2004. acceptance rate ca 18%.

[14] Z. Xu and K. Kondo. Local subdivision process with doo-sabin subdivision surfaces. In *Shape Modeling International, Proceedings*, 2002.

[15] D. Zorin, P. Schröder, and S. Sweldens. Interactive multiresolution mesh editing. In T. Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 259–268. ACM SIGGRAPH, Addison Wesley, Aug. 1997. ISBN 0-89791-896-7.