EUROGRAPHICS '2000 / M. Gross and F.R.A. Hopgood
(Guest Editors)

Volume 19, (*2000*), *Number 3*

Subdivision Surface Tesselation on the Fly using a versatile Mesh Data Structure

Kerstin Müller and Sven Havemann

Institute of ComputerGraphics, TU Braunschweig, Germany

Abstract

Subdivision surfaces have become a standard technique for freeform shape modeling. They are intuitive to use and permit designers to flexibly add detail. But with larger control meshes, efficient adaptive rendering techniques are indispensable for interactive visualization and shape modeling. In this paper, we present a realization of tesselation-on-the-fly for Loop subdivision surfaces as part of a framework for interactive visualization.

1. Introduction

Polygonal meshes have received considerable attention over the last years, due to a progress in diverse fields, just to mention progressive meshes[2], multiresolution analysis[10], or mesh compression. But the world is not only made of polygons. The need to represent complicated freeform shapes finally led us – like many others – to the subdivision surface representation. In fact, subdivision surfaces can be used to *reduce* freeform modeling to mesh modeling. Overcoming the well-known restrictions of tensor product surfaces, with subdivision surfaces virtually every polygonal mesh can be used as control mesh for a freeform object. Rapidly converging subdivision rules quickly produce accurate tesselations at any resolution needed, and designers can freely manipulate the control mesh to add local detail, including fancy features like sharp creases, starting and ending within an otherwise smooth surface.

While subdivision surfaces are a powerful tool in the design of complex shapes, this complexity poses a serious problem in interactive applications. For the most popular subdivision schemes, the number of faces increases by a factor of four with each subdivision step. This exponential growth of the number of faces with the *subdivision depth* makes an accurate uniform tesselation of a moderately big control mesh impractical. With a useful number of subdivision steps in the range of three to five, tesselation results in 64 to 1024 faces *per toplevel face* of the control mesh.

However, most of these faces are invisible. Instead of holding a complete tesselation in memory and to use optimized simplificiation and culling algorithms for maintaining interactive display rates, an alternative approach is a *tesselation-on-the-fly*, only producing faces actually needed for a given scene and camera configuration. For example, covering a 1024×768 display with only 20k triangles will, in average, result in triangles containing 40-65 pixels each, which are 9 - 11 pixels in diameter. These triangles should be small enough to produce any perceivable detail for interactive applications. Interactive display of 20k triangles, however, is now in fact possible with low-cost 3D hardware. The only problem remaining is to find the *appropriate* triangles to represent a scene.

At this point, subdivision surfaces can actually be seen as an adaptive rendering method for a mesh. Given a large control mesh, the goal of interactive subdivision surface rendering is to tesselate visibly important parts with higher resolution than less important parts. But as a rendering method is not an integral feature of a mesh, it seems reasonable to manifest this orthogonality also in the design of data structures. This not only allows to combine different mesh representations with different subdivision schemes. As another advantage, the library of mesh data structures and algorithms can be independently extended, plugging in interactive subdivision surface rendering whenever needed.

2. Motivation

Fast interaction with large-scale three-dimensional environments typically requires a combination of different techniques. Famous approaches for efficient interaction with polygonal models are *progressive meshes*[2] (PMs), specifically the version for view-dependent refinement[3], and *hier-*

© The Eurographics Association and Blackwell Publishers 2000.Published by Blackwell Publishers, 108 Cowley Road, Oxford OX4 1JF, UK and 350 Main Street, Malden, MA 02148, USA.

delivered by
EUROGRAPHICS DIGITAL LIBRARY
www.eg.org diglib.eg.org

archical dynamic simplification[6] (HDS). In general, frameworks for fast 3D interaction are composed of different building blocks:

- A multiresolution model representation scheme
- Measures for visibility and display accuracy
- Fast on-line update of the active scene part

The *active* scene is a coarser version of the original scene, where visibly unimportant, distant or very small, parts of the model have been collapsed, according to the accuracy measure. The active part of a scene has to be updated in every frame, making use of *temporal coherence* in the position and orientation of the viewer. This implies that fast evaluation of accuracy measures and fast update of the active scene database are crucial prerequisites for maintaining an interactive frame rate of at least 15-20 frames per second.

While these approaches work well for triangular worlds, polygonal representations have important drawbacks in that the highest model resolution is limited a priori by the accuracy of the original scene. The obvious solution is to switch to a different model representation, replacing polygonal primitives by some sort of freeform surface. A first, important step into this direction is the work of Subodh Kumar, who showed that tesselation-on-the-fly can cope with models represented by trimmed NURBS. Incrementally converting NURBS to Bézier patches, that are subsequently tesselated, about 10 fps were possible with a scene complexity of 5,000 Bézier patches[4].

A disadvantage of using NURBS though is that they require their control mesh to have a regular structure. Furthermore, stitching NURBS patches together is not straightforward, since maintaining geometric continuity between neighbouring patches severely constraints the positions of their control vertices.

These shortcomings can be remedied by using subdivision surfaces. Loop subdivision surfaces, permit *any* manifold triangle mesh to serve as control mesh for a freeform object. And by using the Catmull/Clark scheme, any polygonal object with *convex* faces of *arbitrary degree* can be taken to represent a complicated freeform shape. Surface continuity is built into the subdivision rules, but designers don't have to do without the features they know from trimmed NURBS. With subdivision surfaces, it is even possible to let a sharp edge start and end in an otherwise smooth surface.

The ability of subdivision surfaces to digest almost any polygonal mesh now opens the possibility to apply the aforementioned sophisticated frameworks for interaction with complex polygonal environments to the subdivision surface control mesh.

One important building block of an integrated realtime-rendering system for very large freeform worlds is the ability to tesselate a given subdivision surface patch extremely fast to any resolution needed. Second, neighbouring faces with different subdivision depths have to be zipped together along their common border, so that no cracks appear between them.

Tesselation-on-the-fly may then be used together with a multiresolution representation of the control mesh to develop fully integrated schemes. Spatial coherence will then not only be used for updating the active part of the control mesh, but also for caching already tesselated faces, incrementally adjusting tesselation quality whenever needed.

3. Subdivision Schemes

As noted before, in principle any polygonal mesh can serve as control mesh for a subdivision surface. However, focusing on the most popular subdivision schemes from Loop[5] and Catmull and Clark[1], a fundamental requirement is that the mesh be locally 2-manifold everywhere, but may have boundaries.

As with both schemes subdivision implies shrinkage, subdividing a face produces sub-faces that share no 3D corner position with their parent face. Fortunately, for both schemes, closed expressions exist to project a given vertex, that is produced at any level of subdivision, onto the limit surface of the infinite subdivision process[8,9]. With the same mathematical tool used to derive these formulae, namely eigenanalysis of the subdivision matrix, closed expressions for the tangent vectors of the limit surface can be derived, thus the limit surface normal can be obtained analytically.

In principle, any subdivision scheme with closed expressions for limit points and limit-surface normals can be used with the algorithmic framework for tesselation-on-the-fly. However, Loop subdivision surfaces are chosen here as an example for a triangle-based scheme. To combine a given mesh representation with Loop subdivision, faces with degree higher than three have to be triangulated in a preprocessing step. This triangulation has to be done with care, because the shape of the freeform surface is not invariant under different triangulations.

3.1. Loop Subdivision Surfaces

Performing one subdivision step on a triangle will result in four subtriangles (cf. Fig 9). The vertices of these sub-faces can be classified into edge and vertex points of the parent triangle. They are computed using affine combinations of vertices from the parent face and its 1-neighborhood, using so-called *vertex masks* and *edge masks*.

For a level l vertex $v_0^{(l)}$ of valence n, i.e. with neighbors $v_1^{(l)},\dots,v_n^{(l)}$, the new vertex of level $l+1$ and its new neighbors are computed using the vertex and edge rules:

$$v_0^{(l+1)} = \frac{w_0(n)v_0^{(l)} + v_1^{(l)} + \dots + v_n^{(l)}}{w_0(n)+n} \quad (1)$$

$$v_i^{(l+1)} = \frac{3v_0^{(l)} + 3v_i^{(l)} + v_{i-1}^{(l)} + v_{i+1}^{(l)}}{8}, i = 1,\dots,n \quad (2)$$

© The Eurographics Association and Blackwell Publishers 2000.

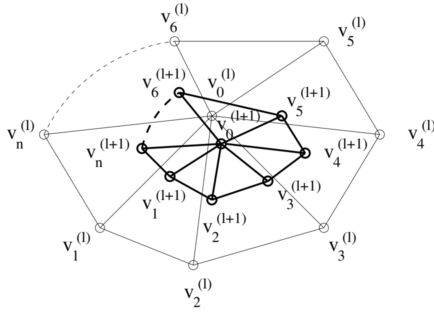


Figure 1: *A regular triangle and its 1-neighborhood*

All indices are taken modulo n, and $w_0(n) = n/a(n) - n$ with $a(n) = \frac{5}{8} - (3 + 2\cos(2\pi/n))^2/64$. The corresponding masks are shown in Fig. 2.

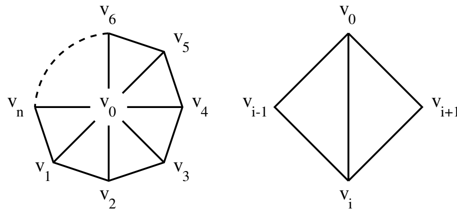


Figure 2: *Vertex mask and edge mask*

If this process is iterated recursively, the mesh converges to a smooth *limit surface*, and every vertex of the toplevel control mesh corresponds to one point on the limit surface. The position of this point $v^{(\infty)}$ as well as the tangent vectors u_1 and u_2 of the limit surface can be computed using an affine combination of the vertex and its neighbors, with $w_0^{(\infty)}(n) = 3n/8a(n)$ and $t_i = \cos(2\pi i/n)$:

$$v^{(\infty)} = \frac{w_0^{(\infty)}(n) v_0^{(0)} + v_1^{(0)} + \ldots + v_n^{(0)}}{w_0^{(\infty)}(n) + n} \quad (3)$$

$$u_1 = \frac{\sum_{i=1}^{n} t_i(n)\, v_i^{(0)}}{\sum_{i=1}^{n} t_i} \quad (4)$$

$$u_2 = \frac{\sum_{i=1}^{n} t_{i-1}(n)\, v_i^{(0)}}{\sum_{i=1}^{n} t_{i-1}} \quad (5)$$

4. Tesselation on the fly

A concise, straightforward, yet naïve method to visualize a subdivision surface is sketched in the following recursive procedure. This algorithm will either blow up the mesh exponentially, saving intermediate results as new faces and vertices, or it computes many vertices several times. But even worse, individual triangles are sent down the graphics pipeline instead of triangle strips, which is a waste of bus bandwidth. Finally, tesselation charges the processor with a lot of work, and in multiprocessor environments, work should be split over several largely independent threads.

```
procedure subdiv(face, depth):
  if (depth > 0) :
    create subfaces 0,1,2,3 of face
    subdiv(subface0, depth − 1)
    subdiv(subface1, depth − 1)
    subdiv(subface2, depth − 1)
    subdiv(subface3, depth − 1)
  else
    compute limit surface points and normals
    render triangle (to OpenGL)
```

In summary, an algorithm for tesselation-on-the-fly should ideally have the following features:

- no dynamic storage allocation
- producing triangle strips
- parallelizable, and
- the tesselation doesn't write to the mesh

In a paper from Kari Pulli and Mark Segal, a tesselation method with these properties was presented, called the *sliding window method*[7]. It is based on the observation that subdividing a *pair* of adjacent triangles at a time is more efficient, because this produces an upper and a lower row of triangles. Iterating this scheme produces triangle strips that double in length with each subdivision level, which can directly be rendered using vertex and normal arrays from OpenGL. The arrays used for one triangle strip may subsequently be overwritten for computing and rendering the next strip or row.

What makes this idea very attractive to an interactive rendering system is the fact that the same scheme can be applied to quad-based subdivision, for instance using the Catmull-Clark scheme. The sliding window method actually operates on quadrangles, because in a preprocessing step, the mesh is partitioned into triangle pairs. Consequently, the same algorithmic framework can be re-used for a Catmull-Clark type of subdivision surface rendering, by simply plugging in different subdivision rules. In fact, using quadrangle strips is almost identical to using triangle strips on the OpenGL level, and it is even possible to use the same type of OpenGL vertex arrays for both schemes to optimize performance.

This generic framework for subdivision surface tesselation realizes an OpenGL-like *geometry engine* by itself, which was the API already proposed by Pulli and Segal: One triangle pair at a time together with its 1-neighborhood is read from the mesh and given to a dedicated tesselation thread working independently. When finished, the main thread, which in standard OpenGL 'owns' the OpenGL context, only needs to collect the results and to stripwise hand the computed vertex arrays over to the OpenGL driver.

With each triangle pair processed individually, it becomes possible to let the subdivision depth vary from face to

© The Eurographics Association and Blackwell Publishers 2000.

face, thereby implementing *adaptive tesselation-on-the-fly*, instead of a much simpler *uniform* tesselation. But then, heuristics are needed telling which approximation accuracy is needed for each of the triangle pairs. However, this strategy is not part of the tesselation engine and can be worked on independently.

The sliding window method is a good candidate for implementing subdivision surface tesselation in hardware. But this requires simple strategies for avoiding cracks between neighboring pairs with different tesselation depths. Both problems will be addressed in the next sections.

4.1. Heuristics for the Choice of the Subdivision Depth

Tesselation in realtime can only be achieved if higher accuracy, i.e. greater subdivision depth, is assigned to visibly important faces. In this context, however, accuracy is very coarse-grained, as the number of triangles in the tesselation increases by powers of four: For all cases except extreme close-ups, the subdivision depth ranges only over seven integral values, from -1 (cull face away) to 5 (1024 sub-triangles). Several properties influence importance:

- visibility
- distance to the eye point
- projected size of the limit surface
- curvature of the limit surface

These criteria are not independent from each other. The limit surface from distant faces usually occupies fewer pixels when projected on the image plane than from faces nearby. A face near the object's silhouette may have a high curvature, but it leaves hardly a trace in the z-buffer. And, of course, the effort to subdivide backfaces or faces with occluded limit surface is completely wasted.

Given only the control mesh, it is hardly possible to compute these measures exactly during the time between consecutive frames. So, appropriate heuristics have to be found. While the projected size of a triangular face can be computed fast, in some cases this can be found to be a very bad estimate for the size of its limit surface's projection. On the other hand, the convex hull from the vertices of a face's 1-neighbourhood is a guaranteed bounding volume for the limit surface, but it is not tight. As the quality parameter is coarse grained, we chose a very rough estimate for the projected size of a patch, namely the size of a bounding sphere around the triangle vertices, divided through the distance from the viewpoint. This strategy is called *project-sphere*.

Besides view-dependent measures of visual importance, there are also object-based measures. To estimate the accuracy needed for a tesselation to adequately represent the limit surface's shape, *normal cones* can be used. To measure the variation of the normal over the patch, one main normal and an angle are computed that bounds the deviation of all normals over the patch from the main normal. On the one hand, this permits to classify a patch as either completely front-facing or backfacing, otherwise it is tagged a visibly important silhouette patch[6]. But the deviation angle can also be taken as a rough estimate for the *curvature* of a patch, directly influencing the subdivision depth necessary for an accurate tesselation.

Computing the normal cone of a subdivision surface, however, is neither elementary nor inexpensive. Computing the curvature only in some points, e.g. the corners, of the curved limit surface is much simpler and faster, still in practice it yields good-looking results. The subdivision depth can in fact directly be computed from the curvature by specifying in advance a maximal angle α that is tolerated between a limit vertex normal and its incident sub-triangle: Each triangle f of the control mesh with vertices v_1, v_2, v_3 is assigned the minimal depth value l so that

$$N_i \cdot N_i^{(l)} < \cos\alpha, \quad i = 1, 2, 3,$$

where N_i is the limit surface normal of vertex v_i, and $N_i^{(l)}$ is the face normal of the sub-face of f incident to a vertex corresponding to v_i on subdivision level l.

This computation is done by looping once through all vertices. Each vertex and its neighborhood are copied to a vertex mask-like structure (cf. Fig. 2, left). The limit surface normal is computed, and if the vertex is back-facing, its depth value is set to -1. Otherwise, subdivision is performed in-place using the vertex mask, until all normals from incident sub-triangles have entered the normal cone, as depicted in Fig. 3.

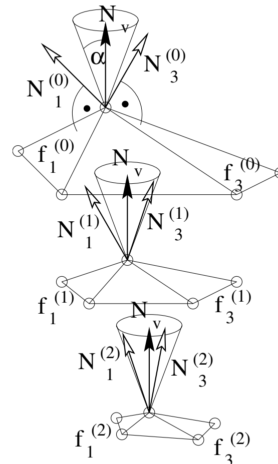


Figure 3: *The normal cone over two subdivision steps*

The prescribed deviation α can be varied according to the classification of a vertex. Faces where all three corner normals point backwards are classified as back-patches and are not displayed at all. Completely front-facing patches do not need a high-resolution tesselation, and silhouette patches are

© The Eurographics Association and Blackwell Publishers 2000.

detected by comparing the vertex normal cones. Unfortunately, as Loop subdivision surfaces are derived from cubic splines, counterexamples for this strategy can be found. Visible artifacts, however, have not been observed in practice.

4.2. Pairing strategy

To use the sliding window method efficiently, a triangle mesh has to be partitioned into triangle pairs. This *pairing step* can be performed by a greedy algorithm. It was proven correct by Pulli and Segal[7], and though it may not be optimal, it behaves well in practice. It can even be modified to take into account varying face depths, by preferably grouping triangles with equal depth values.

In the beginning, each triangle is inserted into one of four disjoint sets $S_0,\dots,S_3$ containing the *free triangles*. A triangle is called free iff it is not paired yet. Boundary vertices may have only one or two neighbors, but for a closed mesh, all triangles are inserted into set S_3 in the beginning.

S_0 : 0 or 1 free neighbors
S_1 : 2 free neighbors, one of them in $S_0 \cup S_1$
S_2 : 2 free neighbors, both of them in S_3
S_3 : 3 free neighbors

Now, the algorithm attempts to create pairs, taking triangles from the nonempty set with the highest priority. S_0 has the *highest* and S_3 the *lowest* priority in this algorithm.

```
while S0,...,S3 are not empty do
1. Take a triangle f from the nonempty set
   with the highest priority
2. LOOK FOR A FREE PARTNER g FOR f:
   if f has free neighbors then
      if f has a neighbor in S0 then take it as g
      else
         if f has neighbors with the same depth then
            let g be the one with the highest priority
         else
            let g be the neighbor with the highest priority
            f.depth := g.depth := max(f.depth,g.depth)
      remove f,g from S0,...,S3
      insert (f,g) into the pair list
   else
      remove f from S0,...,S3
      mark f as unpaired singleton
3. INCREASE NEIGHBOR PRIORITIES:
   move neighbors of f (and g) from S3 to S2
   move neighbors of f (and g) from S2 and S1 to S0
   if any triangle moved from S3 to S2 then
      move all triangles from S2 with neighbors in
      S0, S1 or S2 to S1
```

As a result, a partition of the mesh into triangle pairs is created, where both triangles in a pair have the same subdivision depth, possibly modifying previously assigned depth values. With this modificiation, still very few singletons are created, typically only about one percent. Now, the list of triangle pairs is ready to be delivered to the sliding window method for efficient tesselation.

4.3. The sliding window method

The sliding window method operates on one triangle pair at a time. Unpaired singleton triangles are assigned a dummy partner which is not visualized. As typically only few of them exist, this doesn't introduce a performance problem. In a multithreaded environment, several instances of the algorithm can be active in parallel.

For a given triangle pair, the four corners and the vertex positions from the – possibly irregular – 1-neighborhood are collected from the mesh, and copied to different arrays of 3D points. These arrays are of a fixed size that only depends on the level of subdivision they are used with. Consequently, they are allocated statically, once per thread, and can be reused for each triangle pair that is to be tesselated. Fig. 4 shows the freshly filled arrays in level $l = 0$, right before the start of tesselation.

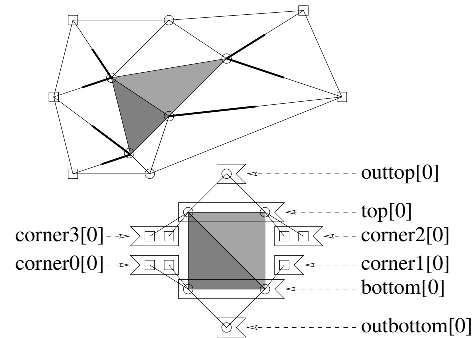


Figure 4: *A pair of triangles with its neighborhood is copied to arrays of 3D vectors*

The core algorithm is shown in the following pseudocode.

```
procedure subdiv(level, top, bottom, outtop, outbottom,
                 corner0, corner1, corner2, corner3)
   if (level>0)
      Perform one subdivision step and
      save the new points in the arrays of level [level].
      SUBDIVIDE UPPER ROW:
      subdiv(level-1, top, middle, outtop, bottom,
             mcorner0, mcorner1, corner2, corner3)
      SUBDIVIDE THE LOWER ROW:
      subdiv(level-1, middle, bottom, top, outbottom,
             corner0, corner1, mcorner1, mcorner0)
   else :
      compute the limit points and normals:
      render triangle-strip (to OpenGL)
```

© The Eurographics Association and Blackwell Publishers 2000.

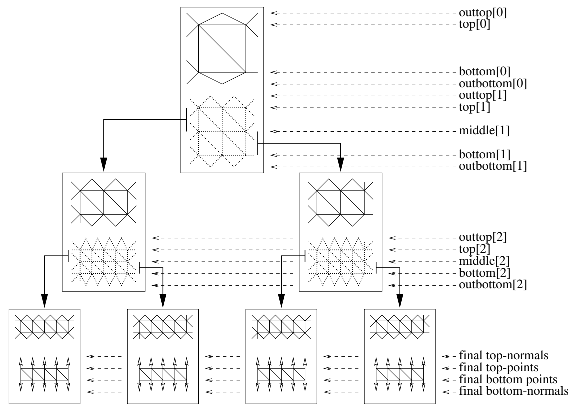


Figure 5: *Two levels of subdivision with the sliding window method. For better readability, the corner arrays are left out.*

The corner vertices of the two faces are held in arrays top and bottom, while the 1-neighborhood is copied to outtop, outbottom and corner0,…,corner3. Newly computed points from each level are saved in eleven arrays that are indexed by the subdivision level: top, middle, bottom, outbottom, outtop, corner0,…,corner3. The arrays mcorner0, mcorner1 hold the neighbors left and right of array middle. The structure of the computation is visualized in Fig. 5. The size of these arrays essentially doubles from level to level.

4.3.1. Space and time consumption

In order to use statically allocated arrays of fixed size, the maximal vertex valence has to be limited. A value for n_{max} of 50 was sufficient in practice, as higher valences only occur in special cases.

As mentioned before, eleven arrays are needed for each level l: Five arrays (outtop, top, middle, bottom, outbottom) of length 2^l+1, and six arrays, corner0,…,corner3 and mcorner0, mcorner1 of length n_{max}. Thus, subdividing m levels uses the following number of 3D vectors:

$$\sum_{l=0}^{m}(5\cdot(2^l+1)+6\cdot n_{max}) = 5\cdot(2^{m+1}-1)+5m+6m\cdot n_{max}$$

This is the complete number of 3D vectors necessary to perform all subdivisions. The results of the computation, the limit points from top and bottom, and the limit surface normals are stored in an interleaved fashion to an OpenGL vertex array and a normal array. This can efficiently be rendered as a triangle strip with normals. So, additionally two arrays of length $2\cdot(2^m+1)$ are needed:

$$5\cdot(2^{m+1}-1)+5m+6m\cdot n_{max}+4\cdot(2^m+1) = 7\cdot 2^{m+1}-1+5m+6m\cdot n_{max}$$

A single pair of triangles is rendered using 2^m+1 triangle strips alltogether, so $n=(2^m+1)^2$ points and the same number of normals have to be computed. Taking the time needed to calculate a vertex as constant (in fact, it depends on its valence), the number of calculations is:

$$\sum_{l=0}^{m} 2^l\cdot\Big(5\cdot(2^l+1)+4\cdot n_{max}\Big) = \frac{5}{3}\cdot 2^{2\cdot(m+1)}+(5+4\cdot n_{max})\cdot 2^{m+1}-\frac{20}{3}-4\cdot n_{max}$$

But this is linear in the number of limit points:

$$O\Big(\frac{5}{3}\cdot(4\cdot(\sqrt{n}-1)^2) +(5+4\cdot n_{max})\cdot 2\cdot(\sqrt{n}-1)-\frac{20}{3}-4\cdot n_{max}\Big) = O(n)$$

5. Crack Prevention

Special care has to be taken for borders between triangle pairs with different subdivision depths. Fig. 6 shows a typical setting: Each of the grey triangles has two or more black neighbors. The singleton triangle in the lower left for example is subdivided only once, while its right neighbor has depth four. Cracks appear because the grey triangles have more degrees of freedom at the border than the black ones.

To repair these cracks, a tesselation scheme producing triangle fans can be used, as shown in Fig. 7. First, a grey triangle is subdivided to the maximal neighbor depth, collecting only vertices on the triangle border according to the respective neighbor subdivision levels. Then, these are used to create a triangle fan around the triangle center that is added to the tesselation.

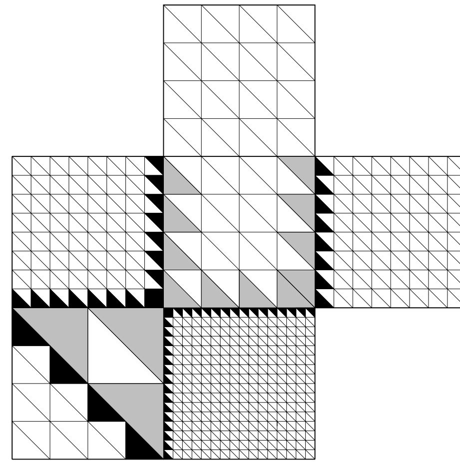


Figure 6: *A part of a mesh with cracks.*

Unfortunately, taking the triangle point with barycentric coordinates $(\frac{1}{3},\frac{1}{3},\frac{1}{3})$ for a center vertex, produces unwanted nonconvexities. An example is given in Fig. 8. Significantly

© The Eurographics Association and Blackwell Publishers 2000.

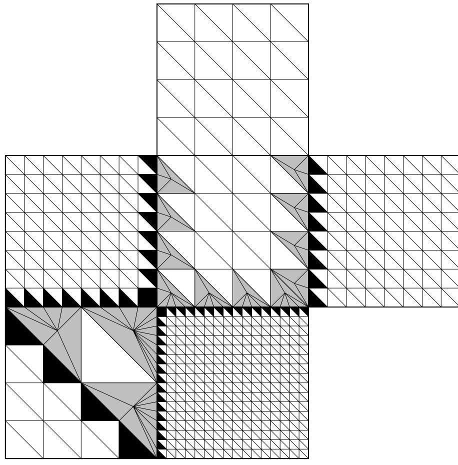


Figure 7: *A part of a mesh with repaired cracks.*

better results are achieved if the triangle center is first projected to the limit surface. But the triangle center cannot be reached by successive subdivision, as it will never appear as the vertex of a triangle for any finite subdivision depth.

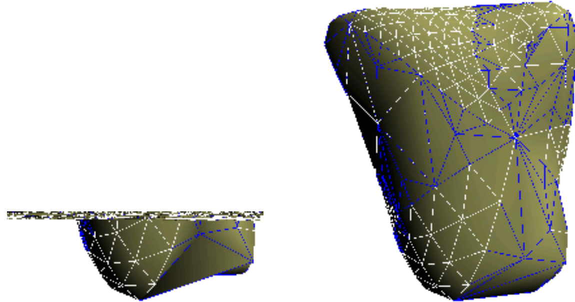


Figure 8: *Repairing nonconvex artifacts: Using the triangle center (left) and projecting the triangle center to the limit surface (right).*

Consequently, a different technique has to be used to obtain the center's limit position. As presented by Stam[8], the limit surface of a triangle for the Loop scheme can be expressed in closed parametric form. For a triangle without extraordinary vertices, i.e. where all vertices have valence six, twelve control points influence the shape of the limit surface:

$$s(\alpha,\beta) = C^T b(\alpha,\beta), \quad \alpha,\beta \geq 0, \quad \alpha+\beta \leq 1,$$

where C is a 12×3 matrix containing the vertices of the face and its 1-neighborhood, using the index scheme as in Fig. 9. The vector $b(\alpha,\beta)$ contains the twelve scalar-valued polynomial basis functions in barycentric coordinates $(\alpha,\beta,1-\alpha-\beta)$ providing the weights of the control points. The definition of $b(\alpha,\beta)$ is given in the appendix. It is sufficient to consider the regular case, because after one subdivision step, the vertices of the central triangle have valence six (cf. Fig. 9).

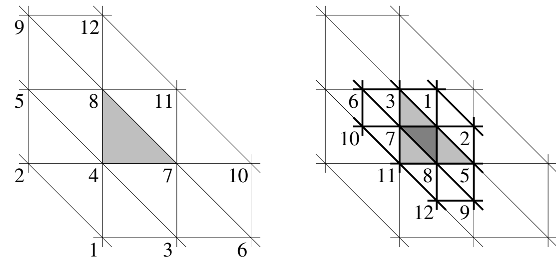


Figure 9: *A regular triangle and its 1-neighborhood. After one subdivision step, the inner triangle is regular*

This subdivision step doesn't cause much overhead, because most of the vertices computed in it are needed for repairing cracks anyway. Evaluating the basis functions $b_i(\alpha,\beta), i = 1,\ldots,12$ at the center coordinates $(\frac{1}{3},\frac{1}{3})$ yields the following weights:

$$\frac{1}{81}\left(\frac{1}{4},\frac{1}{4},\frac{7}{2},23,\frac{7}{2},\frac{1}{4},23,23,\frac{1}{4},\frac{1}{4},\frac{7}{2},\frac{1}{4}\right).$$

The limit surface normal at the center point is the cross product of the tangent vectors. These are obtained using the partial derivatives of the basis functions:

$$n(\alpha,\beta) = \left(C^T \frac{\partial}{\partial\alpha} b(\alpha,\beta)\right) \times \left(C^T \frac{\partial}{\partial\beta} b(\alpha,\beta)\right) \qquad (6)$$

Evaluating the partial derivatives at parameter values $(\frac{1}{3},\frac{1}{3})$ results in the following weight vectors for the limit surface tangents:

$$\frac{1}{27}\left(\frac{-2}{3},\frac{-5}{6},0,-13,-5,\frac{2}{3},13,0,\frac{-1}{6},\frac{5}{6},5,\frac{1}{6}\right)$$

$$\frac{1}{27}\left(\frac{-5}{6},\frac{-2}{3},-5,-13,0,\frac{-1}{6},0,13,\frac{2}{3},\frac{1}{6},5,\frac{5}{6}\right)$$

With this crack prevention scheme, positional discontinuities are resolved, and the use of the limit surface normal at the center vertex remedies color discontinuities. No T-vertices are introduced, and Gouraud shading produces good-looking results, as can be seen in the color plates.

6. Results

Timings have been taken by rotating a series of bumpy spheres with 200 to 4000 faces. The bumpy sphere is created by taking a bunch of random unit vectors, computing a triangulation of the unit sphere from them, and then randomly varying the the length of each vector, choosing a value from $[0.5, 1.5]$. With an increasing number of these random vectors, extremely curved models are created, because the distance variation of neighbouring mesh vertices is not correlated. This effect is demonstrated in Figure 10.

© The Eurographics Association and Blackwell Publishers 2000.

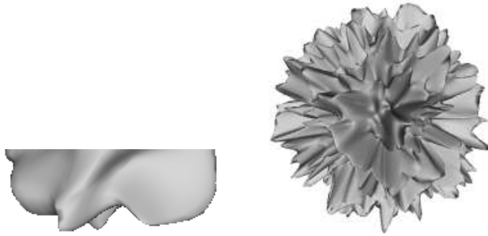


Figure 10: *Bumpy spheres with 200 and 4000 toplevel faces*

The tables in Fig. 11 show effective frame rates for a *single-threaded* test on two target machines. The first one is an SGI Onyx2 with a dual-pipe InfiniteReality II graphics board, employing a MIPS R10k running at 250 MHz. The second machine is a standard PC with a 500 MHz Pentium III (Katmai) and a GeForce 256 graphics board with 32 MB DDR-RAM. Main memory has not been an issue in this test. On both machines, the different tests show the same behaviour, although the PC is faster by a factor of 1.8 (!) compared to the Onyx2. All fancy OpenGL features (except Gouraud shading) have been disabled, and a single light source was active.

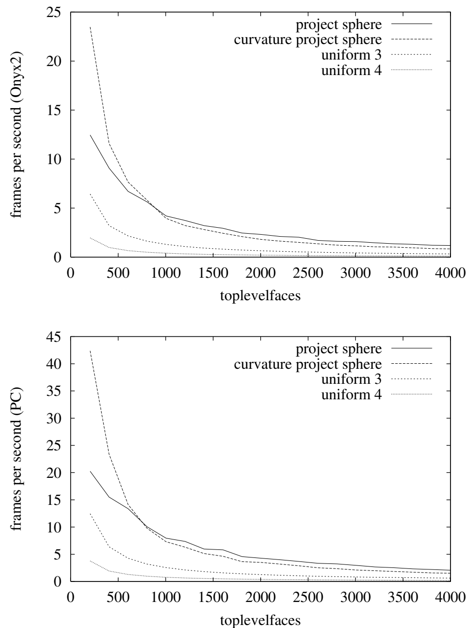


Figure 11: *Frame rates for Onyx2 and standard PC*

These tables show that uniform tesselation behaves much worse than adaptive tesselation-on-the-fly, although (i) no crack prevention has to be done, and (ii) the test scene is very clustered, so it is actually a bad example for adaptive tesselation. This result validates a central thesis of this article: Adaptive tesselation pays off.

The diagrams also show that, as curvature increases with the number of random vertices, the combined curvature-sphere strategy will tend to assign a higher subdivision depth to many faces. In fact, this strategy preserves object detail perceivably better than the project-sphere heuristic alone.

The diagrams in Fig. 12 show the numbers of OpenGL triangles actually drawn, as well as the number of toplevel triangles culled away by the backface culling heuristic. The percentage of backfaces decreases due to the bumpy structure of the model. Again, for the combined strategy, the increasing curvature takes over and prevents the project-sphere heuristic from introducing artifacts.

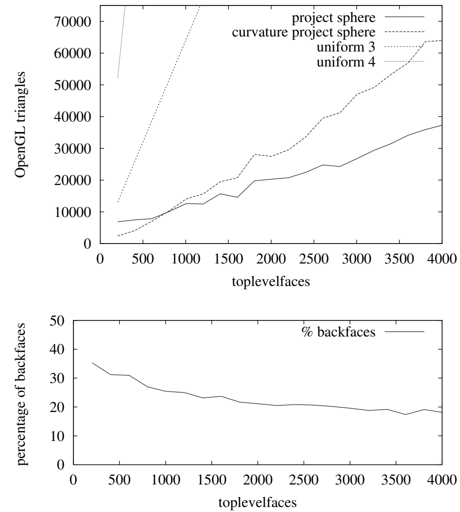


Figure 12: *Numbers of sub-triangles and backfaces*

7. Conclusions and Future Work

In this paper, it was shown that for interactive applications, a moderately optimized implementation of *adaptive tesselation-on-the-fly* for Loop subdivision surfaces can produce 10k triangles in each frame, from an 800 faces triangle mesh, with a frame rate of > 10 fps on a standard PC.

The tesselations generated by the proposed combination of heuristics accurately reproduce the limit surface geometry, up to a point where there is no perceivable difference between adaptive and high-depth uniform tesselations.

The tesselation module is designed in a geometry-engine fashion and can be plugged in as a rendering module for any pair of triangles, yet respecting the tesselation quality of neighbouring triangles.

Although these are encouraging results, much remains to

© The Eurographics Association and Blackwell Publishers 2000.

be done. First, we are confident to achieve a speedup by a factor of around 2 by optimizing the order of computations. The evaluation of heuristics, for instance, the projected size of a patch, can be directly integrated with the sliding window method. Second, the present design makes no use of *any* information that is stored for more than a single frame, and can thus not exploit temporal coherence. Although tesselation-on-the-fly aims at interactive applications where the control mesh is subject to change, due to continuous-level-of-detail techniques, even in this cases the control mesh will not change completely every frame.

The development of caching strategies for the results of preceding frames, as the tesselation, the pairing, and subdivision depth heuristics, will as well challenge our software design as it will also trigger the development of new, integrated algorithms for interactive freeform visualization.

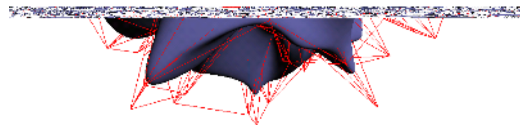
Appendix A: Parametric basis functions

The limit surface of a regular triangle face for Loop subdivision can be expressed in parametric form by $s(\alpha,\beta) = C^T b(\alpha,\beta)$, where C is the 12×3 control point matrix, using an index scheme as in Fig. 9. Barycentric coordinates sum to unity, consequently, in a coordinate tuple (α,β,γ) with $\alpha,\beta,\gamma \geq 0$, $\alpha+\beta+\gamma = 1$, γ can be substituted using $\gamma = 1-\alpha-\beta$. With this substitution, the vector of polynomial basis functions is[8]:

$$
\begin{aligned}
b(\alpha,\beta) \;=\; & \frac{1}{12}(\gamma^4+2\gamma^3\alpha^3, \quad \gamma^4+2\gamma^3\beta, \quad \gamma^4+2\gamma^3\beta+ \\
& 6\gamma^3\alpha+6\gamma^2\alpha\beta+12\gamma^2\alpha^2+6\gamma\alpha^2\beta+6\gamma\alpha^3+2\alpha^3\beta \\
& +\alpha^4, \quad 6\gamma^4+24\gamma^3\beta+24\gamma^2\beta^2+8\gamma\beta^3+\beta^4 \\
& +24\gamma^3\alpha+60\gamma^2\alpha\beta+36\gamma\alpha\beta^2+6\alpha\beta^3+24\gamma^2\alpha^2 \\
& +36\gamma\alpha^2\beta+12\alpha^2\beta^2+8\gamma\alpha^3+6\alpha^3\beta+\alpha^4, \\
& \gamma^4+6\gamma^3\beta+12\gamma^2\beta^2+6\gamma\beta^3+\beta^4+2\gamma^3\alpha \\
& +6\gamma^2\alpha\beta+6\gamma\alpha\beta^2+2\alpha\beta^3, \\
& 2\gamma\alpha^3+\alpha^4, \quad \gamma^4+6\gamma^3\beta+12\gamma^2\beta^2 \\
& +6\gamma\beta^3+\beta^4+8\gamma^3\alpha+36\gamma^2\alpha\beta+36\gamma\alpha\beta^2 \\
& +8\alpha\beta^3+24\gamma^2\alpha^2+60\gamma\alpha^2\beta+24\alpha^2\beta^2 \\
& +24\gamma\alpha^3+24\alpha^3\beta+6\alpha^4, \\
& \gamma^4+8\gamma^3\beta+24\gamma^2\beta^2+24\gamma\beta^3+6\beta^4 \\
& +6\gamma^3\alpha+36\gamma^2\alpha\beta+60\gamma\alpha\beta^2+24\alpha\beta^3 \\
& +12\gamma^2\alpha^2+36\gamma\alpha^2\beta+24\alpha^2\beta^2+6\gamma\alpha^3 \\
& +8\alpha^3\beta+\alpha^4, \quad 2\gamma\beta^3+\beta^4, \quad 2\alpha^3\beta+\alpha^4, \\
& 2\gamma\beta^3+\beta^4+6\gamma\alpha\beta^2+6\alpha\beta^3+6\gamma\alpha^2\beta \\
& +12\alpha^2\beta^2+2\gamma\alpha^3+6\alpha^3\beta+\alpha^4, \quad \beta^4+2\alpha\beta^3)
\end{aligned}
$$

References

1. Ed Catmull and J. H. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10:350–355, September 1978.
2. H. Hoppe. Progressive meshes. In *SIGGRAPH 96 Conference Proceedings*, pages 99–108. ACM SIGGRAPH, august 1996.
3. H. Hoppe. View-dependent refinement of progressive meshes. In *SIGGRAPH 97 Conference Proceedings*. ACM SIGGRAPH, 1997.
4. S. Kumar. *Interactive Rendering of Parametric Spline Surfaces*. PhD thesis, UNC at Chapel Hill, ftp://ftp.cs.unc.edu/pub/publications/techreports/96-039.ps.Z, 1996.
5. C. T. Loop. *Smooth Subdivision Surfaces Based on Triangles*. PhD thesis, Department of Mathematics, University of Utah, 1987.
6. D. Luebke and C. Erikson. View-dependent simplification of arbitrary polygonal environments. In *SIGGRAPH 97 Conference Proceedings*. ACM SIGGRAPH, 1997.
7. Kari Pulli and Mark Segal. Fast rendering of subdivision surfaces. Technical report, University of Washington, 1996.
8. Jos Stam. Evaluation of loop subdivision surfaces. In *SIGGRAPH 98 Conference Proceedings on CDROM*. ACM SIGGRAPH, july 1998.
9. Jos Stam. Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. In *SIGGRAPH 98 Conference Proceedings*, pages 395–404. ACM SIGGRAPH, july 1998.
10. D. Zorin and P. Schröder. Subdivision for modeling and animation. In *Siggraph 99 Course Notes*. ACM Siggraph, 1999.



© The Eurographics Association and Blackwell Publishers 2000.

Figure 13: *Demonstration of crack prevention. Subdivision depth has been randomly assigned. The right picture shows the effect of the backface culling heuristic. The object is rotated to show missing triangle pairs at the back.*

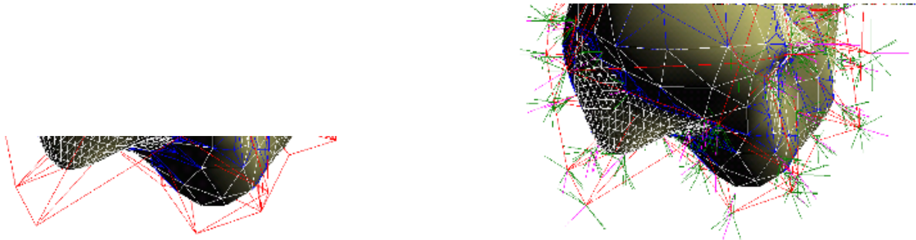


Figure 14: *Curvature dependent assignment of subdivision depth values. At the right, additionally all normal vectors necessary for the computation of the curvature-dependent subdivision depths are shown.*

© The Eurographics Association and Blackwell Publishers 2000.