OpenSG Symposium (2003)
D. Reiners (Editor)

Adaptive Tesselation of Subdivision Surfaces in OpenSG

Volker Settgast, Kerstin Müller, Christoph Fünfzig and Dieter Fellner

Institute of ComputerGraphics,
Technical University of Braunschweig, Germany

Abstract

The need for realistic models especially in virtual reality environments leads us to freeform shapes like subdivision surfaces. Based on a standard polygonal mesh, the modeller can build various kinds of shapes using an arbitrary topology and special geometrical features like creases. Nowadays, it also is a part of most standard modelling packages like Maya and 3DStudio Max. However, the interactive display of subdivision surfaces in current scene-graph systems as static level-of-details is unpractical, because of the exponentially increasing number of polygons during the subdivision steps. Therefore an adaptive algorithm, choosing only the necessary quads and triangles, is required to obtain good looking images at high framerates.
In this paper we present a rendering algorithm considering the static surface properties like curvature as well as viewdependent properties like silhouette location and projection size. Without modifying the basis mesh, the method works patchwise and tesselates each patch recursively using a new datastructure, called Slate. *We show that the algorithm can be implemented efficiently using direct OpenGL. In comparison we consider the algorithm's integration into the scenegraph system OpenSG, where the scenegraph node feeds into an indexed-face-set node.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling: Curve, surface, solid, and object representations I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism: Virtual reality

1. Introduction

Subdivision surfaces and their interactive rendering have become an important issue in computer graphics. The need for realistic models especially in virtual reality environments leads us to freeform shapes to which subdivision surfaces converge. With this surface type the modeller has a lot of possibilities at hand, e.g. an arbitrary topology and special geometrical features like creases. It is now part of most standard modelling packages (for example Maya and 3DStudio Max).

However, the interactive display of subdivision surfaces poses hard problems because of the exponentially increasing number of polygons during the subdivision steps. Thus an adaptive algorithm, choosing only the necessary quads and triangles, is required to obtain good looking images at high framerates. There are some proposals for such algorithms in the literature, which can basically be grouped into two classes.

The first class evaluates basis functions for each valence (and configuration of special features). For the tesselation only linear combinations of basis function values with control points have to be computed. In [7] the basis functions are fetched from precomputed tables.

The second class employs the recursive nature of the subdivision schemes in the algorithm. Pulli and Segal[5] evaluate Loop subdivision by pairing triangles and tesselating them using the sliding window method. This work was refined by Müller and Havemann[4] to prevent cracks between adjacent patches with different tesselation depths. For Catmull-Clark subdivision surfaces a recursive approach was given in [8]. Bischoff et al[9] proposed a hardware solution for Loop surface tesselation.

This work aims at an adaptive tesselation algorithm which recursively subdivides a patch using two static sized *Slates*. By its nature the algorithm is easy to implement, local in its memory references and does not require any precomputation. Subdivision rules for special features can be integrated without changing the algorithm. During execution the basis mesh is left unmodified, which is an essential feature for its use in a scenegraph system. With OpenSG there are mainly two possibilities how to integrate such an adaptive tesselation algorithm. The simple one deposits OpenGL-commands directly from a single scenegraph node, which contains the basis mesh and all quality parameters. A second, layered possibility is to feed one or several indexed-face-set nodes out of the *SubdivisionSurface*-node. This allows to control and change the primitive output in the indexed-face-set node implementation independently. We show the impacts of the two possibilities on the basis of some example models.

2. Preliminary Remarks

The following presentation is based on the Catmull-Clark subdivision scheme. Nevertheless, the technique can easily be transfered to other subdivision schemes based on triangle or quad meshes.

2.1. Notations

We use capital letters to denote faces and patches. Normalisation of a vector is written as $\|v\|$. We assume that normal vectors are always normalised. The superscript describes the subdivision depth, e.g., cp_n^l, whereas the subscript indicates the numeration. For clarity of presentation, we use, e.g., M instead of M^l, if the index l is not essential for the explanation. Also for this reason the figures are schematic.

2.2. Subdivision Surfaces

A subdivision surface is defined by a *control mesh* M^0. By applying the subdivision rules to the mesh M^0, we get a finer mesh M^1. The sequence of control meshes converges against the *limit surface* $L(M^0)$.

The vertices of a control mesh M^l are called *control points* cp_i^l. The *1-neighbourhood* of cp_i^l includes each control point lying on an edge incident on cp_i^l. The *1-neighbourhood of a face* $F = \square cp_1^l cp_2^l cp_3^l cp_4^l$ includes all faces adjacent to any vertex of F and is denoted by $N^l(F)$. The *edge neighbours* of F are the faces having a shared edge with F.

The Catmull-Clark scheme provides three kinds of subdivision rules (see Fig. 1). One rule creates a vertex cp_i^{l+1} in the refined mesh M^{l+1} for each vertex cp_i^l of M^l as an affine combination of the 1-neighbourhood of cp_i^l. A further rule creates a vertex cp_j^{l+1} in M^{l+1} for each edge of M^l as an affine combination of the vertices of the two faces adjacent to that edge. And the last rule generates a new vertex for each face $F = \square cp_1^l cp_2^l cp_3^l cp_4^l$ from the average of all of the points defining the face. The weights for the rules can be found, e.g., in [1],[6].

By applying alternative subdivision rules special features like creases, corners or semi-sharp edges[10] can be created on the surface.

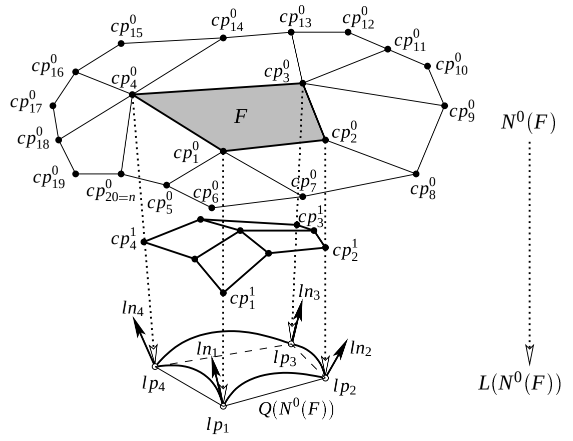

Figure 1: *Example Catmull-Clark subdivision*

A *limit point* is defined as $lp_i := \lim_{l\to\infty} cp_i^l$. The *limit normal* ln_i denotes the surface normal of $L(M)$ at the point lp_i. Calculating the limit point of a control point cp_i can be performed via an affine combination of the 1-neighbourhood of cp_i. The computation is similar to the vertex subdivision rule with different weights. The limit normal of a control point cp_i can be calculated by taking the cross product of two tangent vectors built from a weighted sum of the 1-neighbourhood of cp_i. The weights for the limit points and limit normals can be found in [14], [15].

The 1-neighbourhood of a face, $N(F)$, which is a part of M, contains the control points $cp_1, \ldots, cp_n$. The limit surface $L(N(F)) = L(cp_1, \ldots, cp_n)$ is called the *patch corresponding to F*. This patch is a part of the limit surface $L(M)$. $Q(cp_1, \ldots, cp_n) = Q(N(F)) = \square lp_1 lp_2 lp_3 lp_4$ is the *chord quadrangle* of F. The normal of a face $F = \square cp_1 cp_2 cp_3 cp_4$ is in general not unique, where the $cp_i, i = 1,..,4$ are ordered counter clockwise. Therefore, we define the normal n of F referable to one of its vertex points, e.g., cp_0 as $n = \|(r_1 - r_3) \times r_2\|$ with $r_1 = cp_2 - cp_1$, $r_2 = cp_3 - cp_1$, $r_3 = cp_4 - cp_1$.

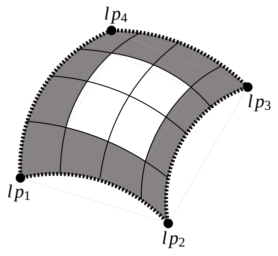

Figure 2: *Inner (white) and outer (grey) subpatches of a patch S after two subdivision steps. The four outer border curves of S are drawn dashed.*

Let $L(cp_1, \ldots, cp_n)$ be the *starting* patch S to be tesselated. After l subdivision steps of S, we obtain 4^l subpatches S_j. The border curves of S_j which are part of the border curves of the *starting* patch are called *outer* border curves. Subpatches S_j without outer border curves are called *inner patches* (see Fig. 2). Similarly subpatches with outer border curves are *outer patches*. In subdivision depth l we get $4 \cdot 2^l$ outer border curves, $4 \cdot 2^l - 4$ outer patches and $4^l - 4 \cdot 2^l + 4$ inner patches.

Consider two neighbouring faces A and B of mesh M^l and the subfaces B_1, B_2, B_3, B_4 of B where B_1, B_2, B_3, B_4 are parts of the mesh M^{l+1} (see Fig. 3). $Q(N(B_1))$ and $Q(N(B_2))$ do in general not lie at the border of $Q(N(A))$. Thus a *gap* may occur between the differently tesselated limit surfaces of A and B. To obtain a gap-less surface, in spite of different subdivision depths of A and B, the tesselation of A – with the lower depth – must be fit to the higher one. We consider this topic more precisely in section 3.4.

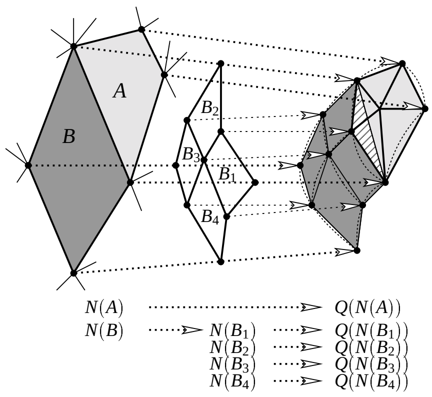

Figure 3: *Two neighbouring tesselated patches at different subdivision depths: a gap (hatched triangle) occurrs between the two surfaces. So it is necessary to modify the tesselation of A to obtain a closed surface. The dashed curves on the right side describe the patches A and B_1, B_2, B_3, B_4.*

3. Tesselation on the fly with two Slates

3.1. Basic Idea

In order to utilise existing computing power and memory optimally, each face of the basis mesh receives a reasonable subdivision depth (Section 3.2): The parts of the mesh with a big curvature and at the silhouette obtain a high subdivision depth as well as faces with a small distance to the camera. Back faces are not subdivided at all. After this subdivision depth assignment, each face of the mesh together with its 1-neighbourhood is fetched from the mesh and given to a dedicated tesselation thread. The subdivision is done down to the assigned depth and the proper limit points and normals are computed (Section 3.3). If the edge neighbours of the considered basis face have a higher subdivision depth, the tesselation at the border is adapted to the neighbour to avoid gaps (Section 3.4). The drawing of the resulting shape chunks is described in detail in Sections 4 and 5.

Because we use a static sized data structure, we must choose a maximum subdivision depth *maxsd* that can be applied by our tesselation algorithm. Setting $maxsd = 5$ is sufficient for common models. *maxval* is the maximum valence of a vertex that can occur in a mesh. In practical cases the valences are smaller than 50.

3.2. Finding the Subdivision Depth

For high quality images at an acceptable frame rate the following items are combined to assign a subdivision depth to each patch:

- curvature
- visibility
- membership to the silhouette
- projected size of the patch

The curvature is considered only once for a new or changed basis mesh. All other items are evaluated per frame. Most items are not evaluated exactly. Instead we use fast approximations to find a suitable subdivision depth for each patch.

As an approximation for the curvature we use the normal cone technique[2] for each vertex. Consider vertex cp_0^0 with valence val, its limit normal ln and the normals n_i^l of the neighbour faces $i = 1, .., val$ at subdivision depth l. We place a normal cone around ln with a given aperture angle α. Then we search for the minimal l, so that

$$\forall i = 1, .., val: \quad ln \cdot n_i^l < \cos\alpha$$

For this subdivision depth l all normal vectors of the neighbour faces are inside the normal cone (see Fig 4). So l is a subdivision depth guaranteeing a maximum curvature in the considered vertex by an user defined $NormalConeAperture = \alpha$.

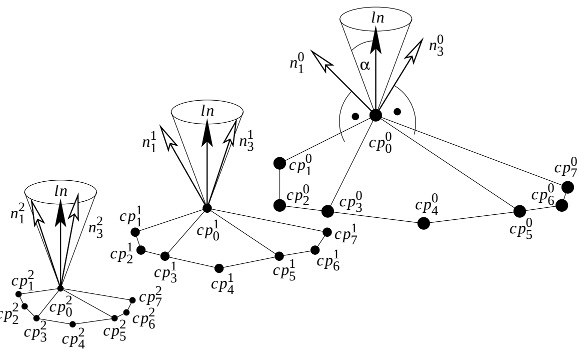

Figure 4: *The normal cone with angle α at subdivision depth $0, 1$ and 2: In subdivision depth 2 the depicted normals of the neighbour faces are inside the normal cone.*

The following steps are done per frame. First, we classify[11] each vertex cp of the basis mesh with limit point lp and limit normal ln into

Back vertex	:	$\varepsilon < ln \cdot camray$
Front vertex	:	$-\varepsilon > ln \cdot camray$
Silhouette vertex	:	$-\varepsilon \leq ln \cdot camray \leq \varepsilon$

with $camray = \|lp - cameraposition\|$ and a user defined $VertexClassifier = \varepsilon$. Afterwards, we traverse each face F of the basis mesh and decide if F belongs to:

back	:	all its vertices are back vertices,
front	:	all its vertices are front vertices,
silhouette	:	otherwise

Back faces will not be subdivided at all. Front faces obtain as subdivision depth the maximum of their vertex depths, assigned by curvature. Silhouette faces get the average of *maxsd* and the maximum of their vertex depths, assigned by curvature.

Lastly, we estimate the projected size of each front face and each silhouette face and adapt their subdivision depths respectively. This ensures that no unnecessary refinement will be done and prevents the presentation from being too coarse, e.g., visible in the highlight. We use a bounding sphere around the considered patch and estimate the radius of the projected sphere onto the image plane. For user defined $ProjectedSizeMin$ and $ProjectedSizeMax$, the maximum and minimum radius R_{min} and R_{max} of the desired subpatches are computed dependent on the distance to the camera, the field-of-view of the camera and the image resolution. If the radius R according to subdivision depth l, that means $R \cdot 1/2^l$, is greater than R_{max}, s further refinements are neccessary until $R \cdot 1/2^{l+s} \leq R_{max}$. We obtain $l+s$ as the new subdivision depth. If in contrast $R \cdot 1/2^l < R_{min}$, then a smaller subdivision depth is sufficient. We search the minimum s so that $R \cdot 1/2^{l-s} \geq R_{min}$ and obtain $l-s$ as the new subdivision depth. In the case $R_{max} \geq R \cdot 1/2^l \geq R_{min}$ no change of the depth is necessary.

3.3. Subdivision Work with Slates

For each face of the basis mesh, we now tesselate the corresponding patch with the chosen subdivision depth. Thereby the basis mesh is not modified, instead we use a separate data structure consisting of two *Slates*. A Slate is composed of a two-dimensional array *Table* of size $(2^{maxsd}+3)^2$ and four one-dimensional arrays $corner_{1,2,3,4}$ of size $(maxval-4) \cdot 2$. The Slates are allocated statically and can be reused for each face that has to be tesselated.

Firstly, the 1-neighbourhood of the considered face F is collected into the Slate (see Fig. 5). The vertices of F and the vertices of its edge neighbour faces are stored in the table. If one of the vertices of F has $valence > 4$, then the remaining vertices of $N(F)$ are stored in the dedicated corner arrays.

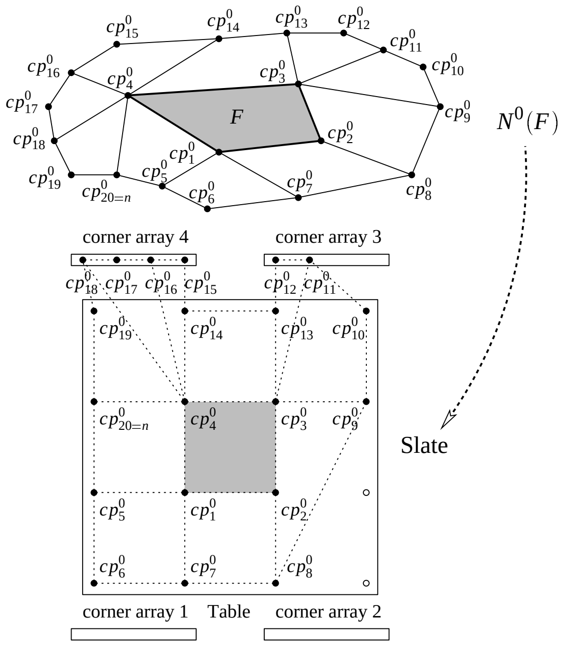

Figure 5: *Collecting the 1-neighbourhood of F from the basis mesh into a Slate.*

The algorithm *tesselate* starts with the chosen subdivision depth and Slate 1 as arguments (see Fig. 6).

```
tesselate (subdivision depth, Slate i)
  if subdivision depth $> 0$ then
    perform one subdivision step and
    save the new points in Slate $j = (i+1)$ mod 2;
    tesselate (subdivision depth - 1 , Slate j);
  else
    compute limit points and normals
    from Slate i;
  end if;
```

3.4. Gap prevention

Due to the fact that we use an adaptive presentation of the surface, neighbour patches can have different subdivision depths. An example is shown in figure 7, six patches with different depths are illustrated schematically. Gaps can occur between the differently tesselated patches depicted as dashed lines. To avoid these gaps, the tesselation of the patches with lower subdivision depth must be modified. For this purpose, we replace the outer chord quadrangles lying at the border to a further subdivided patch (grey marked in Fig. 7) with suitable triangle fans (see Fig. 8).

Such a triangle fan is built from the limit point of the new face point of the next subdivision step (marked by a small circle in Fig. 8), the vertices of the quads and the points ly-

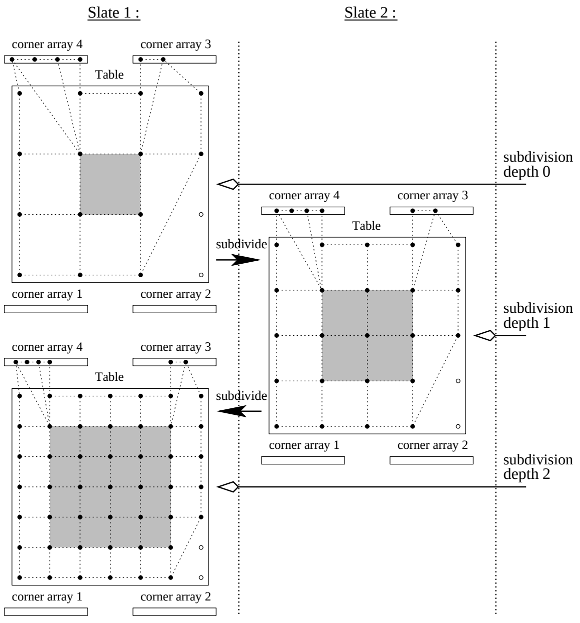

Figure 6: *Subdivision process with Slates.*

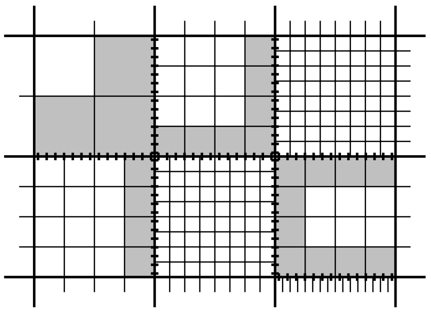

Figure 7: *Schematical presentation of 6 patches at different subdivision depths: Gaps can occur at the dashed lines, so the grey marked quadrangles must be replaced by a suitable tesselation.*

ing at the border to the deeper tesselated neighbour. If neighbour quadrangles of the same patch are replaced like this, the shared edge is subdivided and the resulting limit points (marked by a small box in Fig. 8) are added to the triangle fan. The additional subdivision step is essential to prevent undesired sharp bends on the tesselated surface. The suggested gap prevention method guarantees an adequate geometry for good looking images.

4. Rendering directly to OpenGL

We use the OpenGL API to compare to the performance of OpenSG. At a given depth the limit points and normals are

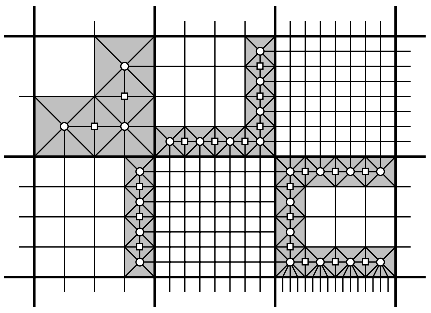

Figure 8: *Schematical presentation of 6 patches at different subdivision depths after the gap prevention: No gaps occur.*

calculated into a single vertex array. For good performance quad strips are used where possible. For each two lines of the tesselation a quad strip is sent to OpenGL. The first and the last two lines and the first and the last quad of every other line have to be handled separately when a finer tesselation is present in the neighbour patch. In that case a triangle fan is used as described in Section 3.4. The *glDrawElements*-function allows to use indices into the limit point and normal array for defining the primitives.

To increase the performance, the limit points and limit normals are cached. They can be reused as long as the subdivision depth is not increased. With these cached limit points and normals at subdivision depth t we have all limit points and normals from subdivision depth 0 to t without further effort.

5. Rendering with OpenSG

5.1. Structure

A scenegraph node in OpenSG consists of the tree structure in the *Node* class and the functionality in the *NodeCore* fieldcontainer. In order to execute the adaptive tesselation, we need a derived NodeCore class: *DynamicCCSubdivision*. The fieldcontainer DynamicCCSubdivision consists of the following fields

Mesh
Singlefield of pointer type to an OpenMesh[13] object. The user sets this field to the basis mesh to be rendered.

Tesselator
Singlefield of pointer type to a tesselator object. This tesselator object is created internally in the *changed*-method in response to the user's assignment of a new basis mesh.

User Parameters
NormalConeAperture, ProjectedSizeMin/Max, VertexClassifier

A natural scenegraph structure for the implementation of the adaptive, patchwise algorithm with gap prevention would be to create two *Geometry*-nodes per patch as childs of the

DynamicCCSubdivision-node. One node would be assigned to the inner part of a patch and the other to the border part to differently tesselated neighbours (see Fig. 8). In this structure *DynamicCCSubdivision* would be derived from the *MaterialGroup*-fieldcontainer, which would specify an overall material for the subdivision surface object. Unfortunately this structure consumes a large amount of memory and adds a considerable runtime overhead, especially for practical models with more than 2000 patches.

Because of this we chose a leaner structure, where *DynamicCCSubdivision* is derived from the *Geometry*-fieldcontainer and does not require any further nodes. The fieldcontainer for storing indexed-face-sets in OpenSG[12] is called *Geometry*. Besides *Material* it has several fields referencing geometry attribute multifields like

Positions
Normals
Colors
TexCoords

The remaining fields are used for the definition of OpenGL primitives

Types
OpenGL primitive types
Lengths
Length of the Indices sections, defining one primitive.
Indices
Index sequences into the attribute multifields.

With **Indices** and an additional multifield **IndexMapping** the *Geometry*-fieldcontainer can be switched between three different modes *Non-Indexed*, *Single-Indexed* and *Multi-Indexed*. For subdivision surfaces with a limit normal per limit vertex and a small number of sharp crease edges we use the *Geometry*-nodes in the *Single-Indexed* mode. In this mode the output implementation in OpenSG employs OpenGL vertex arrays.

5.2. Updating the *Geometry*-Node

DynamicCCSubdivision does a preprocess every time the basis mesh is altered. The OpenSG fields Positions, Normals, Types, Lengths and Indices are assigned in the *Geometry*-object. The Positions and Normals multifields, to which the limitpoints and normals are written, are resized to a static size sufficient for the maximum subdivision depth *maxsd*. In this preprocess the curvature information (Section 3.2) is collected by the tesselator object. The Indices, Types and Lengths fields have to be updated every frame. The inner part of the surface is built out of quad strips only. For the outer part a mixture of quad strips and triangle fans is used depending on the gap prevention process.

In summary, the Indices multifield is the concatenation of the indexarrays used in Section 4 and the resulting OpenGL command sequence is nearly the same. Similarly we use caching of limit points and limit normals like in Section 4.

6. Analysis of Memory Consumption and Runtime

Let us investigate the memory consumption during the tesselation process. As mentioned before, we use two statically allocated structures, named Slate 1 and Slate 2, to perform all subdivision. We choose the size of the used Slates in dependence of *maxsd* and *maxval*:

$$(2^{maxsd}+3)^2+4\cdot ca$$

where $(2^{maxsd}+3)^2$ is the size of the 2D array named by Table, and $ca=(maxval-4)\cdot 2$ is the size of the corner array, of which four are used in one Slate. Two of this Slates are sufficient to calculate all subdivisions, so the total number of 3D vectors necessary to compute all subdivisions are:

$$2\cdot(2^{maxsd}+3)^2+16\cdot maxval-64$$

To obtain a tesselation for a given patch at subdivision depth l $n=(2^l+1)^2$ limit points and the same number of normals have to be computed. In the following, we consider the time to calculate this points and normals. Assuming the time required to compute a vertex as constant (exactly it depends on the valence of the 4 possible extraordinary points), the number of calculations CN is:

$$CN=SN+LN$$

with SN being the number of calculations during the subdivision steps and LN is the number of computations to obtain the limit points and normals, C is the number of vertices in the four corner arrays of the Slate (maximal size of C is $4\cdot(maxval-4)\cdot 2$):

$$\begin{aligned} SN &= \sum_{i=1}^{l}(2^i+3)^2+C \\ LN &= 2\cdot(2^l+1)^2 \end{aligned}$$

So we obtain:

$$CN=\frac{10}{3}\cdot 2^{2l}+16\cdot 2^l+l\cdot(9+C)-\frac{34}{3}$$

This is linear in the number of limit points:

$$\begin{aligned} & O(\frac{10}{3}\cdot(\sqrt{n}-1)^2 \\ & \quad +16\cdot(\sqrt{n}-1)+(9+C)\cdot\ln(\sqrt{n}-1)-\frac{34}{3}) \\ = {} & O(n) \end{aligned}$$

7. Results

Figure 14 and 15 demonstrate the correctness of the adaptive refinement. To show the effect of the backface culling heuristic, we use a ‘second’ camera and have a look at the back parts (see Figure 15). The correct assignment of subdivision depths is visible in the wireframe in Figure 14: Patches that are members of the silhouette have the highest subdivision depth, highly curved parts also have a finer tesselation. Gap

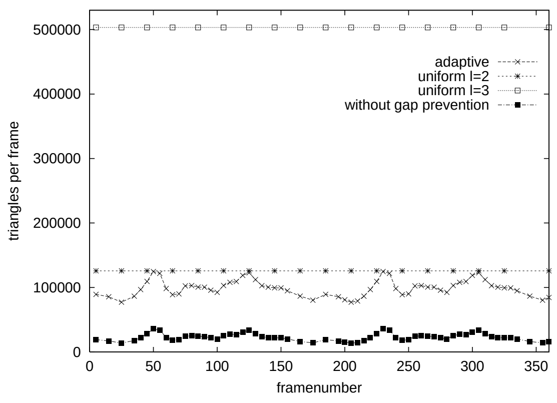

Figure 9: *Resulting triangle counts for adaptive and uniform tesselation of testscene II.*

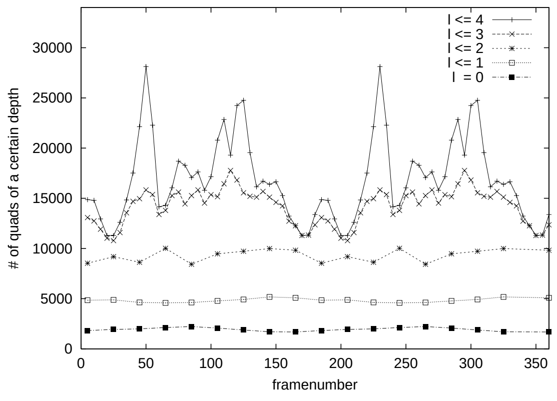

Figure 10: *Distribution of the resulting quadrangles (without gap prevention) to certain subdivision depths for testscene II.*

prevention is also done between the differently tesselated patches.

For the next tests with the results illustrated in the figures 9 – 12 we choose a scene of a spacestation (see Figures 16 and 17) with a variable number of spacecrafts. The camera eye performs a double rotation around the y–axis, facing the model center. Benchmarks were performed on a standard PC with Intel P4 1.7 GHz, 512 MB main memory and a GeForce 4 TI 4600 graphics board.

Figure 9 shows the resulting triangle counts for the adaptive and uniform tesselation. The adaptive algorithm selects only those triangles which are necessary for a high quality image. In contrast, the uniform tesselation takes all triangles at a given depth, unfortunately also the non-visibles. The curve of the adaptive tesselation varies, because the number of drawn triangles changes during the framesequence.

Considering the framerate, the direct rendering with OpenGL is faster than the OpenSG version because of the missing administrative overhead. But both solutions are faster than a comparable uniform tesselation. During the first round of the rotation the framerates increase because the limit point and normal caches get successively filled.

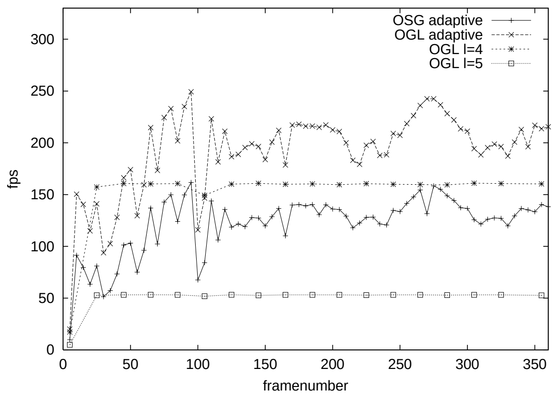

Figure 11: *Framerates for testscene I. The adaptive algorithm uses depths 1 – 5*

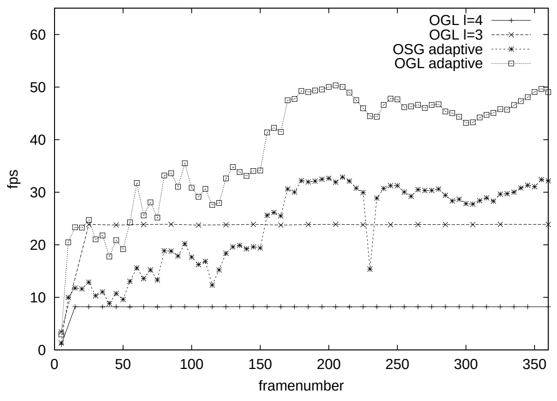

Figure 12: *Framerates for testscene II. The adaptive algorithm uses depths 0 – 4*

8. Further work

The tesselation algorithm presented here can easily be adapted to other kinds of subdivision surfaces based on quad meshes. For subdivision schemes using a triangle mesh (e.g. Loop[3]), an additional step is necessary to work optimally with the Slate structure. If the mesh is new or has changed, it is partitioned into triangle pairs by, e.g., a greedy algorithm[4],[5]. In practice, very few singletons remain that obtain a dummy partner. The resulting triangle pairs are treated like the quadrangles described before: Each triangle pair gets a joint subdivision depth, is collected into a Slate with their 1-neighbourhood and tesselated in the same way.

To achieve a nearly constant framerate, an estimation of runtime must be done. If the estimated time for the next frame composed of the calculation time for new points and the time to draw the cached and new computed quads and triangles is too high, the quality of the image must be adapted to the available time to keep the framerate.

9. Conclusion

We have shown that an adaptive tesselation algorithm using two static sized *Slates* performs very good in practice. A classification scheme detects visually important parts of the surface for additional refinements to obtain high quality images. As a data structure for the basis mesh, we use the OpenMesh developed in the OpenSG PLUS project. But the algorithm can also be adapted to another mesh with the functionality to collect the 1-neighbourhood and to store special feature information for vertices and edges like sharpness, weights etc. We have compared two kinds of integration into OpenSG. The simple one deposits OpenGL-commands directly from a single scenegraph node. The second one uses a layered approach, where the *DynamicCCSubdivision*-node feeds one (or several) Geometry-nodes. Despite it is somewhat slower, the output implementation can be changed independently and the updates per frame can be done in a dedicated thread assisted by OpenSG.

In conclusion the algorithm is simple and efficient, easy to implement and also easy to extend to other subdivision rules.

Acknowledgements

This work has been supported by the OpenSG PLUS project funded by the German Federal Ministry of Education and Science (bmb+f) under grant 01 IRA 02G and by the ModNav3D project funded by the German Research Council (DFG) under grant Fe-431/4-3.

References

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

2. Leon A. Shirman and Salim S. Abi-Ezzi. The Cone of Normals Technique for Fast Processing of Curved Patches. *Proc. of Eurographics*, **12**(3):261–272, 1993. 3

3. C. Loop. Smooth subdivision surfaces based on triangles. *Master thesis, University of Utah*, 1987. 7

4. K. Müller and S. Havemann. Subdivision Surface Tesselation on the Fly using a versatile Mesh Data Structure. *Computer Graphics Forum (Eurographics 2000 Proc.)*, **19**(3):151–159, 2000. 1, 7

5. K. Pulli and M. Segal. Fast Rendering of Subdivision Surfaces. *Technical report, University of Washington*, 1996. 1, 7

6. D. Zorin and P. Schröder. Subdivision for modeling and animation. *SIGGRAPH 99 Course Notes*, 1999. 2

7. J. Bolz and P. Schröder. Rapid Evaluation of Catmull-Clark Subdivision Surfaces. *Web3D'02 Conference Proceedings*, pp. 11–17, 2002. 1

8. S. Havemann. Interactive rendering of Catmull/Clark surfaces with crease edges. *The Visual Computer*, **18**:286–298, 2002. 1

9. S. Bischoff, L. Kobbelt and H. P. Seidel. Towards Hardware Implementation of Loop Subdivision. *Eurographics Workshop on Graphics Hardware*, pp. 41–50, 2000. 1

10. T. DeRose, M. Kass and T. Truong. Subdivision Surfaces in Character Animation. *SIGGRAPH 98 Conference Proceedings, Annual Conference Series*, pp. 85–94, 1998 2

11. D. Luebke and C. Erikson. View-dependent simplification of arbitrary polygonal environments. *SIGGRAPH 97 Conference Proceedings, Annual Conference Series*, pp. 199–208, 1997 4

12. Dirk Reiners, Gerrit Voß and Johannes Behr. OpenSG - Basic Concepts. *Proc. OpenSG Symposium 2002* 6

13. Mario Botsch, Stephan Steinberg, Stephan Bischoff and Leif Kobbelt. OpenMesh - a generic and efficient polygon mesh data structure. *Proc. OpenSG Symposium 2002* 5

14. H. Biermann, A. Levin and D. Zorin. Piecewise Smooth Subdivision Surfaces with Normal Control. *SIGGRAPH 2000 Conference Proceedings, Annual Conference Series*, pp. 113–120, 2000 2

15. H. Halstead, M. Kass and T. DeRose. Efficient, Fair Interpolation using Catmull–Clark Surfaces. *COMPUTER GRAPHICS Proceedings, Annual Conference Series*, pp. 35–44, 1993 2

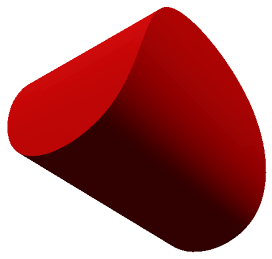

Figure 13: *Subdivision Surface with sharp edges.*

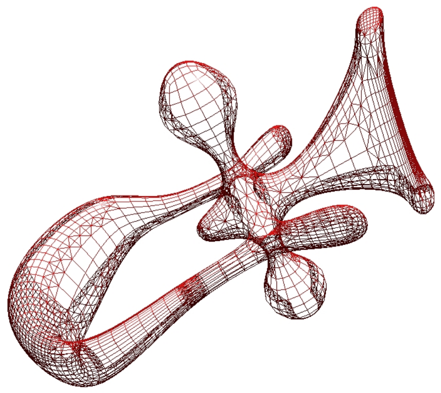

Figure 14: *Wireframe view to show the adaptive tesselation.*

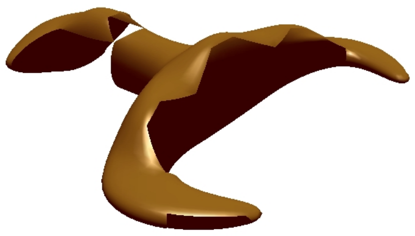

Figure 15: *Effect of the backface culling heuristic: The camera is in front of the spaceship.*

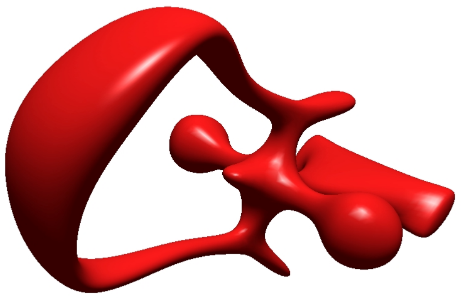

Figure 16: *Testscene I for framerate (104 basis faces with depths 1 – 5).*

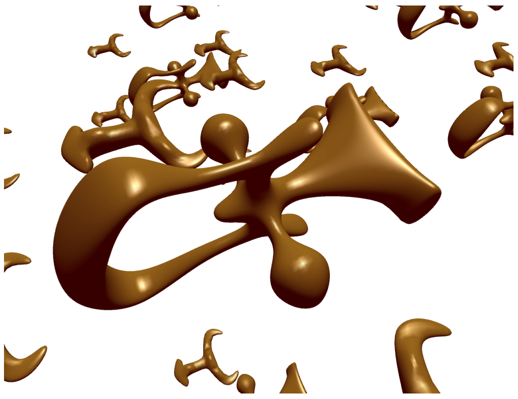

Figure 17: *Testscene II for framerate (3932 basis faces with depths 0 – 4).*

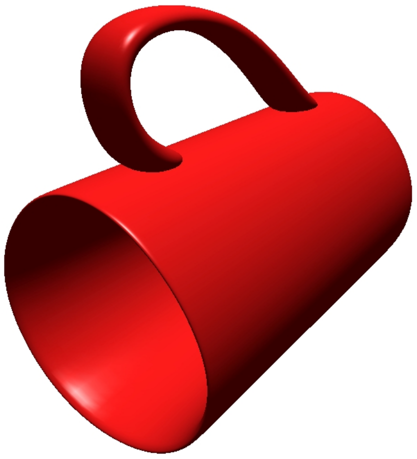

Figure 18: *Cup model with 70 basis faces, subdivided using depths 1 – 5.*

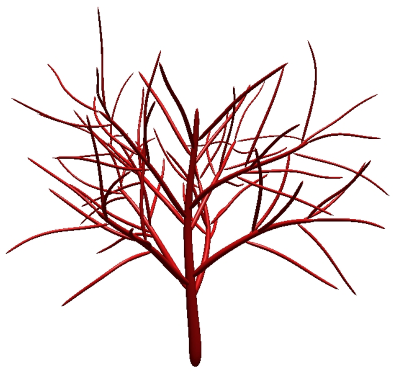

Figure 19: *Tree model with 1434 basis faces, subdivided using depths 1 – 3 (Model by the CHARISMATIC team).*