

Flexible Adaptive Tessellation of Subdivision Surfaces

Tatiana Surazhsky*
Samsung Telecom Research Israel

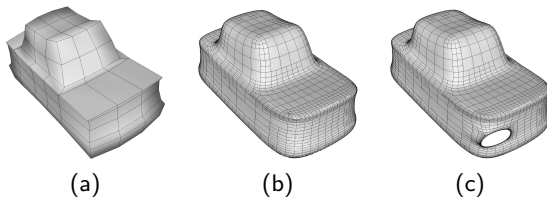


Figure 1: (a) shows the level 0 control mesh; (b) shows the *quads* of the adaptively subdivided surface; (c) is the same model with face deleted.

Abstract

Subdivision surfaces are a popular representation for the freeform shape in geometric modeling. While the subdivision process rapidly converges to a smooth shape, computing and rendering all the vertices is too expensive in the sense of the memory consumption and run time, and also superfluous. Our work was inspired by a demand for a rendering of dynamic scenes and an interactive rendering of the animated character model with reduced memory consumption and *no dynamic allocations*. The subdivision surface is rendered with no modifications of the control mesh, while allowing flexible tessellation on all subdivision levels as well as feature support (sharp edges and boundaries and vertex editing) for each level.

CR Descriptors: I.3.3 [Computer Graphics]: Picture/image generation - *Display algorithms*; I.3.6 [Computer Graphics]: Methodology and Techniques.

Keywords: Subdivision surfaces, rendering, adaptive tessellation

1 Introduction

In this work we introduce a novel approach to adaptive tessellation of subdivision surfaces. A subdivision process repeatedly refines a mesh of vertices, edges and faces and finally converges to a smooth *limit surface*.

Subdivision surfaces became a popular representation type for surfaces. For example, Catmull-Clark scheme [4] is successfully used in animation [6] and game engines [16]. Even “Dassault Systemes” used subdivision surfaces as a base for their “Catia – Imagine & Shape” product in order to enable designers and engineers to quickly and intuitively transform

*Samsung Electronics, Samsung Telecom Research Israel, Beit Lumir, 22 Maskit st., Herzliya Pituah, 46733 Israel; e-mail: tatiana.surazhsky@samsung.com

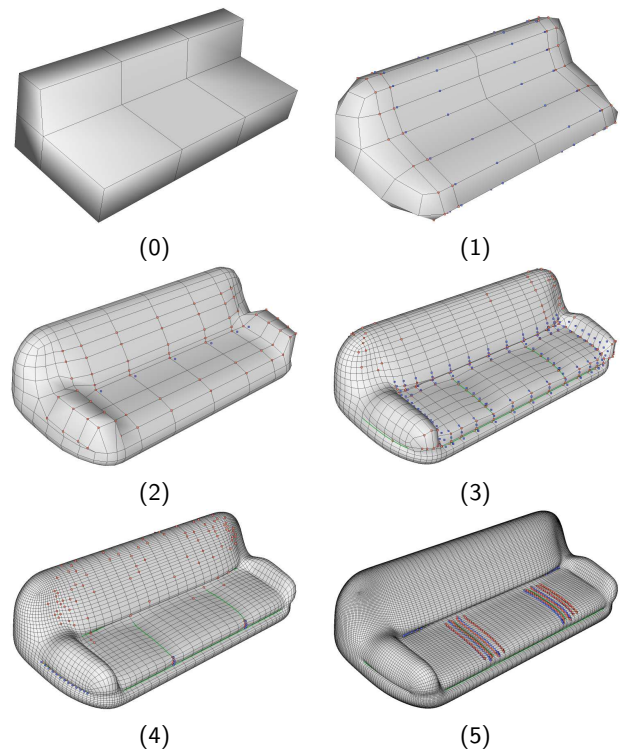


Figure 2: (0) is the base mesh (level 0), (k) is a level $k = 1, 2, 3, 4, 5$ mesh with the vertices edited by the designer. Blue points are the *original* locations computed by the scheme, the red points are the *modified* locations. The green lines mark the sharp edges. The final shape is shown on Figure 3.

a shape idea into a 3D geometric model [5]. Among the advantages of the subdivision representation over the spline surfaces are their simplicity and ability to represent shapes of different topologies.

The refinement process converges rapidly to a visually pleasing smooth surface, however rendering all the polygons of the fine mesh is costly in terms of memory consumption and run time. Moreover, rendering all these polygons is superfluous, for example if they are not visible or have low curvature. *Adaptive* subdivision solves these problems by subdividing only the polygons in the problematic areas, such as the regions with high curvature or that are close to silhouette. The concept of adaptive subdivision is closely related with the T-splines of [12], where the control points are inserted only in the regions with more details, unlike the NURB case with the regularly structured control mesh.

There are few algorithms described in literature that explore

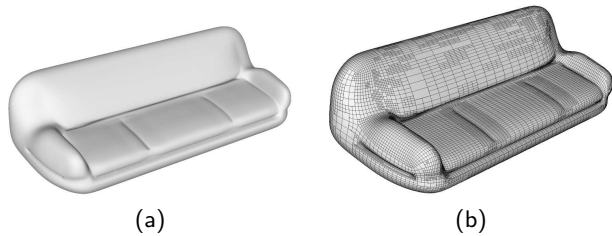


Figure 3: A sofa modeled by LOD approach. (b) shows the *quads* of the adaptively subdivided surface.

the adaptive tessellation problem in the context of the subdivision surfaces. Our method was inspired by the work of [13]. This work introduces a novel approach to adaptive subdivision without any modifications of the coarse control mesh and with *no dynamic allocations*, which is especially useful for the embedded systems – such as mobile phones, PDAs and game consoles with a limited RAM volume, small data storage, minimized instruction/data traffic, etc. Often the developers for these platforms cannot rely on GPU for the acceleration of rendering or computations.

The limitation of [13] is that *each coarse face* of the level-0 control mesh is *uniformly subdivided*. The finer subdivision for only restricted areas of the coarse face can be required, for example, due to the features introduced in higher subdivision meshes as a result of the *level of detail* (LOD) modeling approach, see Figure 2 and Figure 3. Another reason for adaptive subdivision inside the coarse face could be silhouette proximity.

Main contribution In this work we remove the constraint of [13]. In addition our method allows to introduce *sharp features and modifications of the positions of control vertices on any subdivision level*, i.e. LOD modeling, which is commonly used in subdivision modeling.

The LOD approach is natural for the designer and allows to reduce storage space required for the final model. The whole fine mesh is not stored with the model, just a *coarse base mesh* and the introduced *modification* are required for the reconstruction of the model, see Figure 2 and Figure 3. Moreover, the modifications of the fine meshes can be stored either as a scalar valued displacement map as in [9], or as a small vector valued delta in the local coordinate frame.

2 Related work

The idea of [13] is similar to that of [7, 10]: It loads one face of the mesh with the required for subdivision neighborhood to a dedicated process/function that computes its subdivision up to a given depth and renders the result. The subdivision depth passed to this function may vary from face to face, which results in the *adaptively subdivided* mesh. The criteria for the subdivision depth parameter comprise the curvature of the limit surface patch, its projected size, visibility and silhouette proximity. Both [13] and [10] use statically

allocated data structure of a fixed size to avoid any *dynamic allocations*. We describe the data structure of [13] in Section 3.2.

In [2] rapid evaluation of subdivision surfaces is performed with the aid of the precomputed tessellations of basis functions for each valence and configuration of special features.

In [3, 14] GPU oriented evaluation of subdivision surfaces is introduced. These works also employ the local nature of the subdivision schemes to process faces independently.

3 Rendering subdivision surfaces

The whole rendering process resembles that of [13]. Each coarse face is passed to the rendering function that initializes static data structure with the vertices of the face and their neighbors. The output of the function is a set of rendered polygons (triangles) that visualize a *smooth patch of the limit surface* corresponding to the input face.

Our static data structure unlike the *slate* of [13] keeps a *hierarchy* of tables for the computations. For each subdivision level $k = 0, \dots, \text{max}_{level}$ there is a table of the size $(2^k + 3) \times (2^k + 3)$ that stores the geometry of the control mesh on the level k . The space complexity of this data structure does not exceed that of [13], i.e. the size of the two tables keeping the geometry for the maximal resolution: $k = \text{max}_{level}$, see Section 3.2 for more details. We also store a similar set of tables with *pseudo-faces*.

The pseudo-face structure serves to indicate the following:

- The face is *active*: it is initialized and its control points are computed and written to the table;
- The face is *subdivided*: the four corresponding next level faces are active;
- Edge/vertex boolean tags (sharp/semi-sharp/boundary, etc).

The additional information allows to subdivide a face *on any level* according to some *oracle* function that takes into account curvature of the surface, visibility, silhouette proximity and special feature presence on *this* subdivision level. The information is also used for closing the gaps between the differently subdivided quads.

In our work we used *Catmull-Clark* subdivision scheme, although the method is not restricted to it and the extension to other schemes is straightforward.

3.1 Catmull-Clark subdivision rules

Catmull-Clark, described in [4], is one of the most popular subdivision schemes, see [18]. It is based on the tensor product bicubic spline and produces surfaces that are C^2 everywhere except at the extraordinary vertices, where it is C^1 , see [11].

The scheme is defined for meshes with quadrilateral faces, with generalized rules that admit general meshes as well,

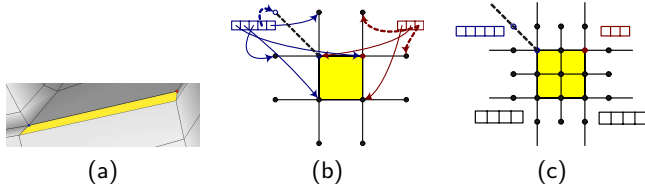


Figure 4: (a) is a face of the base mesh to be subdivided. (b) shows an initialization of the level-0 table for this face. Blue and red corners are of *non-regular* valence 5 and 3 respectively. 1D arrays of (b) store the positions of their immediate neighbors. (c) is the table storing positions of the level-1 mesh.

see [4]. The rules can be summarized as follows:

- a *face control point* f^{j+1} is computed as the average of its corners: $f^{j+1} = \frac{1}{n} \sum_{i=0}^n v_i^j$;
- an *edge control point* e^{j+1} is the average of the edge endpoints and the newly computed adjacent face control points: $e^{j+1} = \frac{v_1^j + v_2^j + f_1^{j+1} + f_2^{j+1}}{4}$;
- *even control points* that correspond to the previous level control points: $v^{j+1} = \frac{k-2}{k} v^j + \frac{1}{k^2} \sum_{i=0}^{k-1} e_i^j + \frac{1}{k^2} \sum_{i=0}^{k-1} f_i^{j+1}$, where k is the valence of the vertex and n is the number of vertices in the face.

Vertices on the boundaries or creases are computed using different rules that use coefficients for the cubic splines, see [6].

Subdivision process defines a smooth *limit surface*, which locally has a closed form and can be evaluated, see [1, 15].

Further in the paper we assume that *all the faces are quadrilaterals* (or quads), which is a regular case for the Catmull-Clark scheme. Otherwise the surface should be subdivided once at the preprocessing stage.

3.2 Data structure

Denote the maximal subdivision level for the scene as k_{max} and the maximal vertex valence by val_{max} . The *slate* data structure of [13] comprises two tables of the size $(2^{k_{max}} + 3) \times (2^{k_{max}} + 3)$ for the subdivision computation and two sets of four one dimensional arrays of the size $2 \times val_{max}$ for the corner neighbors positions. The subdivision is computed by passing the current level information for the subdivision function in one of the slates and storing the computed result in the second slate. When the result is stored in the second slate, the geometry of the previous level control mesh is overwritten, which means that the base face cannot be subdivided adaptively and each *coarse face* of the level-zero control mesh is subdivided in a *uniform* fashion!

We propose a much more flexible configuration that allows one to subdivide the surface *adaptively on each level*. Our static data structure contains for *each* level $k = 0, \dots, k_{max}$ the following:

- one two-dimensional array of the size $(2^k + 3) \times (2^k + 3)$ for positions of the control vertices;

- one two-dimensional array of the size $2^k \times 2^k$ for the pseudo-faces.
- four one-dimensional arrays for the positions of the immediate neighbors (sharing an edge) of the corner vertices;
- four one-dimensional arrays for the positions of the “opposite” vertices (that do not have a common edge) in the neighboring faces of the corner vertices;

Note, that $\sum_{k=0}^{k_{max}} (2^k + 3)^2 < 2 \cdot (2^{k_{max}} + 3)^2$, for $k_{max} > 0$. For example, for $k_{max} = 4$ we have that $\sum_{k=0}^4 (2^k + 3)^2 = 572$, while $2 \cdot (2^4 + 3)^2 = 722$.

To perform one subdivision step for a face we average the face vertices and the vertices of the neighboring faces. We initialize level-zero tables similarly to [13] with the positions of the base mesh vertices. Figure 4 (b) shows an initialization of the zero-level table for the face of the initial control mesh, shown in (a). Blue and red corners have *non-regular* valence 5 and 3 respectively. One-dimensional arrays of (b) are initialized with the positions of their immediate neighbors, i.e. vertices that share an edge with the corresponding corner. Note, that there are also similar arrays for the position of the “opposite” vertices, that are not shown on the figure. These “opposite” arrays are filled with the positions of the vertices sharing a face with the corner, but not an edge.

When the faces of the level k are subdivided, the positions of the resulting control mesh are written into the $(k + 1)$ -level table.

The information stored in the pseudo-face tables is used for choosing subdivision rules – boundaries, sharp or semi-sharp edges require special subdivision masks. Moreover, we must know which faces were subdivided on each level in order to render correctly the resulting surface without any *gaps* on the surface.

3.3 Oracle function

Each *active* face, that has been initialized is tested by the *oracle* function. If oracle returns a positive answer the face is subdivided and four new faces on the next level are *activated*: the positions of their vertices are written into the tables and they are marked as active.

Main oracle We used a simple oracle function to determine if the selected face must be subdivided. The input for the oracle consists of the *positions* and the *normals* of the four vertices, see Figure 5 (a).

The oracle considers the following properties of the face:

- *visibility*: at least one normal is pointing toward the viewer; in addition a hardware accelerated depth test (for example, provided by the Open GL) may be performed;
- the *curvature* of the patch: the normals deviation within the face;

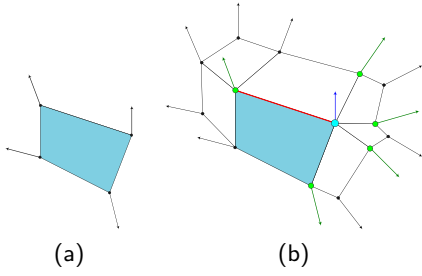


Figure 5: (a) shows input normals and vertices for the main oracle. On (b) is the data used by the *edge oracle*. Green vertices and normals are used to compute the surface curvature at the blue vertex.

- *silhouette proximity*: if the normals form the angle close to $\pi/2$ with the viewing direction, the subdivision depth is increased;
- *feature edges and edited vertices within the face on the current level*: if any such edge or vertex is present, the face is subdivided.

Several detailed descriptions of different oracle functions can be found in [8, 10, 13, 17].

Edge oracle A *different* oracle function is used to determine the subdivision depth on the *edges of the input mesh*. The reason for the computing the subdivision depth along the coarse edges is the desire to *seamlessly* render all the patches as one surface. The rendering process is discussed in more detail in Section 3.4. The edge oracle employs the same reasoning as the main oracle function. The difference is that the positions and normals of the *edge end-points and their neighbors on the base mesh* are sent as an input to the edge oracle, see Figure 5 (b).

The computation of the subdivision depths on the coarse edges is performed at a preprocessing stage and stays valid as long as the geometry of the base mesh does not change.

Consistency The edges that share a vertex must not have drastically different subdivision depths. This property is enforced by the properties of the shared vertex that is passed as the input to the oracle for both edges. In order to emphasize this consistency the edge oracle works as follows. First it evaluates the depths d_0 and d_1 separately using the positions and the normals of the neighbors at each edge vertex, see Figure 5 (b). The resulting edge subdivision depth is computed by averaging the two values. It’s also possible to enable even more flexible adaptive subdivision by setting a *varying subdivision depth along the edge* changing it gradually from d_0 to d_1 .

The edge and the main oracle functions must return consistent results, hence the two functions must work similarly and use the same reasoning.

Additional constraints The resulting *edge depths* are passed as hard constraints for the subdivision function and

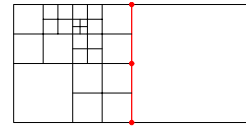


Figure 6: The subdivision depth between the adjacent faces changes *continuously*.



Figure 7: Seamless rendering of an adaptively subdivided mesh requires a valid mesh structure – no T-junctions.

are taken into account by the *main oracle*.

One more reasonable constraint is that the subdivision depth between the adjacent faces must change *continuously*. The maximal difference between the two adjacent sub-faces equals *one* inside the coarse face. Moreover, the difference between the depth of the sub-face adjacent to the coarse base edge and the precomputed depth of that edge does not exceed one as well, see Figure 6.

These constraints enable to define a simple tessellation strategy that provides a valid mesh for the adaptively subdivided surface with no T-junctions, see Figure 7, and hence no rendering artifacts such as “cracks” between the faces.

3.4 Rendering and gaps prevention

Finally rendering the resulting surface we expect to see a smooth surface, except for the predefined creases.

We need to render all the faces projected to the limit surface with the real limit surface normals. Otherwise the positions of the vertices on the edges between differently subdivided quads would be different. Moreover, the input to the oracle must also be positions and normals of the limit surface. The evaluation of the limit surface on the coarse levels is reused for the *even* control points of the finer levels.

Rendering an adaptively subdivided mesh we may get *gaps* between differently subdivided faces, see T-junction topology of the mesh on Figure 6. To eliminate the gaps we must render a tessellated version of the output which does not have any T-junctions, see Figure 7. There are various ways to produce the tessellation, see [7, 10, 13].

Each of the methods requires information about the subdivision depths of the neighboring faces. This information is available from the pseudo-face tables for the sub-faces within the same base face.

In [7, 10, 13] the subdivision depths of the faces of the initial mesh is computed before the actual subdivision is performed and the subdivision of each base face is *uniform*.

In our work the sub-faces that are adjacent to the base mesh edges are tessellated using the precomputed results of the corresponding *edge oracle*. Recall that the difference between the subdivision depths of the adjacent faces may not exceed one level. This is a reasonable constraint since we are rendering a *smooth* surface and its curvature is continuous everywhere except for the creases. The same is true for the difference between the subdivision depth of the face adjacent to the base mesh edge and the precomputed depth of the edge.

4 Experimental results

The fully subdivided mesh on Figure 1 (a) consists of 150K quads, while the adaptively subdivided version (b) comprises less than 5K polygons (depending on the viewing direction), which leads to the reduced amount of computations and faster rendering. The reason for the reduction of the size of the rendered mesh is the full flexibility of the method: the faces on *each level* are *not subdivided uniformly*.

In the example of the Figure 3 the number of quads in the level 6 subdivision mesh is 123K, while an adaptively subdivided model contains just 12K faces. The model contains 743 modifications on various subdivision levels – control vertices moved to new locations and sharp edges introduced on non-zero subdivision levels. Every face containing at least one modification was subdivided to avoid visual differences with fully subdivided model. The base faces with the modifications cover more than half of the model and if one does not have the option of adaptive subdivision inside the face on each level half of the surface would be subdivided up to the maximal level, making the rendering much more expensive.

5 Conclusion

Our method allows to produce high-quality rendering of 3D scenes interactively without *dynamic allocations*. The technique is more flexible than the previous ones and thus is well suited for the rendering of LOD models and models with fine features.

The approach is especially useful for the embedded systems that have a limited RAM volume and require minimized instruction/data traffic, etc. GPU for the acceleration of rendering or computations is not always available on such platforms.

Although it is also possible to use GPU for the subdivision computations with our technique since it handles every face independently of the rest.

6 Acknowledgments

The author would like to thank Vitaly Surazhsky for interesting and fruitful discussions, Adam Gur and Shlomi Avtalyon for creating the illustrative 3D subdivision models. This research was supported by Samsung Telecom Research Israel.

References

- [1] H. BIERMANN, A. LEVIN, AND D. ZORIN, *Piecewise smooth subdivision surfaces with normal control*, in Proc. of SIGGRAPH'00, 2000, pp. 113–120.
- [2] J. BOLZ AND P. SCHRÖDER, *Rapid evaluation of catmull-clark subdivision surfaces*, in Proc. of Web3D'02 Symposium, February 2002, pp. 11–17.
- [3] J. BOLZ AND P. SCHRÖDER, *Evaluation of subdivision surfaces on programmable graphics hardware*, <http://www.multires.caltech.edu/pubs/GPUSubD.pdf>.
- [4] E. CATMULL AND J. CLARK, *Recursively generated b-spline surfaces on arbitrary topological meshes*, Comp. Aided Design, (1978), pp. 350–355.
- [5] DASSAULT SYSTEMES, *Catia – Imagine & Shape (IMA)*, Web Resource. <http://www.3ds.com/home/>, 2005.
- [6] T. DEROSE, M. KASS, AND T. TRUONG, *Subdivision surfaces in character animation*, in Proc. of SIGGRAPH'98, Annual Conference Series, ACM, 1998, pp. 85–94.
- [7] S. HAVEMANN, *Interactive rendering of catmull/clark surfaces with crease edges.*, The Visual Computer, 18 (2002), pp. 286–298.
- [8] S. LAI AND F. F. CHENG, *Adaptive rendering of catmull-clark subdivision surfaces*, in Proc. of Comp. Aided Design and Comp. Graphics, Dec. 2005, pp. 125–132.
- [9] A. LEE, H. MORETON, AND H. HOPPE, *Displaced subdivision surfaces*, in Proc. of SIGGRAPH'00, 2000, pp. 85–94.
- [10] K. MÜLLER AND S. HAVEMANN, *Subdivision surface tessellation on the fly using a versatile mesh data structure*, Computer Graphics Forum, 19(3) (2000), pp. 151–159.
- [11] U. REIF, *A unified approach to subdivision algorithms near extraordinary vertices.*, Computer Aided Geometric Design, 12 (1995), pp. 153–174.
- [12] T. SEDERBERG, D. L. CARDON, G. T. FINNIGAN, J. ZHENG, AND T. LYCHE, *T-spline simplification and local refinement*, ACM Transaction on Graphics, 23 (2004), pp. 276–283.
- [13] V. SETTGAST, K. MÜLLER, C. FÜNFZIG, AND D. W. FELLNER, *Adaptive tessellation of subdivision surfaces*, Computers & Graphics, (2004), pp. 73–78.
- [14] L.-J. SHIUE, I. JONES, AND J. PETERS, *A realtime GPU kernel*, ACM Trans. on Graphics, 24 (2005), pp. 1010–1015.
- [15] J. STAM, *Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values*, in Proc. of SIGGRAPH'98, 1998, pp. 395–404.
- [16] VALVE CORPORATION, *Half-life 2*, Web Resource. <http://half-life2.com/>, 2004.
- [17] X. WU AND J. PETERS, *An accurate error measure for adaptive subdivision surfaces*, in Proc. of Shape Modeling and Applications, June 2005, pp. 51–56.
- [18] D. ZORIN AND P. SCHRÖDER, *Subdivision for Modeling and Animation*, SIGGRAPH'00 Course Notes, 2000.