MeshToSS: Converting Subdivision Surfaces from Dense Meshes

Takashi Kanai

Keio University, Faculty of Environmental Information
Endo 5322, Fujisawa-city, Kanagawa, 252-8520, Japan
Email: kanai@sfc.keio.ac.jp

Software Web page:
http://graphics.sfc.keio.ac.jp/MeshToSS/

Abstract

The theoretical aspects are discussed of our developed software, `MeshToSS`, for the conversion to Loop subdivision surfaces of dense triangular meshes. This software generates subdivision surfaces that approximate to their original mesh. The output is either a control mesh or its subdivided mesh applied to the Loop subdivision scheme. Our simple approach for the conversion is based on a well-known mesh simplification technique that applies a sequence of edge collapse operations. We show that this enables fast and flexible conversion to subdivision surfaces.

1 Introduction

An unstructured triangular mesh is the most common surface representation and is widely used in computer graphics (CG). 3D objects which have both arbitrary connectivity and complicated geometry can be defined by using meshes. Such meshes can be obtained by the reconstruction of range images, although it is quite difficult for the user to handle these meshes.

On the other hand, a subdivision surface [12] is one of the surface representations that can define a smooth surface by repeatedly and infinitely subdividing a coarse control mesh according to a certain rule. In particular, a network of C^1 continuous triangular surfaces with arbitrary connectivity can be defined by the Loop subdivision scheme [8]. The control mesh of a Loop surface is defined much like a triangular mesh, and then the utilization of various algorithms for triangular meshes (e.g., compression and transmission) is possible.

There are some approaches for converting subdivision surfaces from a range of points or polygonal meshes. Hoppe et al. [5] have proposed an automatic fitting method to subdivision surfaces from a range of points. This approach is, however, based on global non-linear optimization requiring considerable computing time. Suzuki et al. [10] have proposed a rapid local fitting scheme to generate subdivision surfaces, although users have to manually input a coarse control mesh.

This paper introduces an algorithm implemented in our `MeshToSS` software for converting polygonal meshes to Loop subdivision surfaces. This software reads VRML polygon data as the input, converts to a subdivision surface, and then generates the control mesh of a surface or a semi-regular structured mesh that is regularly subdivided by a 4-to-1 split scheme.

Our approach for converting Loop surfaces is an extension of the mesh simplification algorithm using the QEM (*quadric error metric*) method proposed by Garland and Heckbert [3] and incorporates some significant properties from the original scheme. Firstly, it can preserve shape features such as boundaries, creases and so on by overlaying the edges of a control mesh along these features.[1] Secondly, the computation time is much faster than that of other optimization-based conversion approaches, because the energy minimization is simplified by using QEM. Thirdly, we can construct a continuous multiresolution representation of subdivision surfaces as a progressive mesh [4] by using our scheme. This enables more flexible resolution control of the subdivision surfaces. We can obtain from

[1]Our approach does not deal with piecewise smooth subdivision surfaces, so creases are converted to shapes with small rounding.

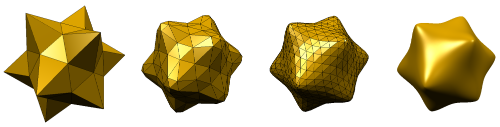
Figure 1: Loop subdivision surface.

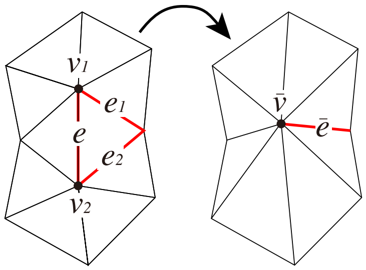

Figure 2: Edge collapse operation $\{v_1, v_2\} \rightarrow \bar{v}$.

our software a Loop surface with arbitrary resolution according to the number of vertices or to an approximation error specified by the user.

2 Basic Setup

We describe in this section the Loop subdivision scheme and QEM-based mesh simplification that are used in our software.

2.1 Loop Subdivision Scheme

We explain here an overview of the Loop subdivision scheme [8] to show some basic features of subdivision surfaces. In the scheme of a subdivision surface, repeatedly and infinitely applying subdivision steps defines a smooth surface. Each subdivision step generally consists of two sub-steps called *splitting* and *positioning*.

In the splitting step, the Loop subdivision scheme uses a 4-to-1 split. A vertex is inserted in each edge to generate two sub-edges. Each triangular face is subdivided into four sub-faces. The inserted vertex and an original vertex are respectively called the *odd vertex* and *even vertex*. In the positioning step, each new vertex position is calculated by a linear combination of its 1-ring neighbor vertices.

An infinitely subdivided triangular mesh converges to a smooth surface (Figure 1). A Loop surface has the generalized form of a three-dimensional box spline. Loop [8] has shown that the limit surface is C^2-continuous, except at extraordinary vertices, where it is C^1-continuous.

2.2 QEM-based Mesh Simplification

We explain here an overview of the original QEM-based mesh simplification proposed by Garland and Heckbert [3].

For each face f of mesh M, quadric function $Q^f(\mathbf{v})$ is defined by $Q^f(\mathbf{v}) = (\mathbf{n}^T\mathbf{v} + d)^2$ as the square of the distance from the point $\mathbf{v}$ to a plane including face f, where $\mathbf{n}$ denotes the normal vector of f. For each vertex v of M, a quadric function is defined as follows:

$$\begin{aligned} Q^v(\mathbf{v}) &= \sum_{f \ni v} \mathrm{area}(f) \cdot Q^f(\mathbf{v}) \\ &= \mathbf{v}^{\mathrm{T}}\mathbf{A}\mathbf{v} + 2\mathbf{b}^{\mathrm{T}}\mathbf{v} + c, \end{aligned} \tag{1}$$

where $\mathbf{A}$, $\mathbf{b}$ and c respectively denote a 3×3 symmetric matrix, a column vector and a scalar, and $\mathrm{area}(f)$ denotes the area of triangular face f. The reason for adopting a weighted sum using $\mathrm{area}(f)$ is so that Q^v depends on the size of the triangular face (see [2] for more details). Consequently, $6 + 3 + 1 = 10$ floating point values consisting of $Q^v = (\mathbf{A}, \mathbf{b}, c)$ are stored in each vertex v. Note that $Q^v(\mathbf{v}) = 0$ at the beginning of the algorithm.

The algorithm applies an edge collapse operation $\{v_1, v_2\} \rightarrow \bar{v}$ (Figure 2) to simplify M. In this operation for edge e, we calculate position $\mathbf{v}^{min}$ of vertex $\bar{v}$ that minimizes the sum of QEMs of the two end vertices, $Q^{\bar{v}} = Q^{v_1} + Q^{v_2}$. This is the position where gradient $\nabla Q^{\bar{v}}$ is zero, and $\mathbf{v}^{min}$ can be found by solving the following linear equation:

$$\mathbf{A}\mathbf{v}^{min} = -\mathbf{b}. \tag{2}$$

The cost of each edge e is defined by $Q^{\bar{v}}(\mathbf{v}^{min})$. We calculate the costs for all edges in advance at the beginning of the algorithm. Edge collapse operations are processed in order of the edges with the lowest to highest cost. The costs of neighbor edges are updated after each operation.

3 Approximating Subdivision Surfaces

In this section, we discuss the method for directly calculating the control mesh of a Loop surface from

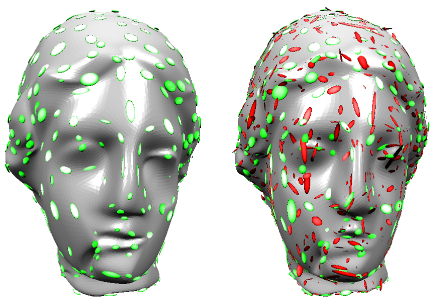

Figure 3: QEM visualization: (a) QEM management by vertices only, (b) QEM management by vertices and edges.

a mesh by using the QEM-based mesh simplification. The principle is to define an evaluation function based on using subdivided vertices by the Loop subdivision scheme, and not the vertices of the control mesh themselves.

3.1 Vertex-Edge QEM Management

QEMs are stored only by vertices in the original mesh simplification algorithm [3]. To find an optimized vertex position in each edge collapse operation, only a new vertex is used to evaluate a QEM. For our purpose, we need to use a new vertex and its neighbor vertices to establish a more accurate and sophisticated evaluation. We therefore established another QEM management method to do this.

In both vertices and edges QEMs are stored. At the beginning of the algorithm, each edge e stores the following QEM (called an *edge QEM*):

$$\mathcal{Q}^{e} = \mathrm{area}(f_l)\cdot \mathcal{Q}^{f_l} + \mathrm{area}(f_r)\cdot \mathcal{Q}^{f_r}, \tag{3}$$

where f_l and f_r denote faces neighboring e. If e is a boundary edge, only either element of Equation (3) is defined. All elements of a QEM in each vertex (called a *vertex QEM*) are set to zero. In each edge collapse operation, the QEM of deleted edge e is integrated in that of new vertex $\bar{v}$. To generalize this, QEMs are updated at each edge collapse operation as follows (see Figure 2 for the notation):

$$\begin{cases} \mathcal{Q}^{\bar{v}} = \mathcal{Q}^{\bar{e}} + \mathcal{Q}^{v_1} + \mathcal{Q}^{v_2}, \\ \mathcal{Q}^{\bar{e}} = \mathcal{Q}^{e_1} + \mathcal{Q}^{e_2} \end{cases} \tag{4}$$

Figure 3 demonstrates the visualization of QEMs according to the method in [3]. The iso-surface of

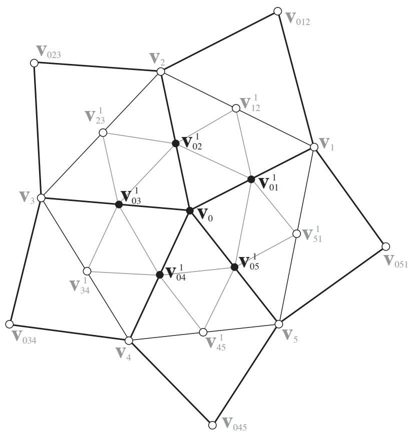

Figure 4: Neigboring vertices centered at vertex $\mathbf{v}_0$.

a QEM can be represented by an ellipsoid. Figure 3(a) denotes QEMs managed only by vertices. QEMs are sparsely situated on the control mesh if the number of vertices is too few. Figure 3(b) denotes QEMs managed by both vertices and edges. A light-colored ellipsoid is a vertex QEM, and a dark-colored ellipsoid is an edge QEM. It can been seen that the ellipsoids are distributed more densely, even if the number of vertices of the control mesh is the same as that shown in Figure 3(a). The size of an ellipsoid indicates the scale of floating point values of a QEM. A vertex QEM tends to be larger than an edge QEM in a simplified mesh, because QEMs are more accumulated at vertices as simplification proceeds by our vertex-edge QEM management method.

3.2 Evaluation Function Using Subdivision Points

Our evaluation function for optimizing a vertex position is defined by using points on the Loop surface and not using vertices of the control mesh. We use vertices of a mesh subdivided twice from the control mesh, instead of using exact points (limit points) on the Loop surface. This is done because the calculation is simpler, both results not being any different in our experiment. A twice-subdivided mesh by the Loop scheme is similar to its limit surface in most cases. The calculation of an arbitrary point on a Loop surface is possible, however, at least a twice-subdivided mesh being needed for this (see [9] for more details).

We will next explain the notation of neighboring vertices centered at vertex $\mathbf{v}_0$ of the control mesh after an edge collapse operation (Figure 4). A superscript number indicates the number of subdivision; for example, $\mathbf{v}^{(1)}$ is a vertex subdivided once. $\mathbf{v}_j$ $(j = 1 \ldots \kappa_0)$ shows a 1-ring neighbor vertex of the control mesh, where κ_0 denotes the valence of $\mathbf{v}_0$. $\mathbf{v}_{0j}^{(1)}$ shows an odd vertex that has been subdivided once and generated at the midpoint of an edge.

Our evaluation function can be defined by the following equation:

$$\mathcal{Q}(\mathbf{v}_0) = \mathcal{Q}^{v_0}(\mathbf{v}_0^{(2)}) + \sum_{j}^{\kappa_0} \mathcal{Q}^{e_j}(\mathbf{v}_{0j}^{(2)}), \qquad (5)$$

where $\mathbf{v}_0^{(2)}$ denotes an even vertex subdividing $\mathbf{v}_0$ twice, and $\mathbf{v}_{0j}^{(2)}$ denotes an even vertex subdividing odd vertex $\mathbf{v}_{0j}^{(1)}$ once more. The first element of the right term in Equation (5) is for shape evaluation around vertex $\mathbf{v}_0^{(2)}$, for which vertex QEM $\mathcal{Q}^{v_0} = (\mathbf{A}_0, \mathbf{b}_0, c_0)$ of vertex v_0 is used. The second element is for shape evaluation around edge e_j neighboring v_0, for which edge QEM $\mathcal{Q}^{e_j} = (\mathbf{A}_{0j}, \mathbf{b}_{0j}, c_{0j})$ of edge e_j is used.

We regard $\mathbf{v}_0$ as the only variable for calculating the optimized vertex position so that we need to find $\mathbf{v}_0$ minimizing $\mathcal{Q}(\mathbf{v}_0)$. As discussed in Section 2.1, $\mathbf{v}_0^{(2)}$ and $\mathbf{v}_{0j}^{(2)}$ can each be represented as a linear function of $\mathbf{v}_0$, so they are rewritten as follows:

$$\begin{aligned} \mathbf{v}_0^{(2)} &= \alpha_0 \mathbf{v}_0 + \mathbf{e}_0, \\ \mathbf{v}_{0j}^{(2)} &= \alpha_{0j} \mathbf{v}_0 + \mathbf{e}_{0j} \end{aligned} \qquad (6)$$

where α_0 and α_{0j} denote the coefficients of element $\mathbf{v}_0$, $\mathbf{e}_0$ and $\mathbf{e}_{0j}$ denote vectors summed all other vertex positions and coefficients. Combining Equation (5) and (6):

$$\begin{aligned} \mathcal{Q}(\mathbf{v}_0) = \mathbf{v}_0^{\mathrm{T}} & \left(\alpha_0^2 \mathbf{A}_0 + \sum_{j=1}^{\kappa_0} \alpha_{0j}^2 \mathbf{A}_{0j} \right) \mathbf{v}_0 \\ & + 2\alpha_0 (\mathbf{A}_0 \mathbf{e}_0 + \mathbf{b}_0)^{\mathrm{T}} \mathbf{v}_0 \\ & + 2 \sum_{j=1}^{\kappa_0} \alpha_{0j} (\mathbf{A}_{0j} \mathbf{e}_{0j} + \mathbf{b}_{0j})^{\mathrm{T}} \mathbf{v}_0 \\ & + \mathbf{e}_0^{\mathrm{T}} \mathbf{A}_0 \mathbf{e}_0 + 2\mathbf{b}_0^{\mathrm{T}} \mathbf{e}_0 + c_0 \\ & + \sum_{j=1}^{\kappa_0} \left(\mathbf{e}_{0j}^{\mathrm{T}} \mathbf{A}_{0j} \mathbf{e}_{0j} + 2\mathbf{b}_{0j}^{\mathrm{T}} \mathbf{e}_{0j} + c_{0j} \right) \\ = \mathbf{v}_0^{\mathrm{T}} \bar{\mathbf{A}} \mathbf{v}_0 + 2\bar{\mathbf{b}}^{\mathrm{T}} \mathbf{v}_0 + \bar{c}. & \qquad (7) \end{aligned}$$

$\mathcal{Q}(\mathbf{v}_0)$ is also a quadric function of $\mathbf{v}_0$. Vertex position $\mathbf{v}_0^{min}$ minimizing $\mathcal{Q}$ can be found by solving the following 3×3 linear system:

$$\bar{\mathbf{A}} \mathbf{v}_0^{min} = -\bar{\mathbf{b}}. \qquad (8)$$

3.3 Algorithm

We next describe our algorithm for converting a Loop subdivision surface from mesh M. This is a natural extension of the original mesh simplification algorithm in [3].

1. Calculate 10 floating point values for each edge QEM and store the results in that edge. All elements of each vertex QEM are set to zero.
2. For each edge e, compute $\mathbf{v}_0^{min}$ and cost $\mathcal{Q}(\mathbf{v}_0^{min})$. This is done by calculating $\bar{\mathbf{A}}$, $\bar{\mathbf{b}}$ and $\bar{c}$, and by solving Equation (8). e is inserted in a heap, the top of the heap being an edge of the minimum cost.
3. For e at the top of the heap, run the edge collapse operation $(\mathbf{v}_1, \mathbf{v}_2) \rightarrow \bar{\mathbf{v}}$ $(= \mathbf{v}_0^{min})$ and delete e from the heap. Update QEMs according to the rule in Equation (4) and costs of all edges including $\bar{\mathbf{v}}$.
4. The algorithm terminates if the number of vertices or an approximation error (we use here the square root of $\mathcal{Q}(\mathbf{v}_0)$ for simplicity) reaches a prescribed value.

One of the differences between the original scheme and ours is that our algorithm needs to calculate $\bar{\mathbf{A}}$, $\bar{\mathbf{b}}$ and $\bar{c}$ at each edge collapse stage. This leads to extra computation time. The original scheme does not have to do this because a vertex QEM can be directly used for the evaluation.

The other difference is in space efficiency; our algorithm requires a roughly four times larger memory space because QEMs must be stored in both vertices and edges. Furthermore, our algorithm cannot use a memory-efficient simplification scheme like the "memoryless simplification" proposed by Lindstrom and Turk [7] that does not need to store QEM explicitly, because it is computed on the fly with simplification. The mesh processed with our algorithm is the control mesh of a subdivision surface, and is not similar to its original mesh. Consequently, our method cannot use a simplified mesh directly for evaluating the shape.

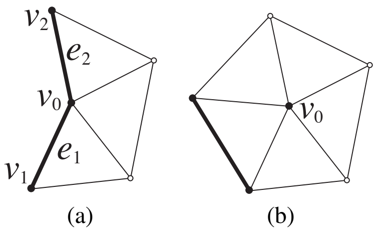

Figure 5: Boundary case.

3.4 Evaluation Function of the Boundary

We will discuss the case of a mesh with boundary edges. An additional QEM is stored in boundary edge e, this being the same function as that defined in [3]. Assume a plane perpendicular to a face neighboring a boundary edge such that $(\mathbf{n}\times\mathbf{l})^T\mathbf{v}+d' = 0$, where $\mathbf{n}$ denotes the normal vector of the face. We define an additional QEM as the square of the distance between a vertex and the plane:

$$\mathcal{Q}'^{e_x}(\mathbf{v}) = \gamma((\mathbf{n}\times\mathbf{l})^T\mathbf{v}+d')^2 \quad (x=1,2) \quad (9)$$

where $\mathbf{l}$ denotes the directional vector of e, and γ denotes a scalar quantity as a penalty. We set $\gamma = 1,000.0$.

We will consider two cases for the calculation of an evaluation function near the boundaries as shown in Figure 5. The bold lines in Figure 5 indicate boundary edges. One case is for v_0 being a boundary vertex (Figure 5(a)), for which we define the evaluation function as follows:

$$\begin{aligned}\mathcal{Q}(\mathbf{v}_0) = {} & \mathcal{Q}^{v_0}(\mathbf{v}_0^{(2)}) \\ & + \mathcal{Q}^{e_1}(\mathbf{v}_{01}^{(2)}) + \mathcal{Q}^{e_2}(\mathbf{v}_{02}^{(2)}) \\ & + \mathcal{Q}'^{e_1}(\mathbf{v}_0^{(2)}) + \mathcal{Q}'^{e_2}(\mathbf{v}_0^{(2)}). \qquad (10)\end{aligned}$$

The first element evaluates twice subdivided vertex $\mathbf{v}_0^{(2)}$ by using the vertex QEM of v_0. The second and third elements evaluate even vertices $\mathbf{v}_{01}^{(2)}$ and $\mathbf{v}_{02}^{(2)}$ by using edge QEMs of boundary edges e_1 and e_2, respectively. Boundary subdivision rules are adopted for these vertices. Since $\mathbf{v}_0^{(2)}$, $\mathbf{v}_{01}^{(2)}$ and $\mathbf{v}_{02}^{(2)}$ are also represented as a linear function of $\mathbf{v}_0$ as shown in Equation (6), an optimized position minimizing $\mathcal{Q}$ can be found by solving the linear system. We can also consider in Equation (10) that internal edges neighboring $\mathbf{v}_0$ should be included for the evaluation of $\mathcal{Q}(\mathbf{v}_0)$. We found, however, in our experiments that it was meaningless to use these

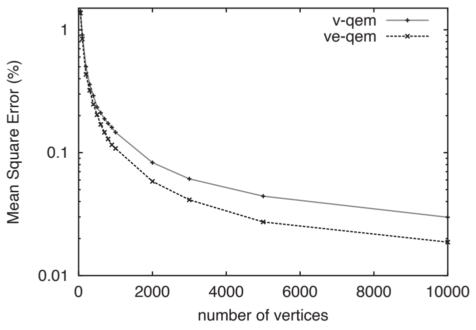

Figure 6: Mean square errors plotted between the subdivision surfaces and original mesh. The horizontal axis denotes the number of vertices in the control mesh.

elements, because the optimized position of $\mathbf{v}_0$ is strongly pulled toward the boundaries by penalty elements $\mathcal{Q}'^{e_x}$.

The other case is for the edge between two 1-ring neighbor vertices of v_0 being a boundary edge as shown in Figure 5(b). In this case, we basically use the evaluation function of Equation (5) for optimization, provided that boundary subdivision rules are adopted for the boundary vertices.

4 Experimental Results

Figure 9 demonstrates the results of converting Loop subdivision surfaces for the Stanford bunny model (35,947 vertices and 69,451 faces). We tested here two QEM management schemes: Case 1 involves vertex management of the original QEM scheme (v-qem) shown in Figure 9 (d)-(e), a simplified evaluation function in Equation (5) considering only the first element; Case 2 is the vertex-edge QEM management method proposed in Section 3.1 (ve-qem) and shown in Figure 9 (f)-(g). As can be seen from the results presented in Figure 9, subdivision surfaces converted by using both Case 1 and Case 2 principles preserve features such as bumps and creases. This indicates that our approach inherits the good aspects of the original QEM scheme. In the comparison between control meshes with 1000 vertices, we cannot recognize any visual difference between them; for 300 vertices, however, slight differences are apparent around the ears and nose.

A more intelligible comparison between Case 1 and Case 2 is given in Figure 6 which shows a graph of the relationship between the number of vertices

type	qem manag.	cont. mesh v	cont. mesh f	sub. mesh f	time (sec.)
simp.	v-qem	1,000	1,907	-	50.4
simp.	v-qem	300	571	-	51.6
sub.	v-qem	1,000	1,908	30,528	58.3
sub.	v-qem	300	570	9,120	59.2
sub.	ve-qem	1,000	1,949	31,184	137.8
sub.	ve-qem	300	582	9,312	139.3

Table 1: Statistical summary of the examples in Figure 9. From left to right: conversion type (simp. or sub.), QEM management type (v-qem or ve-qem), number of vertices and faces in the control mesh, the number of faces in the twice subdivided mesh, and the computation time.

of the control meshes and the approximation errors. The horizontal axis of this graph shows the number of vertices in the control mesh. The vertical axis shows the percentage on a logarithmic scale of the mean square error (L^2-norm) over the diagonal length of the bounding box of a mesh. These errors between the subdivision surface (actually the mesh subidivided twice) and its original mesh were measured with an IRI-CNR Metro tool [1]. It can be seen that both graphs progressively decrease as the number of vertices increases. This is another characteristic of the QEM-based simplification scheme. We can use this property as a criterion for the error control of subdivision surfaces, making it possible to create a control mesh within an approximation error determined by the user. Comparing the graphs for Case 1 and Case 2, the subdivision surfaces from Case 2 generally have lower approximation errors than those from Case 1. This proves that our vertex-edge QEM management scheme produces more tightly fitted subdivision surfaces, except when the number of vertices is extremely low (< 100).

Table 1 summarizes the number of elements and the computation time for the examples in Figure 9. The computation times were collected in a standard AT-compatible PC environment (Pentium III 500MHz CPU, 256MB memory). We also show the results of the mesh simplification (simp.) in our implementation for comparison. The computation time for Case 2 is twice as long as that for a mesh simplification, while the time for Case 1 is almost the same. This is mainly due to the larger numbers of vertices needed to compute an optimized vertex.

Figure 10 demonstrates other results, especially for regular meshes (sphere, 14,582 vertices; knots,

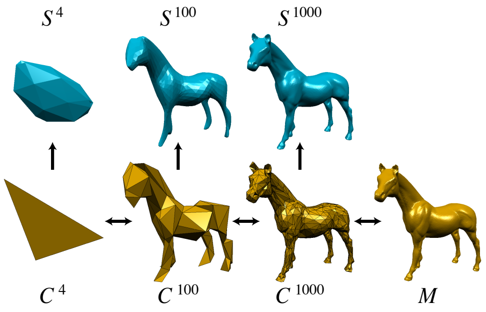

Figure 7: Progressive subdivision surface representaion.

23,232 vertices) created by a standard CAD modeling tool. These results show that we can obtain a well-defined subdivision surface that approximates to the original mesh.

5 Software Information

The `MeshToSS` software reads a mesh in the VRML format. The software outputs not only the control mesh of a Loop surface, but also the subdivided mesh. The output data format of the control mesh is also VRML, with a simple extension in the comment region so that it can be distinguished from an ordinary mesh.

The number of vertices and the approximation error can be specified to obtain a control mesh with the desired level of detail. These parameters can be specified in a dialog box. We use the percentage of the square of the evaluation function value over the size of the mesh as the parameter for the approximation error.

This free software, including source codes, Windows 98/NT/2000 executive binaries and documentations can be downloaded from the Web page. It is also possible to compile source codes on a UNIX platform such as Linux.

6 Summary and Discussion

We have presented a new algorithm for converting Loop subdivision surfaces from polygonal meshes and have introduced our `MeshToSS` software. We have shown that our conversion algorithm is fast, feature preserving, and flexible for LOD control. This software can be freely downloaded from the Web site.

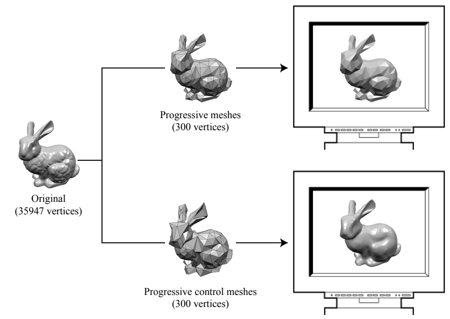

Figure 8: Application to progressive transmission.

One direction for future work is an extension of our conversion approach; for example, an application to a mesh with attributes such as color and texture coordinates, and to piecewise smooth subdivision surfaces [5] can be considered. A fitting approach to other non-triangular subdivision surfaces with our algorithm can be found in [11].

Another direction is its use for Web applications. Our approach is based on applying a sequence of edge collapse operations. Consequently, we can construct the representation of a new shape similar to that with *Progressive Mesh* (PM) [4]. As shown in Figure 7, we can traverse control meshes with a tree-structured level of detail hierarchy by using edge collapse operations and their inverse operations called *vertex split*. A subdivision surface, S^x, is obtained from each control mesh, C^x. Furthermore, we can restore the original mesh, M, by sequentially applying vertex split operations from a coarse control mesh.

Progressive transmission [6] is useful for the display of 3D models by a client/server-based networking system. The server first sends a coarse base mesh and then sequences of detail information one by one. The client receives these data and displays intermediate meshes while restoring an original mesh. We consider that our progressive representation can be used for the progressive transmission of *meshes*. As shown in Figure 8, the server sends the most coarse control mesh, and a sequence of records for vertex split operations in order. The client receives the base control mesh, constructs a subdivision surface, and displays the surface while restoring an original mesh. An intermediate subdivision surface is an approximation of the original mesh. When the transmission has been completed, the original mesh can be displayed. We imagine that more visually understandable data can be obtained, especially at the coarse level, compared to the original PM representation.

Acknowledgement

Development of this software was funded by RISE (Research Institute of Software Engineering) under support from IPA (Information-technology Promotion Agency). The “bunny” mesh is from Stanford University, the “horse” mesh and the “venus” mesh are from Cyberware Inc., and the “knot” mesh is from Washington University. We thank the reviewers for useful comments.

References

[1] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2):167–174, 1998.

[2] M. Garland. *Quadric-Based Polygonal Surface Simplification*. PhD thesis, Carnegie Mellon University, School of Computer Science, 1999.

[3] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *Computer Graphics (Proc. SIGGRAPH 97)*, pages 209–216. ACM Press, New York, 1997.

[4] H. Hoppe. Progressive meshes. In *Computer Graphics (Proc. SIGGRAPH 96)*, pages 99–108. ACM Press, New York, 1996.

[5] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle. Piecewise smooth surface reconstruction. In *Computer Graphics (Proc. SIGGRAPH 94)*, pages 295–302. ACM Press, New York, 1994.

[6] U. Labsik, L. P. Kobbelt, R. Schneider, and H.-P. Seidel. Progressive transmission of subdivision surfaces. *Computational Geometry*, 15(1–3):25–39, 2000.

[7] P. Lindstrom and G. Turk. Evaluation of memoryless simplification. *IEEE Trans. Visualization and Computer Graphics*, 5(2):98–115, 1999.

[8] C. Loop. Smooth subdivision surfaces based on triangles. Master’s thesis, University of Utah, Department of Mathematics, 1987.

[9] J. Stam. Evaluation of Loop subdivision surfaces. In *SIGGRAPH 99 Course Notes No.37 "Subdivision for Modeling and Animation"*. ACM SIGGRAPH, 1999.

[10] H. Suzuki, S. Takeuchi, T. Kanai, and F. Kimura. Subdivision surface fitting to a range of points. In *Proc. 7th Pacific Graphics International Conference (Pacific Graphics ’99)*, pages 158–167. IEEE CS Press, Los Alamitos, CA, 1999.

[11] S. Takeuchi, T. Kanai, H. Suzuki, K. Shimada, and F. Kimura. Subdivision surface fitting with QEM-based mesh simplification and reconstruction of approximated B-spline surfaces. In *Proc. 8th Pacific Graphics International Conference (Pacific Graphics 2000)*, pages 202–212. IEEE CS Press, Los Alamitos, CA, 2000.

[12] D. Zorin. Survey of subdivision schemes. In *SIGGRAPH 2000 Course Notes No.23 "Subdivision for Modeling and Animation"*. ACM SIGGRAPH, 2000.

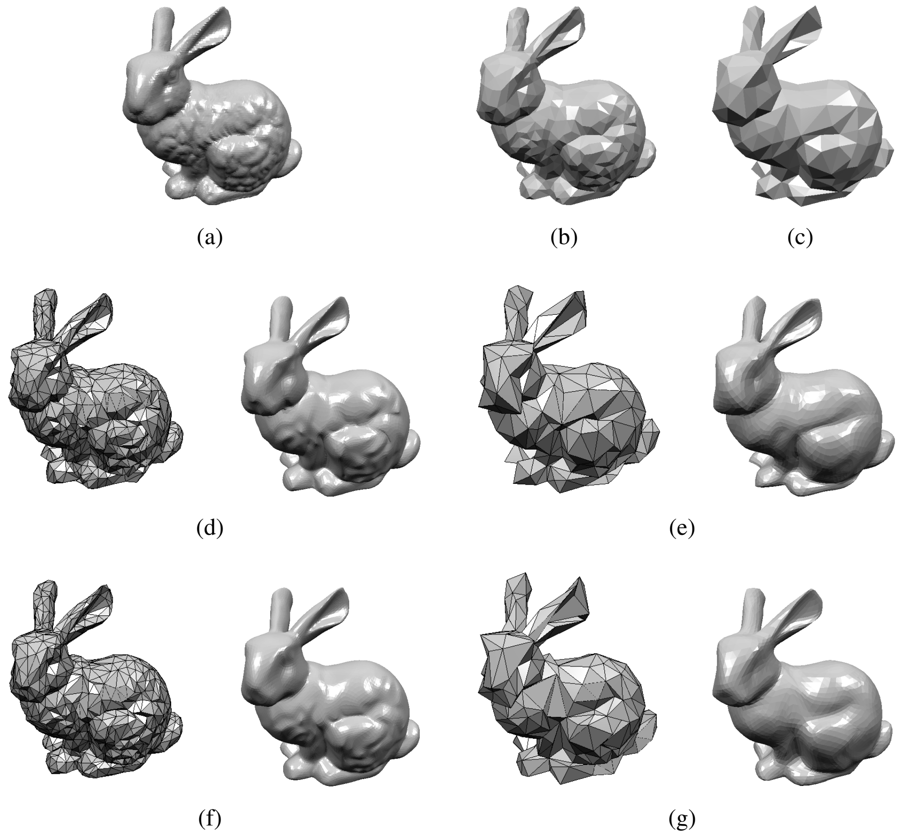

Figure 9: Conversion results for subdivision surfaces of the “bunny” model. (a) Original mesh. (b)-(c) Simplified meshes ((b) 1,000 vertices, (c) 300 vertices). (d)-(e) Results by vertex QEM management (Case 1) ((d) 1,000 vertices, (e) 300 vertices). (f)-(g) Results by vertex-edge QEM management (Case 2) ((f) 1,000 vertices, (g) 300 vertices). For (d)-(g): left: control mesh, right: subdivision surface.

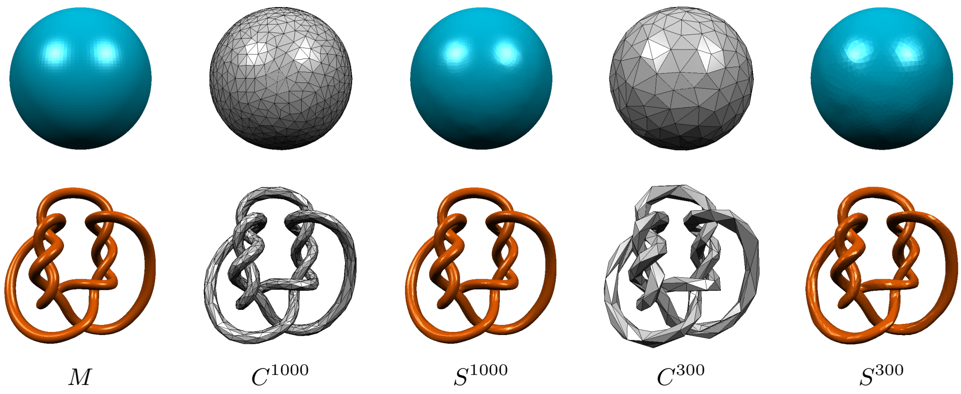

Figure 10: Conversion results for meshes with regular connectivities. Upper: the “sphere” model (14,582 vertices). Bottom: the “knot” model (23,232 vertices).