

HONEYCOMB SUBDIVISION

ERGUN AKLEMAN *

VINOD SRINIVASAN

Texas A&M University

Abstract

In this paper, we introduce a new subdivision scheme which we call honeycomb subdivision. After one iteration of the scheme each vertex becomes exactly 3-valent and with consecutive applications regular regions strongly resembles a honeycomb. This scheme can be considered as a dual for triangle schemes. The major advantage of the new scheme is that it creates a natural looking mesh structure.

1. Introduction

Although subdivision surfaces were introduced more than 20 years ago by Doo, Sabin, Catmull and Clark [5, 8], they were ignored by the computer graphics industry until they were used in 1998 Academy Award-winning short film “*Geri’s Game*” by Pixar [7, 20]. Since then subdivision surfaces have become increasingly popular in the computer graphics and modeling industry. This is not a surprise since subdivision methods solve the fundamental problem of tensor product parametric surfaces [12, 13] without sacrificing the speed of shape computation [5, 8, 11, 9, 18, 7]. Unlike tensor product surfaces, with subdivision surfaces, control meshes do not have to have a regular rectangular structure. Subdivision algorithms can smooth any 2-manifold (or 2-manifold with boundary) mesh [20, 21].

Subdivision surfaces assume that users first provide an irregular 2-manifold control mesh, \mathcal{M}_0 . By applying a set of subdivision rules, a sequence of finer and finer 2-manifold meshes $\mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_n, \dots$ are created. These meshes eventually converge to a “smooth” limit surface \mathcal{M}_∞ [10].

One way to classify subdivision schemes is based on what kind of regularity emerges with the application of the scheme [16]. In other words, the pattern of regular regions can be used to characterize the scheme. Based on regular regions, existing subdivision schemes can be classified into three major categories:

- Corner cutting schemes such as Doo-Sabin [8, 3]: after one iteration all vertices ¹ become 4-valent and the number of non-4-sided faces remains invariant after the first refinement.
- Vertex insertion schemes such as Catmull-Clark [5]: after one iteration all faces become 4-sided and the number of non-4-valent vertices is constant after the first refinement.
- Triangular schemes such as $\sqrt{3}$ -subdivision [10, 11]: after one iteration all faces become 3-sided and the number of non-6-valent vertices is constant after the first refinement.

Notice that in this list vertex insertion and corner cutting schemes are dual, i.e., one of them makes every face 4-sided, the other one makes every vertex 4-valent. There is currently no dual for triangular schemes. The scheme we present in this paper provides the missing dual for triangular schemes:

- Honeycomb schemes: after one iteration all vertices become 3-valent and the number of non-6-sided faces is constant after first refinement.

We call such schemes honeycomb since the resulting meshes strongly resemble honeycombs, which our dictionary defines as (1) A structure of hexagonal, thin-walled cells constructed from beeswax by honeybees to hold honey and larvae, (2) Something resembling this structure in configuration or pattern.

Figure 1 shows five iterations of our honeycomb scheme. In this example, the control mesh \mathcal{M}_0 is a dodecahedron, \mathcal{M}_1 is a truncated icosahedron or soccer ball [19]. More interestingly, the mesh structures from \mathcal{M}_2 to \mathcal{M}_5 can be found in virus structures [17]. As seen in this example, the most important property of the new scheme is that the resulting mesh structures strongly resemble natural structures such as cells or honeycombs. It is also interesting to note that their structure looks similar to Voronoi diagrams.

*Corresponding Author. Visualization Sciences Program, Department of Architecture. Address: 216 Langford Center, College Station, Texas 77843-3137. email: ergun@viz.tamu.edu. phone: +(979) 845-6599. fax: +(979) 845-4491.

¹Vertices and faces are also called vertets and facets in order to avoid confusion with the vertices and faces of a solid model [16].

The remainder of this paper is organized as follows. In the next section, we provide refinement rules for our general honeycomb scheme. In Section 3, we introduce the coefficients of our honeycomb algorithm that are used to create the meshes shown in Figure 1. Section 4 explains how to implement a remeshing algorithm for honeycomb subdivision. Section 5 discusses implementation and results. Finally, our conclusion is given in Section 6.

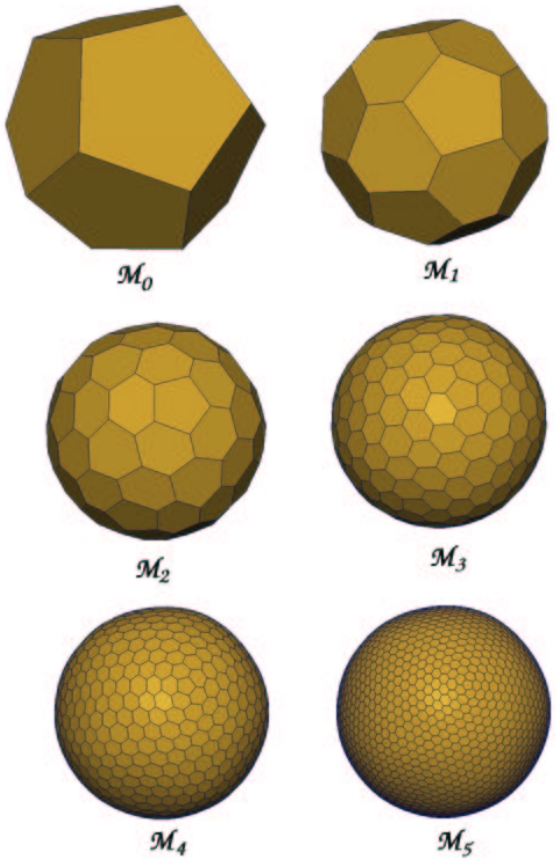


Figure 1. Five iterations of our honeycomb scheme over a dodecahedron.

2. Refinement Rules for Honeycomb Schemes

We give the refinement rules for a general honeycomb algorithm as follows (The algorithm is also illustrated in Figure 2, where the black vertices and edges are new and constitute the new mesh, and the gray vertices and edges are in the old mesh and are removed by the algorithm.):

- **Step 1 :** For each edge e_n of a face $f = \{e_0, e_1, \dots, e_n, \dots, e_{N-1}\}$ of the mesh, create a new

vertex v'_n (See Figure 2.B) (we assume that the vertex and edge indices are given in the order of a face traversing). Compute the position of each new vertex as a linear combination of the old vertex positions $\{v_0, v_1, \dots, v_n, \dots, v_{N-1}\}$. Note that since v'_n 's correspond to edges instead of vertices, in order to derive the coefficients of a honeycomb scheme, it will be useful to use the following linear equation:

$$v'_n = \sum_{m=0}^{N-1} a_{n,m} ((1-t)v_m + tv_{(m+1) \pmod{N}}) \quad (1)$$

where the $a_{n,m}$'s are real coefficients, N is the valence of the face and t is a real number between 0 and 1.

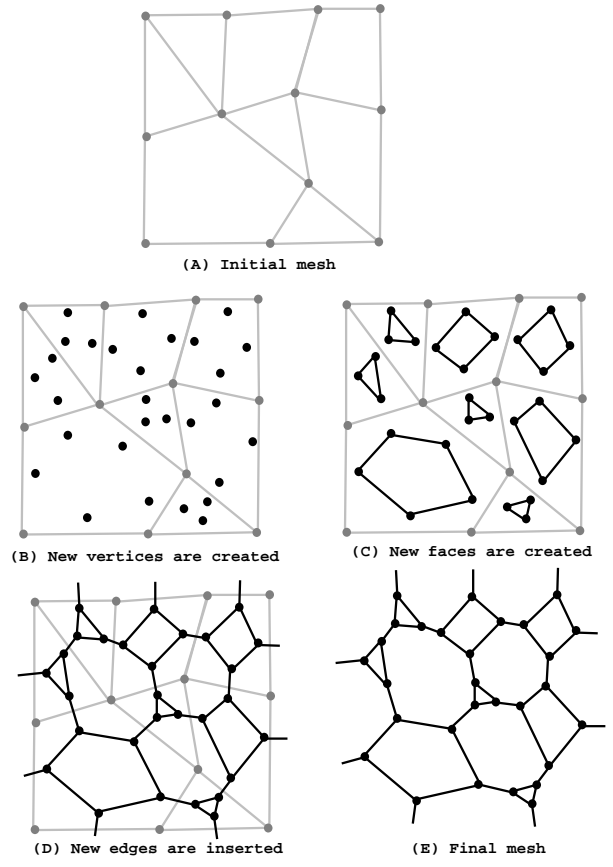


Figure 2. Illustration of honeycomb process.

Remark 1. Note that $(1-t)v_m + tv_{(m+1) \pmod{N}}$ is a point on the edge e_m . In our examples we generally use $t = 0.5$. The Figure 3 shows the effect of the parameter t . The four shapes in Figure 3 are obtained by four consecutive iterations of the honeycomb scheme for a dodecahedral control shape. The image of the shape for $t = 0.5$ is M_4 in Figure 1.

- **Step 2 :** For each face, create a new face by connecting all the new points that have been generated by that face

(See Figure 2.C);

- **Step 3 :** For each edge of the mesh, connect the two new points that have been generated for that edge (See Figure 2.D);
- **Step 4 :** Remove all old vertices and edges (See Figure 2.E).

Remark 2. Let F_n , E_n and V_n denote the number of faces, edges and vertices created by n_{th} iteration. It is straightforward to show that $F_n = F_{n-1} + V_{n-1}$, $E_n = 3E_{n-1}$ and $V_n = 2E_{n-1}$. Based on these equations, it follows that

$$\lim_{n \rightarrow \infty} \frac{F_n}{F_{n-1}} \rightarrow 3.$$

In other words, this honeycomb subdivision scheme increases the number of faces 3-fold at each iteration. Therefore, regular honeycomb scheme can also be called a $\sqrt{3}$ honeycomb algorithm [10].

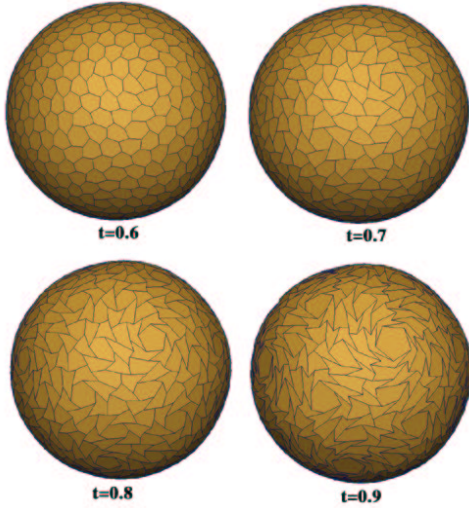


Figure 3. Effect of the parameter t .

Remark 3. The support region of our honeycomb scheme (the region that is influenced by a given control vertex [16]) has a fractal boundary, a Koch island [15].

Remark 4. It is difficult to identify the precision set of any honeycomb scheme (comparing the scheme with a parametric surface) since the regular regions of the honeycomb scheme do not correspond correspond known to any parameterization scheme such as the tensor product or box-spline parameterizations.

3. The Coefficients of Our Honeycomb Algorithm

Like any subdivision scheme relying on approximation, in honeycomb subdivision the coefficients $a_{n,m}$ in step 1 in the above algorithm must satisfy the following conditions:

1. $a_{n,m} \geq 0$ for all n and m and

2. for all n $\sum_{m=0}^{N-1} a_{n,m} = 1$.

These conditions guarantee convergence of the algorithm and provide C^0 continuity and affine invariance properties.

The coefficients of our scheme are inspired by the coefficients of the Doo-Sabin scheme. Let vertex indices in a face be given in rotation order. We use the following formula to compute the coefficients $a_{n,m}$ which are needed to compute the position of v'_n .

$$a_{n,m} = \begin{cases} a & \text{if } n = m \\ \beta \frac{1-a}{3N-5} & \text{otherwise} \end{cases} \quad (2)$$

where

$$\beta = 3 + 2 \cos \left(\frac{2(n-m)\pi}{N} \right).$$

In these equations, the parameter a in equation (2) is provided by the users and is as a tension parameter [4]. Figure 4 illustrates the effect of a . The four shapes in this figure are obtained by four consecutive iterations of the honeycomb scheme for a dodecahedral control shape.

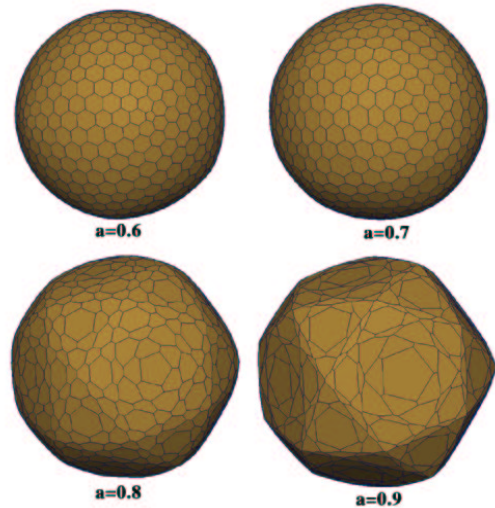


Figure 4. Effect of the parameter a .

Note that the value of a controls the speed of convergence. We suggest making a a decreasing function of N ,

i.e, for higher-sided faces (larger values of N) the algorithm must converge faster. In all the examples in this paper except Figure 4, we use the following function for a

$$a = 0.45 + \frac{1.25}{N}.$$

It is easy to verify that in this scheme, each coefficient $a_{n,m}$ is always greater than zero if $1 > a > 0$, and the coefficients do add up to 1. By using Fourier analysis, similar to Doo-Sabin’s approach, we can show that the new scheme also has three distinct but complex eigenvalues². The complex part comes from the phase shift introduced by the term $v_m + v_{(m+1) \pmod{N}}$. Absolute values of the other eigenvectors will be smaller than one if $a < 1$, regardless of the value of N . Similar to Doo-Sabin’s scheme, only one eigenvector corresponds to the eigenvalue 1, two eigenvectors correspond to the second largest (in absolute value) eigenvalue and the rest of the eigenvectors correspond to the smallest eigenvalue. Because of this property our scheme provides tangent plane continuity.

4. Remeshing Algorithm for Regular Honeycomb Subdivision

In this section, we present a honeycomb remeshing algorithm that only uses the operators provided by Doubly Linked Face List (DLFL) structure [6, 1, 2]. Since DLFL structure and its operators are topological robust, our algorithm always guarantees topological consistency of 2-manifold meshes.

In order to implement operations such as “undo”, it is better to preserve the previous mesh \mathcal{M}_n . The remeshing algorithm presented in this section creates \mathcal{M}_{n+1} while preserving the previous mesh \mathcal{M}_n .

Two fundamental topology changing operators of DLFL and one high-level operator are sufficient to develop a honeycomb remeshing algorithm:

1. CREATEVERTEX(v) creates a 2-manifold surface with one vertex v and one face f which we call a *point sphere* as shown in Figure 5. This operator is the same as the Euler operator $MVFS$ [14] and effectively adds a new surface component to the current 2-manifold. The CREATEVERTEX operator is essential in the initial stage of creation of new mesh and creates a new surface component in the given 2-manifold. In particular, it is necessary when a new surface component is created in an empty manifold.

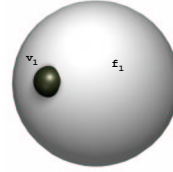


Figure 5. A point-sphere that consists of only one vertex and one face.

2. INSERTEDGE(c_1, c_2, e) inserts a new edge e to the mesh structure between two corners c_1 and c_2 (a *corner* is a subsequence of a face boundary walk consisting of two consecutive edges plus the vertex between them).

In general, if INSERTEDGE inserts an edge between two corners of the same face, the new edge divides the face into two faces without changing the topology. On the other hand, if INSERTEDGE inserts an edge between corners of two different faces (this includes the situation in which an endpoint or both endpoints of the new edge correspond to point-spheres), the new edge merges the two faces into one and changes the topology of the 2-manifold. This situation is shown in Figure 6.

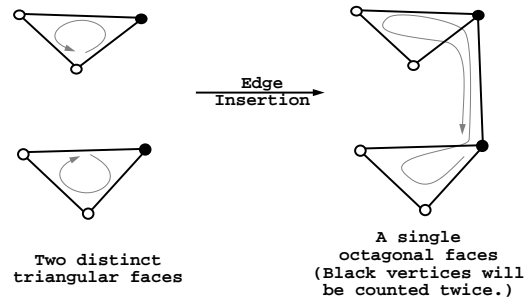


Figure 6. Inserting an edge between two different faces merges the two faces.

For the development of our honeycomb remeshing algorithm we also use one high-level operator that simplifies the algorithm.

3. CREATEFACEMANIFOLD(v_0, v_1, \dots, v_{N-1}). creates a two sided face (a manifold surface). This high-level operator can be implemented efficiently with the two fundamental operations given above by the following procedure.

²We want to point out that it is possible to derive coefficients that give real eigenvalues. In fact, we tested various sets of coefficients that gave real eigenvalues. However, the visual quality of the shapes turned out better with complex eigenvalues. Based on this information, we deduce that it is better to use complex eigenvalues because of the rotation inherent in the scheme.

- (a) for $i = 0$ to $N - 1$ do
 CREATEVERTEX(v_i);
- (b) for $i = 0$ to $N - 1$ do
 INSERTEDGE($v_i, v_{(i+1) \pmod N}, e$).

Remark 1. Each of these vertices before insert edge operation are smaller than valence 2. Therefore, we do not have to specify the corners.

Based on these three operators, the algorithm proceeds as follows for a given mesh \mathcal{M}_n under the DLFL structure,

- **Step 1:** for each face $f = \{v_0, v_1, \dots, v_{N-1}\}$ in the mesh \mathcal{M}_n
 - 1.1. compute the positions of the new vertices $v'_0, v'_1, \dots, v'_{N-1}$ using formulas (1), and (2) (see Figure 2.B);
 - 1.2. CREATEFACEMANIFOLD($v'_0, v'_1, \dots, v'_{N-1}$) (see Figure 2.C).

Remark 2. Step 1 creates two new faces $f' = (v'_0, v'_1, \dots, v'_{N-1})$ and $f'' = (v'_{N-1}, v'_{N-2}, \dots, v'_1, v'_0)$ for each old face $f = (v_0, v_1, \dots, v_{N-1})$ (see Figure 2.C). We call f' and f'' “the front face” and “the back face” respectively. Note that each new vertex v'_i is a vertex of both front and back faces, so we can talk about the “front corner” and “back corner” at vertex v'_i . More specifically, the front corner of v'_i is the vertex-face pair $\{v'_i, f'\}$, while the back corner at v'_i is the vertex-face pair $\{v'_i, f''\}$.

- **Step 2:** For each old edge $e = \{u, w\}$ the two sides of e (half edges [14]) induce two pairs of new vertices $\{v', v''\}$. Insert an edge between the back corners of the vertices v' and v'' (see Figure 2.D).
- **Step 3:** Do not draw (or delete) \mathcal{M}_n (see Figure 2.E).

Let \mathcal{M}_{n+1} denote the new mesh structure created by this algorithm. Although during this algorithm, insert edge operations change the topology in each step, it is straightforward to show \mathcal{M}_{n+1} has exactly the same topology as \mathcal{M}_n . Since the algorithm creates exactly the same mesh structure as in Figure 2, \mathcal{M}_{n+1} is the desired mesh.

5. Implementation and Results

The remeshing algorithm explained in the previous section has been implemented in a C++ program. We applied the honeycomb algorithm to the various control shapes. Figure 7 shows examples of applying our honeycomb scheme. The images at the left in Figure 7 show the control meshes and images at the right show the mesh after four iterations of our honeycomb algorithm. It is interesting to note that the honeycomb algorithm creates a very natural

looking mesh structure. It therefore can be used after other subdivision schemes to create natural looking meshes.

The honeycomb algorithm can create lateral artifacts [16]. Figure 8.(B) shows such an artifact. Although the hexagonal prism control mesh shown in Figure 8.(A) is a convex shape, we see smooth but periodic lumps and bumps across the height as shown in Figure 8.(B). These periodic bumps are created during the first iteration hexagonalization and smoothed after one more iteration. We call this periodic bumping “*Bricklayers’ Problem*” since a straight boundary cannot be obtained with hexagonal tiles. Bricklayers solve this problem quite simply: they cut hexagonal tiles along the borders. To implement their solution: (1) convert the faces of the control mesh into hexagons, (2) cut hexagonal faces at the boundaries as bricklayers. As shown in Figure 8.(D), when we use a control shape shown in Figure 8.(C) periodic bumps disappear.

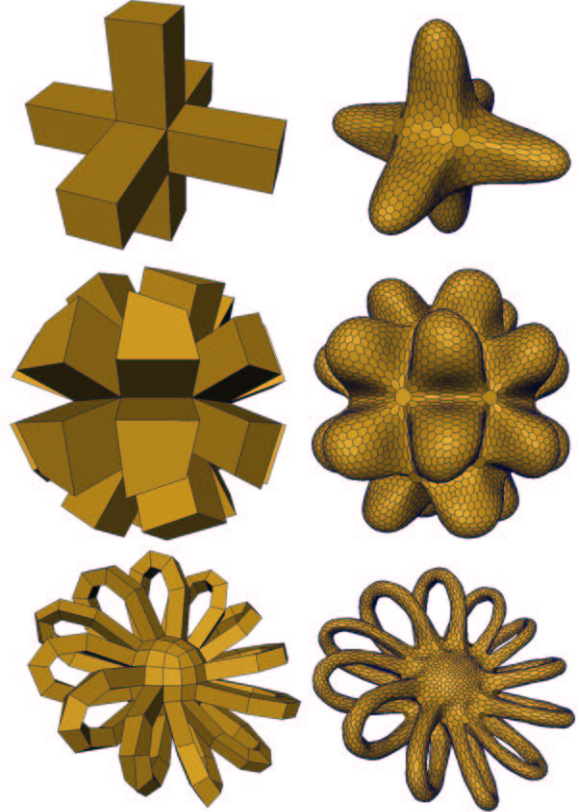


Figure 7. Examples of honeycomb subdivision. The images at the left show the control meshes and images at the right show the mesh structures after four iterations of our honeycomb algorithm.

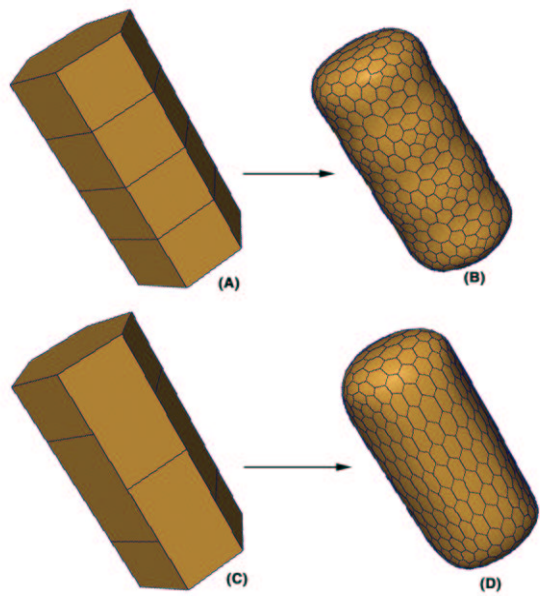


Figure 8. Lateral artifacts and bricklayer's solution.

6. Discussion, Conclusion and Future Work

In this paper, we introduced a new subdivision algorithm. The algorithm provides a tension parameter to control the shape of subdivided surface. We implemented a honeycomb algorithm, over the DLFL structure. We observed that honeycomb schemes can create natural looking mesh structures.

The main problem with honeycomb schemes are the lateral artifacts. Our current solution requires user intervention which may not be desirable for many applications. We expect that this problem can be solved with non-stationary planarization schemes. We are currently working on the development of such a scheme.

Another problem we face is that it is very difficult to identify the precision set for any given honeycomb scheme. Unfortunately, we still do not have a promising approach to solve this problem.

7. Acknowledgments

We are very grateful to the Malcolm Sabin who gave us encouragement when we first identified the honeycomb remeshing scheme. We are thankful to Gary Greenfield for his helpful suggestions. This work was partially funded by the Research Council of College of Architecture and the Interdisciplinary Program of Texas A&M University .

References

- [1] E. Akleman and J. Chen, "Guaranteeing the 2-Manifold Property for meshes with Doubly Linked Face List", *International Journal of Shape Modeling* Volume 5, No 2, pp. 149-177.
- [2] E. Akleman, J. Chen, and V. Srinivasan, "A New Paradigm for Changing Topology During Subdivision Modeling," *Pacific Graphics 2000*, October 2000, pp. 192-201.
- [3] E. Akleman, J. Chen, F. Eryoldas and V. Srinivasan, "Handle and Hole Improvement with a New Corner Cutting Scheme with Tension," *Shape Modeling 2001*, May 2001, pp. 183-192.
- [4] R. H. Bartels, J. C. Beatty, and B. A. Barsky, *An Introduction to Splines for use in Computer Graphics and Geometric Modeling*, Morgan Kaufmann Publishers, Los Altos, CA, 1987.
- [5] E. Catmull and J. Clark, "Recursively Generated B-spline Surfaces on Arbitrary Topological Meshes", *Computer Aided Design*, No. 10, September 1978, pp. 350-355.
- [6] J. Chen, "Algorithmic Graph Embeddings", *Theoretical Computer Science*, No. 181, 1997, pp. 247-266.
- [7] T. DeRose, M. Kass and T. Truong, "Subdivision Surfaces in Character Animation", *Computer Graphics*, No. 32, August 1998, pp. 85-94.
- [8] D. Doo and M. Sabin, "Behavior of Recursive Subdivision Surfaces Near Extraordinary Points", *Computer Aided Design*, No. 10, September 1978, pp. 356-360.
- [9] M. Halstead, M. Kass, and T. DeRose, "Efficient, fair interpolation using Catmull-Clark surfaces", *Computer Graphics*, No. 27, August 1993, pp. 35-44.
- [10] Kobbelt L., " $\sqrt{3}$ -Subdivision", *Computer Graphics*, No. 34, August 2000, pp. 103-112.
- [11] C. Loop, "Smooth Subdivision Surfaces Based on Triangles", Master's Thesis, Department of Mathematics, University of Utah, 1987.
- [12] C. Loop and T. DeRose, "Generalized B-spline surfaces with arbitrary topology", *Computer Graphics*, No. 24, August 1991, pp. 101-165.
- [13] C. Loop, "Smooth spline surfaces over irregular meshes", *Computer Graphics*, No. 28, August 1994, pp. 303-310.

- [14] M. Mantyla, *An Introduction to Solid Modeling*, Computer Science Press, Rockville, Ma., 1988.
- [15] B. Mandelbrot, *The Fractal Geometry of Nature*, W. H. Freeman and Co., New York, 1980.
- [16] M. Sabin, "Subdivision: Tutorial Notes", *Shape Modeling International 2001, Tutorial*, May 2000.
- [17] I. Stewart, *Game, Set and Math: Enigmas and Conundrums*, Penguin Books, London, 1991.
- [18] T. W. Sederberg, D. Sewell, and M. Sabin, "Non-Uniform recursive Subdivision Surfaces", *Computer Graphics*, No. 32, August 1998, pp. 387-394.
- [19] R. Williams, *The Geometrical Foundation of Natural Structures*, Dover Publications, Inc., 1972.
- [20] D. Zorin and P. Schröder, co-editors, *Subdivision for Modeling and Animation*, ACM SIGGRAPH'99 Course Notes no. 37, August, 1999.
- [21] D. Zorin and P. Schröder, editor, *Subdivision for Modeling and Animation*, ACM SIGGRAPH'2000 Course Notes no. 23, July, 2000.