ACM Symposium on Solid Modeling and Applications (2004)
G. Elber, N. Patrikalakis, P. Brunet (Editors)

# Connected & Manifold Sierpinsky Polyhedra

Vinod Srinivasan and Ergun Akleman[†]

Visualization Sciences Program,
Department of Architecture,
Texas A&M University

**Abstract**

*In this paper, we present a subdivision-inspired scheme to construct generalized Sierpinski polyhedron. Unlike usual Sierpinski polyhedra construction schemes, which create either an infinite set of disconnected tetrahedra or a non-manifold polyhedron, our robust construction scheme creates one connected and manifold polyhedron. Moreover, unlike the original schemes, this new scheme can be applied to any manifold polyhedral mesh and based on the shape of this initial polyhedra a large variety of Sierpinski polyhedra can be obtained.Our basic scheme can be viewed as applying simplest subdivision scheme [23] to an input polyhedron, but retaining old vertices. The porous structure is then obtained by removing the refined facets of the simplest subdivision.*

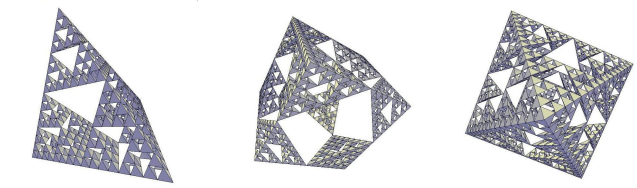
## 1. Motivation

Fractal geometry has emerged as one of the major mathematical approaches for designing unusual 3D shapes during the last two decades. Examples of such shapes introduced by fractal geometry include the Sierpinski tetrahedron, the Menger sponge, the Mandelbrot set and Julia sets [20].

Fractal geometry shapes are artistically intriguing and aesthetically pleasing [20]. Moreover, they provide unique challenges for the development of robust and computationally efficient shape construction approaches. Most shape construction algorithms for fractal geometry are given by a set of rules that are applied to an initial shape. However, the limit shapes are often independent of initial shapes and the algorithms are geometric in nature and hard to generalize.

Subdivision schemes provide a fresh alternative to fractal construction algorithms. They are conceptually similar to fractal constructions, i.e., they are also given by a set of rules that are applied to an initial shape. However, the subdivision schemes have three advantages: (1) their underlying rules (remeshing schemes) are mesh topological in nature, (2) the rules can simply be applied to any manifold polygonal mesh, (3) the limit shapes depend on initial shapes.

In this paper, we present a computationally efficient, robust and simple to implement subdivision scheme that allows construction of connected & manifold polyhedra that have the Sierpinski property. Two shapes created by using our approach are shown in Figure 1. As seen in this figure, our scheme is dependent on the initial shapes, i.e., different initial shapes give different (but similar looking) limit shapes. Moreover, some of these shapes cannot be constructed by geometrically combining affine copies of a Sierpinski tetrahedron. For instance, although the shape in Figure 1(B) can be assembled from affine copies of (A), (C) is topologically different from the Sierpinski polyhedron in (A) and cannot be created by affine copies of (A).



(A) (B) (C)

**Figure 1:** *Two shapes created by using our approach. Initial shapes are (A) a tetrahedron (B) a cube and (C) an octahedron.*

## 2. Introduction

A large class of fractal geometrical shapes, including the Sierpinski tetrahedron are self-similar. Such shapes can be constructed with a simple procedure that exploits their self-similarity property (introduced by Barnsley [7]). The most common approach is to repetitively take the union of transformed (e.g. scaled, rotated, translated and mirrored) copies of an initial shape. For instance, if a self-similar Fractal shape can be given as

$$S = U_{k=0}^{K} A_k S$$

[†] Corresponding Author: Address: Visualization Laboratory, 216 Langford Center, College Station, Texas 77843-3137. email: ergun@viz.tamu.edu. phone: +(979) 845-6599. fax: +(979) 845-4491.

where $A_k$ is a $4 \times 4$ transformation matrix in homogenous coordinate system and $U$ is the union operator, then, an algorithm to construct such a shape involves creating a series of shapes starting from an initial shape $S_0$ as

$$S_n \;=\; U_{k=0}^{K}\; A_k \;S_{n-1}$$

In practice, it is possible to ignore the union operation since using disconnected copies will visually give the same results. As a result, the number of the copies increases $K$ times in each iteration. Since the number of copies increases very quickly, after a few iterations a good approximation of $S$ can be obtained. Because of its simplicity, this approach is widely used to create fractal shapes. Another important property of this approach is that it is dimension independent, i.e. the same conceptual algorithm can be used both for 2D and 3D shape construction. Moreover, this algorithm is independent of the way the shape is represented, e.g, $S_0$ can be a set of points, an implicit surface, a polygonal surface, or a NURBS surface. Regardless of the type of the initial shape, the same algorithm can be used.

One of the problems with this shape construction approach is that the algorithms are specific to the target (limit) shapes. Each algorithm approaches its target shape regardless of the shape of the initial object. These algorithms do not allow construction of different target shapes from different initial shapes.

Fortunately, this approach is not the only method for constructing fractal shapes. These alternative approaches are usually not dimension independent and are hard to implement in 3D, so they have not been widely used in 3D applications. However, several of these approaches allow construction of a variety of fractal shapes from different initial shapes. A notable example is one of Mandelbrot's alternative Sierpinski triangle constructions that relies upon "*cutting out 'tremas'*" as defined by Mandelbrot [20].

### 2.1. Generalization of Mandelbrot's Alternative Sierpinski Triangle Construction

We observe that an attractive property of the above construction is that the initial shape does not have to be a uniform triangle (Mandelbrot did not explicitly mention it [20]). The initial shape can be a convex polygon by simply restating the construction algorithm as *from each convex polygon cut a convex polygon that is created by connecting the midpoints of each edge* as shown in Figure 2. After first application of this algorithm, all polygons become triangles. The algorithm also works for non-convex polygons, however, some parts of the initial polygon can be removed by cut operation as shown in Figure 3. If we interpret the "cut" operation as an "exclusive-or" operation instead of a set-difference, we can safely apply this construction to even non-convex shapes as shown in Figure 4.

### 2.2. Extension to 3D

It is hard to extend this algorithm to three dimensions using set operations. To construct a generalized Sierpinski polyhedron, we need to take a set-difference (or ex-or) of the initial polyhedron with a polyhedron that is constructed by connecting midpoints of each edge in the original polyhedron. There exist two problems:

1. Unlike the union operation, which can be visually implied without any implementation as such (we simply have to render all the objects), the set difference operation needs to be implemented. In this particular case set difference is particularly hard to implement since it creates non-manifold shapes [18].
2. Construction of a polyhedron by connecting the midpoints of each edge of the initial polyhedron can also be hard in solid modeling. In the case of a tetrahedron, the problem is easy since the shape that is constructed is an octahedron; i.e. the faces are triangular and therefore planar. But for most cases, the faces may not be triangular and hence may not be planar, complicating the set-difference procedure even further.

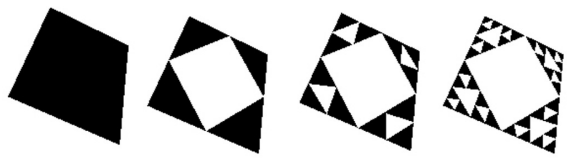


**Figure 2:** *Generalization of Mandelbrot's alternative Sierpinski triangle construction to convex polygons.*

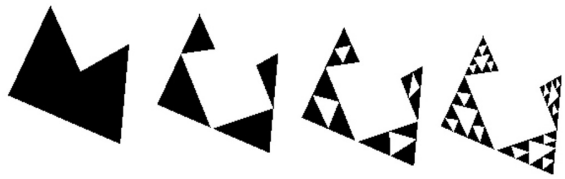


**Figure 3:** *Mandelbrot's alternative Sierpinski triangle construction for non-convex polygons.*

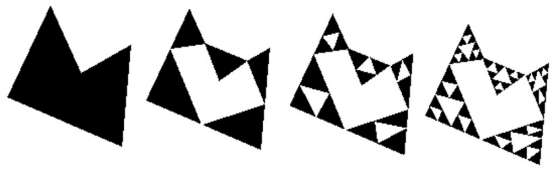


**Figure 4:** *Mandelbrot's alternative Sierpinski triangle construction for non-convex polygons by using "Exclusive-Or" instead of "set-difference".*

In this paper, we present an algorithm that provides a 3D version of the generalized Sierpinski triangle construction. Our 3D construction scheme is very similar to subdivision schemes [12, 10, 14, 23, 13, 25, 21], although we do not construct a smooth shape, .

- Like every subdivision scheme, our algorithm is based on a simple remeshing scheme.
- Similar to subdivision schemes, our algorithm can be applied to any polygonal manifold mesh
- Subdivision schemes create locally regular regions that approach parametric surfaces such as cubic B-Spline surfaces. Our algorithm also creates regular regions that approach Sierpinski tetrahedra.

- Similar to subdivision schemes, our scheme has extraordinary points. Initial valence-$n$ vertices continue to exist as a part of $n$-sided pyramids. The newly created pyramids are all 3-sided, i.e., tetrahedral.

However, unlike subdivision schemes, we change the topology of the initial mesh. Unlike popular algorithms for self-similar fractals, we do not create distinct surfaces. Instead each connected component in our initial manifold mesh remains connected. After the first iteration, each iteration increases the genus of the shape four times.

Our approach is based on topological mesh modeling, in which we guarantee topological manifold property of the meshes, but we do not keep any geometric information about the shape except the positions of the vertices. The faces and edges can have any shape, but, since we do not have to deal with their shapes, our algorithms become simple and allow us to provide real-time interaction. Since we largely ignore the geometry of the shapes, we do not have to deal with hard problems such as self-interactions or set operations. Moreover, we can represent non-manifold looking shapes with manifold meshes. This particular problem and our solution illustrate that some hard solid modeling problems can have simple solutions using topological mesh modeling.

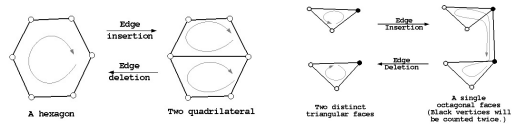
## 3. Topological Mesh Modeling

Akleman and Chen introduced Topological Mesh Modeling along with the Doubly Linked Face List (DLFL) data structure [2, 11, 1]. Their INSERTEDGE and DELETEEDGE [2] operators can effectively change the topology of a manifold mesh by inserting and deleting handles. They can be used effectively to implement subdivision schemes and allow topology change during subdivision modeling [3, 4]. Akleman, Chen and Srinivasan have recently developed a user interface [5] and theoretically shown [6, 11] that all and only orientable 2-manifold structures can be created using the two simplest Euler operations MVFS (make a surface with a single vertex and a single face) and KVFS (inverse of MVFS) [19] along with INSERTEDGE and DELETEEDGE. Moreover, these four operators can be efficiently implemented [11] on almost every mesh data structure including winged-edge, [9], half-edge [19] and quad-edge[17]. Using only these four operators, software development for mesh modeling with a topologically guaranteed orientable manifold property can be greatly simplified.

For the algorithm presented in this paper, we only need two operators: SUBDIVIDEEDGE and INSERTEDGE. The former, the SUBDIVIDEEDGE operator, is not a part of the minimal operator set, but it can be implemented by minimal operators. It simply subdivides a given edge into two smaller edges, inserting a new vertex at the midpoint. The later, INSERTEDGE, is a part of the minimal operator set and needs a detailed explanation since it is crucial for our algorithm.

INSERTEDGE$(c_1, c_2, e)$ inserts a new edge $e$ to the mesh structure between two corners $c_1$ and $c_2$. Formally, a *corner* is a subsequence of a face boundary walk consisting of two consecutive edges plus the vertex between them. But, we can consider a corner as a {vertex, face} pair or as a sequence of three vertices belonging to a face. For example if $f = \{v_1, v_2, v_3, v_4\}$ is a face, $v_1, v_2, v_3$ is a corner referring to the vertex $v_2$ in face $f$ and $\{v_3, v_4, v_1\}$ is the corner referring to the vertex $v_4$ in face $f$.

If INSERTEDGE inserts an edge between two corners of the same face, the new edge divides the face into two faces, as shown in Figure 5(A). On the other hand, if INSERTEDGE inserts an edge between corners of two different faces, the new edge merges the two faces into one as shown in Figure 5(B). This operation changes the topology of the mesh. if the two faces belong to the same surface, INSERTEDGE increases the genus of the surface. If the two faces belong to two disconnnected surfaces, then INSERTEDGE connects these two surfaces.




(A) (B)

**Figure 5:** *Inserting an edge between two corners of the same face (A) divides it into two faces and deleting an edge between two faces merges the two faces into one. On the other hand, inserting an edge between two different faces (B) merges the two faces and deleting an edge with both sides in the same face splits the face into two.*

## 4. Sierpinski Subdivision Algorithm

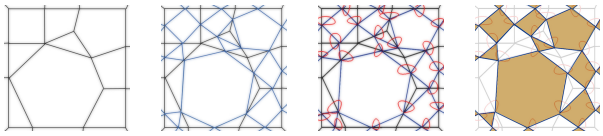
Let $V$ be the list of vertices, $E$ the list of edges and $F$ the list of faces in the original mesh. Moreover, let an edge whose end points (vertices) are the same be referred to as a *self-loop*. Then the algorithm is given as follows:

1. For every edge $e_i$ in $E$ that is not a self-loop, subdivide $e_i$ at its mid-point. Let $V_m$ be the list of all newly created edge mid-points.
2. For every vertex $v_i$ in $V$
   a. For every corner $c_i = \{a, v_i, b\}$ which points to $v_i$, insert an edge (shown as blue edges in Figure 6.B) between the corners $\{c, a, v_i\}$ and $\{v_i, b, d\}$, where $\{c, a, v_i, b, d\}$ forms a sub-sequence of vertices defining a face in the mesh.

   This will subdivide each face $f$ in the original mesh into as many triangles as the number of vertices in $f$ plus one central face (shown as yellow faces in Figure 6.D) which will have the same number of vertices as $f$.
3. For each vertex $v_i$ in $V_m$
   a. Find the corners $c_1$ and $c_2$ pointing to $v_i$ that are also part of one of the central faces created in the previous step.
   b. Insert an edge (shown as red edges in Figure 6.C) between $c_1$ and $c_2$.

After step 3 in the above process, all the central faces created in step 2 will no longer exist and we will have holes in their place. The restriction on edges not being self-loops in step 1 is necessary for recursive operation, since the edges inserted in step 3 above will all be self-loops.

The most important part of this algorithm is that the back faces (shown as white faces Figure 6.D) are automatically created. Each one of these faces has one self-loop on each one of its own vertices. Since (in practice) the self-loop edges have zero length, resulting faces look like they have $n$ number of sides instead of $2n$. For instance a triangle looking face is actually a hexagon as shown in

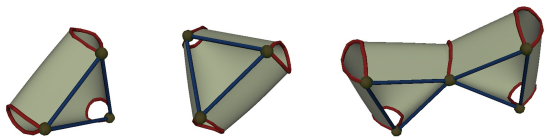


(A) (B) (C) (D)

**Figure 6:** *Visual presentation of the algorithm. (A) is the initial mesh, in (B), each edge is subdivided and midpoints are connected by inserting edges (shown as blue) to create new faces. In (C), an edge (shown as red) is inserted between two corners of each midpoint vertex. The yellow faces are automatically eliminated and new (white) faces are created.*

Figure 7(A) and (B). (Self-loops do not form separate faces, but they belong to the triangle looking backfaces. Holes can appear - becomes visible- only after all the self-loops are intserted.)

These self-loops act like boundaries that allows connection of each pyramid to another. The topological structure of one such connection between two hexagons (that look like triangles) is illustrated in Figure 7(C). Since these self-loops cover zero area, the resulting shapes look like non-manifolds, although they are manifolds.



(A) (B) (C)

**Figure 7:** *(A) and (B) are two topological renderings of a hexagonal face that looks like a triangle. (C) shows how to create a non-manifold looking manifold structure by appropriately connecting triangle-looking hexagonal faces in (A) and (B). .*

Note that this algorithm does not provide a set-difference or an exclusive-or operation. It gives acceptable results only for this particular problem. The algorithm is attractive mainly because of its computational efficiency and simplicity.

## 5. Implementation and Results

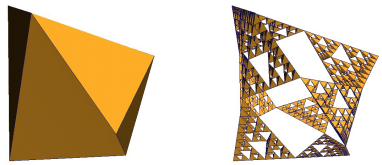
The algorithm above is implemented and included in our existing 2-manifold mesh modeling system [5] as an option. Our system is implemented in C++ and OpenGL, FLTK [15]. All the examples in this paper were created using this system.

Although we assume that the faces and edges can have any shape, in practice, their shapes are defined by OpenGL; i.e., edges are straight lines and faces are hardware rendered polygons. Therefore, although the algorithm can be applied to any manifold mesh, the quality of the results depends on properties of the initial manifold mesh.

In the discussion of the evaluation of the results, we ignore self-loops since they are only useful for simplification of the algorithm and otherwise invisible to viewers. For instance, we would call a face a triangle even if it is actually a hexagon with three self-loops.

To evaluate the results, we have classified vertices into 5 categories. Our classification of a vertex is based on the pyramid created by the straight edges (ignoring self-loops) that share that particular vertex: (1) *Convex vertex*. The tip of the pyramid is convex. (2) *Star vertex*. The tip of the pyramid is star. (3) *Concave vertex*. The tip of the pyramid is concave. (4) *Planar vertex*. The tip of the pyramid is flattened. (5) *Saddle vertex*. The tip of the pyramid is saddle shaped. Based on this classification, we identified the following cases.

- If the initial mesh consists of only 3-valence convex vertices such as in a dodecahedron or a cube [24], after the first iteration of the algorithm, the resulting mesh consists of only tetrahedral shapes. Since in a tetrahedral shape, each face is a triangle there is no problem in rendering. The faces of the convex polyhedron with 3-valence vertices do not have to be planar.as shown in Figure 8.

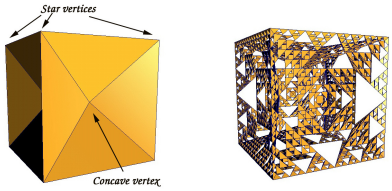


(A) (B)

**Figure 8:** *(A) An initial mesh (deformed cube) that consists of only 3-valence convex vertices with non-planar faces (B) the resulting shape after 4 iterations.*

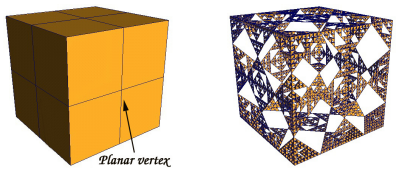
- If the initial mesh includes some non-3-valence convex vertices such as in an icosahedron or an octahedron [24], the resulting mesh always includes non-triangular faces. The planarity of these faces depends on the vertex positions in the initial mesh. This is not a grave problem since with each iteration such non-planar faces become more and more planar even if they were not planar initially.
- If the initial mesh includes some star vertices, the resulting mesh always includes star shaped faces. Even if these faces are planar, hardware rendering can sometimes create visual problems when star shapes are converted to triangles. However, again this is not an important issue since many interactive renderers do not create any problems. as shown in Figure 9.
- If the initial mesh includes some concave vertices, each one of these concave vertices creates a geometrically inverted pyramid, i.e. normal vectors points inside of the pyramid instead of outside. This problem is not easy to see and can be corrected easily by inverting normals.
- If the initial mesh includes some planar vertices, each one of these planar vertices creates a flattened pyramid. This problem can be visually annoying but cannot be corrected. Unfortunately, the number of flattened pyramids increases with each iteration. as shown in Figure 10. Therefore, it is better to avoid planar vertices.
- If the initial mesh includes some saddle vertices, each one of these saddle vertices creates a self-intersecting pyramid. This problem can also be be visually annoying and cannot be corrected. However, the number of self-intersecting pyramids stays the same in each iteration and in every iteration they become smaller and less annoying.

One of the advantages of our approach is that our algorithm




(A) (B)

**Figure 9:** *(A) An initial mesh that includes both star and concave vertices (B) the resulting shape after 4 iterations.*




(A) (B)

**Figure 10:** *(A) An initial mesh that includes some planar vertices (B) the resulting shape after 4 iterations.*

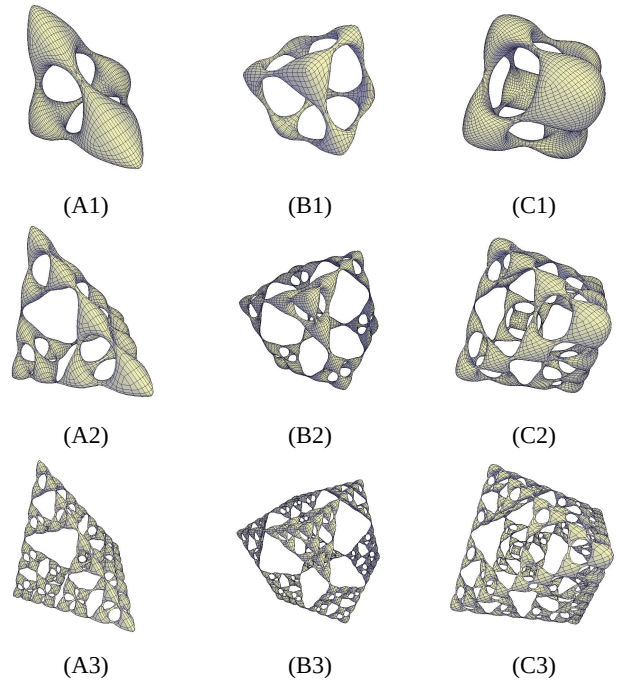
produces connected and manifold polyhedra. If we smooth these polyhedra by applying Doo-Sabin or Catmull-Clark [12, 10], the smoothed shape remains connnected as shown in Figure 11. We have used Doo-Sabin scheme since it is particularly useful for smoothing such non-manifold looking manifold meshes (see [3, 4] for detailed discussions). The Figure 12 shows examples of other subdivision schemes that can effectively smooth these Sierpinsky polyhedra.

## 6. Conclusion and Future Work

In this paper, we have presented a new subdivision scheme to construct very high genus polyhedra that have Sierpinski property. Our new scheme is robust, computationally efficient and simple. The scheme can be applied to any manifold polyhedral mesh. A large variety of Sierpinski polyhedra can be obtained based on the shape of the initial polyhedra. Although, our underlying data structure is DLFL, as shown in [11] the algorithm presented in this paper can be implemented by using any common data structure such as winged-edge,[9], half-edge [19] or quad-edge[17]. This approach illustrates that topological mesh modeling can provide simple and efficient solutions to some hard solid modeling problems.

## 7. Acknowledgments

We thank anonymous reviewers for their helpful comments. Their comments helped to improve the paper.

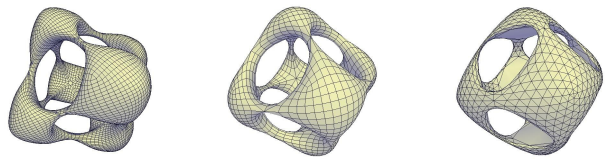


(A1) (B1) (C1)

(A2) (B2) (C2)

(A3) (B3) (C3)

**Figure 11:** *Examples of smoothed Sierpinsky polyhedra. Column (A) is tetrahedron, (B) is cube and (C) is octahedron. First, second and third rows shows Doo-Sabin smoothing after one, two and three iterations of our Sierpinsky scheme.*



(A) (B) (C)

**Figure 12:** *An example of smoothing effects of subdivision schemes over a simple Sierpinsky octahedron. (A) Doo-Sabin (B) Catmull-Clark and (C) extended Loop (modified to include non-trinagles).*

## References

[1] Topological mesh modeling site: All papers are available online at www-viz.tamu.edu/faculty/ergun/topology.

[2] E. Akleman and J. Chen, Guaranteeing the 2-manifold property for meshes with doubly linked face list, *International Journal of Shape Modeling* (Special issue on International Conference on Shape Modeling and Applications 1999), Volume 5, No. 2, pp. 149-177, 1999.

[3] E. Akleman, J. Chen, and V. Srinivasan, A new paradigm for changing topology during subdivision modeling, *Proceedings 8th Pacific Conference on Computer Graphics and Applications (PG'2000)*, pp. 192-201, HongKong, China. Oct. 2000.

[4] E. Akleman, J. Chen, V. Srinivasan and F. Eryoldas, "A New Corner Cutting Scheme with Tension and Handle-Face Reconstruction", *International Journal of Shape Modeling*, (Special issue on International Conference on Shape Modeling and Applications 2001), Volume 7, No. 2, pp. 111-121, 2001.

[5] E. Akleman, J. Chen and V. Srinivasan, "An Interactive System for Robust Topological Modeling of Meshes", *Technical*

*Sketch, SIGGRAPH'2001*, Los Angeles, California, August 2001.

[6] E. Akleman, J. Chen and V. Srinivasan, A Minimal and Complete Set of Operators for the Development of Robust Manifold Mesh Modelers, *Graphical Models Journal*, (Special issue on International Conference on Shape Modeling and Applications 2002) 65, (2003) pp. 286-304.

[7] M. Barnsley, *Fractals Everywhere*, Academic Press, Inc. San Diego Ca. 1988.

[8] 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.

[9] B. J. Baumgart, "Winged-edge polyhedron representation", Technical Report CS-320, Stanford University, 1972.

[10] E. Catmull and J. Clark, "Recursively Generated B-spline Surfaces on Arbitrary Topological Meshes", *Computer Aided Design*, 10, (September 1978) 350-355.

[11] J. Chen and E. Akleman, Topologically robust mesh modeling: concepts, structures and operations, Submitted to a journal. Technical report is available at www-viz.tamu.edu/faculty/ergun/topology.

[12] D. Doo and M. Sabin, "Behavior of Recursive Subdivision Surfaces Near Extraordinary Points", *Computer Aided Design*, 10, (September 1978) 356-360.

[13] Kobbelt L., "SquareRoot-3-Subdivision", *Computer Graphics*, No. 34, August 2000, pp. 103-112.

[14] C. Loop, "Smooth Subdivision Surfaces Based on Triangles", Master's Thesis, Department of Mathematics, University of Utah, 1987.

[15] The Fast Light Toolkit Home Page: http://www.fltk.org/

[16] A. T. Fomenko and T. L. Kunii, *Topological Modeling for Visualization*, (Springer-Verlag, New York, 1997).

[17] L. Guibas, J. Stolfi, "Primitives for the manipulation of general subdivisions and computation of Voronoi diagrams", *ACM Transaction on Graphics*, 4, (1985) 74-123.

[18] C. M. Hoffmann, *Geometric & Solid Modeling, An Introduction*, (Morgan Kaufman Publishers, Inc., San Mateo, Ca., 1989).

[19] M. Mäntylä, *An Introduction to Solid Modeling*, Computer Science Press, Rockville, MA, 1988.

[20] B. Mandelbrot, *The Fractal Geometry of Nature*, W. H. Freeman and Co., New York, 1980.

[21] M. Sabin, "Subdivision: Tutorial Notes", *Shape Modeling International 2001, Tutorial*, May 2001.

[22] I. Stewart, *Game, Set and Math: Enigmas and Conundrums*, Penguin Books, London, 1991.

[23] J. Peters and U. Reif, "The Simplest Subdivision Scheme for Smoothing Polyhedra", *ACM Transactions on Graphics*, Vol. 16, No. 4, pp. 420-431, October 1997.

[24] R. Williams, *The Geometrical Foundation of Natural Structures*, Dover Publications, Inc., 1972.

[25] D. Zorin and P. Schröder, editor, *Subdivision for Modeling and Animation*, ACM SIGGRAPH'2000 Course Notes no. 23, July, 2000.

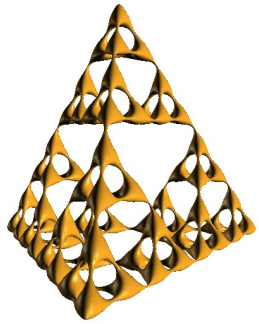


**Figure 13:** *A detailed example of smoothed Sierpinski polyhedra:. We have applied 3 iterations of our Sierpinski subdivision to a tetrahedron and then smoothed it using Doo-Sabin subdivision.*

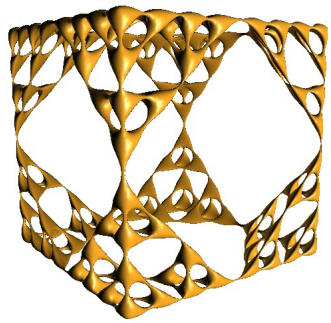


**Figure 14:** *A detailed example of smoothed Sierpinski polyhedra:. We have applied 3 iterations of our Sierpinski subdivision to a cube. then smoothed it using Doo-Sabin subdivision.*