MODELING HIGH-GENUS SURFACES

A Dissertation

by

VINOD SRINIVASAN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2004

Major Subject: Architecture

MODELING HIGH-GENUS SURFACES

A Dissertation

by

VINOD SRINIVASAN

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Approved as to style and content by:

Ergun Akleman (Chair of Committee)	
Donald H. House (Member)	Michael K. Lindell (Member)
Jianer Chen (Member)	Phillip J. Tabb (Head of Department)

May 2004

Major Subject: Architecture

ABSTRACT

Modeling High-Genus Surfaces. (May 2004)

Vinod Srinivasan, B.Tech., Indian Institute of Technology, Madras, India;

M.S., Texas A&M University

Chair of Advisory Committee: Dr. Ergun Akleman

The goal of this research is to develop new, interactive methods for creating very high-genus 2-manifold meshes. The various approaches investigated in this research can be categorized into two groups – interactive methods, where the user primarily controls the creation of the high-genus mesh, and automatic methods, where there is minimal user interaction and the program automatically creates the high-genus mesh.

In the interactive category, two different methods have been developed. The first allows the creation of multi-segment, curved handles between two different faces, which can belong to the same mesh or to geometrically distinct meshes. The second method, which is referred to as "rind modeling", provides for easy creation of surfaces resembling peeled and punctured rinds.

The automatic category also includes two different methods. The first one automates the process of creating generalized Sierpinski polyhedra, while the second one allows the creation of Menger sponge-type meshes.

Efficient and robust algorithms for these approaches and user-friendly tools for these algorithms have been developed and implemented.

To the pursuit of knowledge and peace

असतो मा सद्गमय ।
तमसो मा ज्योतिर्गमय ।
मृत्योर्मा अमृतंगमय ।
ॐ शान्तिः शान्तिः शान्तिः ॥

Lead us from untruth to truth,
from ignorance to enlightenment,
from death to immortality!
May there be peace in all three worlds!

ACKNOWLEDGMENTS

I would like to thank my parents and my family for supporting me through the long years of completing my Ph.D. I would also like to thank Dr. Seema Endley for providing the psychological boost that one needs every so often, especially when working on a doctoral degree.

I would like to thank Dr. Ergun Akleman for accepting me as his student and for his invaluable guidance and assistance throughout the duration of my research. I would also like to thank Dr. Donald House, Dr. Michael Lindell and Dr. Jianer Chen for serving on my committee and for their help and guidance with several aspects of my research.

I am grateful to the staff of the Visualization Lab at Texas A&M University for their valuable assistance with equipment and computers. I would also like to thank students in the Visualization Lab who have contributed to the development of the software and helped me improve it through their constructive feedback.

I would also like to thank the staff and my colleagues at the Ocean Drilling Program, especially Rakesh Mithal and David Fackler, for providing me with a means of support and for being extremely accomodative during the last few years of my Ph.D.

I will forever be grateful to Gary Hufford for introducing me to the wonderful concept of object-oriented analysis, design and development.

TABLE OF CONTENTS

CHAPTER Page

LIST OF FIGURES

FIGURE Page

FIGURE Page

FIGURE Page

CHAPTER I

MOTIVATION AND INTRODUCTION

Fig. 1. Nested elephant sculpture from India.

The inspiration for this research came from looking at models such as the elephant sculpture shown in Figure 1. Models such as these are carved from a single marble block. In this particular instance, the belly of the elephant is made into a shell. Holes are then punched in the shell. Through these holes a smaller elephant is then sculpted inside the shell. The exceptional skills of the artisan are evident from the intricate details in the models. Figure 2 shows some more examples of this kind of sculpture, all carved from soapstone. Wood sculptures of this kind are also common.

The journal model is *IEEE Transactions on Visualization and Computer Graphics*.

Fig. 2. Nested swan and rabbit sculptures.

The most prominent feature of these models, which sets them apart from other sculptures of similar objects, are the holes in the surface of the model. From a topological perspective, the holes make these models *high-genus* surfaces. A rigorous definition of topological terms such as *genus* is given in Chapter II. For now, the genus of an object can be correlated with the number of holes in the object – the higher the number of holes, the higher the genus.

Robust modeling of 2-manifold polygonal meshes with arbitrary topology (such as high-genus meshes) has always been a challenge in computer graphics. When we introduce the requirement for interactivity in the modeling process, the problem is complicated further, primarily because we are trying to work on three-dimensional objects through a two-dimensional interface.

In computer graphics, we are not only interested in being able to model objects, we would also like to produce high quality renderings of those models. In recent years, *subdivision surfaces* have gained prominence as a useful modeling and rendering tool in the computer graphics industry. One of the requirements of most subdivision schemes

is that the surface be a valid 2-manifold mesh [10, 12, 43]. Ensuring topological consistency during modeling has thus become more important.

Another important application in computer graphics is physically based modeling, which includes physical simulations of natural processes using computer models. In the real world, every object is an *orientable 2-manifold* surface, which in simple terms means that the object has a well defined interior and exterior. This is because it is physically impossible to have a thickness of zero. Thus manifold representations of real-world objects are also useful functional models, since they can be used in physical simulations without having to simulate a manifold surface programmatically.

I.1. High-genus objects from China

Fig. 3. Two views of a high-genus nested sculpture from China.

Figure 3 shows two views of a Chinese sculpture which contains sixteen nested balls. Each inner ball has been carved through the holes in the outer balls. An important feature of this model, which makes it different from the sculptures shown earlier, is that the inner balls are completely disconnected from the outer ball. Each

ball can freely rotate independent of the other balls. From a modeling perspective this can be easily achieved by simply combining scaled copies of the outer shell. That still leaves us with having to create the outer shell which is a high-genus surface.

I.2. High-genus objects from contemporary artists

The domain of artistic high-genus objects is not restricted to India and China. The famous artist M.C. Escher has also created several drawings of objects which have very high genus [13]. The *Möbius Strip* and *Cube with Ribbons* [8] are two of the more interesting examples. Several of sculptor George Hart's creations [23], which include sculptures made from materials such as oak, brass and acrylic, are also examples of high-genus objects. Sculptor Helaman Ferguson has also created several high-genus objects from a range of materials including marble, bronze and stone [15].

I.3. High-genus man-made objects

The examples of high-genus objects given above are all from art. However, high-genus objects are much more common in the real world. Numerous man-made objects have high genus. Buildings are high-genus objects when one considers doors and windows as holes in the building structure. Communication towers, perforated wood and stone screens, decorative window panes, bridges, gates and automobiles are some more examples of high-genus man-made objects. The ubiquitous computer monitor, without which there is not much to do in computer graphics, is also a high genus object, with the large number of holes in the casing for heat dissipation. Figure 4 shows some examples of such objects.

Fig. 4. Examples of man-made high-genus objects.

I.4. High-genus objects from mathematics

In the last two decades, fractal geometry has emerged as one of the major mathematical approaches for designing unusual 3D shapes. Examples of such shapes introduced by fractal geometry include the Sierpinski gasket, the Menger sponge, the Mandelbrot set and Julia sets [27].

Fractal geometry shapes are artistically intriguing and aesthetically pleasing [27]. Fractals have also been used to model naturally occurring objects such as snowflakes and clouds. The Menger Sponge has been used to model the porous structure of soil for simulation of various geophysical processes [32, 36].

The shapes from fractal geometry provide unique challenges for the development of robust and computationally efficient shape construction approaches. The simple set of rules that govern the construction of fractal shapes lend themselves to automatic modeling of such shapes. Of interest in this research is the ability to automatically create high-genus shapes.

I.5. Current modeling tools

In spite of the prevalence of high-genus objects all around, surprisingly, there aren't many software tools for efficient and interactive modeling of such objects. Although it is possible to create high-genus meshes using currently available three-dimensional modeling packages, the process is cumbersome and time consuming, especially for shapes like the ones shown earlier. Software such as Alias Wavefront's Maya do provide some topological operations [4], but the operations are not designed for high-genus modeling. Moreover, the available operations are not necessarily manifold preserving, and do not guarantee topological consistency, both of which are very important in three-dimensional modeling [1].

I.6. Organization of this document

Fig. 5. More examples of high-genus sculptures.

What started out as an exploration of tools to model objects such as the ones shown in Figure 5 has led to an extensive set of algorithms and tools which have ap-

plications beyond just high-genus modeling. Figuratively speaking, the ideas hatched from the eggs in the figure have matured and grown into an impressive collection of tools for mesh modeling.

Chapter II explains various concepts and terms used in this dissertation and talks about previous work in this area. It also talks about data structures and fundamental mesh modeling operators.

Chapter III gives details of the various *High-level Mesh Modeling Operators* that are made use of by the high-genus modeling tools.

The tools that have been developed in this research can be grouped into two categories – interactive tools and automatic tools. Chapter IV describes the first tool in the interactive category, namely, the creation of *Multi-Segment Curved Handles*. The second interactive tool, *Rind Modeling* is described in Chapter V.

The next two tools fall into the automatic category. Chapter VI presents an automated approach for creating *Generalized Sierpinski Polyhedra*, while Chapter VII presents a tool for automatic construction of *Generalized Menger Sponges*.

Finally, Chapter VIII summarizes the work done in this research and presents results and conclusions. Some ideas for future work are also given.

CHAPTER II

BACKGROUND

In this chapter, various concepts relevant to this research and terms used in this dissertation are explained. Previous work done in the area of high genus modeling is also mentioned. Topological mesh modeling operators, which are fundamental to developing high-genus modeling tools, are discussed. A minimal set of fundamental operators through which any topological operation can be performed is also presented.

II.1. Topological concepts

Topology primarily concerns itself with the *qualitative* characteristics of a geometrical object rather than its *quantitative* dimensions [22]. Topological mesh modeling operators are thus greatly simplified since geometric considerations become secondary. This in turn simplifies the development of the tools needed for this research, since they are primarily topological in nature. To get a better understanding of how the tools work, it is useful to have some knowledge of the topological concepts involved.

Fundamental to this research is the topological concept of a *2-manifold*. A *2-manifold* or a *2-dimensional manifold* is a topological space where every point has a neighborhood topologically equivalent to an open disk. In other words, the geometrical object resembles the plane locally [22].

A *closed surface* is a connected, closed, 2-manifold [22]. That is, it consists of a single piece and has no edges (boundaries). A closed surface is *orientable* if it does not contain a Möbius band [19, 22]. In simpler terms, an orientable surface is one which has a well defined interior and exterior. The Möbius strip is an example of a non-orientable surface – one can walk around and reach any point on the surface

without ever leaving the surface or passing through the surface. In contrast, such a traversal is not possible on a sphere or a torus, both of which are orientable surfaces.

All 2-manifolds in this research are assumed to be orientable unless explicitly stated otherwise. A 2-manifold in general consists of a number of surfaces, each of which is homeomorphic (topologically equivalent) to a sphere with zero or more handles. The number of handles on the sphere is called the *genus* of the surface. Equivalently, one could define the genus to be the number of holes in the surface. The *genus* of a 2-manifold is the sum of the genera of its component surfaces [19]. Thus a high-genus surface is one which has a large number of handles or holes.

II.2. Previous work in high-genus modeling

The creation of very high genus, smooth, 2-manifold surfaces has been an ongoing area of research interest in computer graphics and shape modeling [17]. Ferguson, Rockwood and Cox used Bezier patches to create high-genus 2-manifold surfaces [14]. Welch and Witkin used handles to design triangulated free-form surfaces [40]. Takahashi, Shinagawa and Kunii developed a feature-based approach to create smooth 2-manifold surfaces with high genus [35].

II.3. Data structures for mesh representation

Meshes are commonly used in computer graphics to represent objects [25]. Several data structures have been proposed to represent 2-manifold mesh structures. Some of these are "face-based" in which mesh faces are explicitly given in consistent and oriented directions [5, 29], while others are "edge-based" in which adjacency relationships around each edge are given [7, 9, 42, 38, 28, 20, 21, 26, 37]. Baumgart's winged-edge structure [7] is the most well known edge-based representation, based on

which several variants have been proposed, including Weiler's edge based structure [38], Mäntylä's half-edge structure [28] and Guibas and Stolfi's quad-edge structure [20].

Several of the above data structures, including Weiler's *radial-edge* structure [39], Karasick's *star-edge* structure [26] and Vanecek's *edge-based* data structure [37] can support a wide range of non-manifold surfaces. Mäntylä's *half-edge* representation [28] is one data structure that is designed to support manifold meshes. It is possible to make the internal representation of the objects valid orientable 2-manifold sturctures even when the corresponding geometric shapes appear to be non-manifold [24].

II.4. Mesh operators

Since this research is about modeling meshes, operators on meshes are of primary concern. Mäntylä made a systematic study of mesh modeling operators [28] and studied the Euler operators proposed by Baumgart [7]. Guibas and Stolfi also proposed a set of operations on 2-manifold structures [20].

An important consideration in topological mesh modeling is that the operations on 2-manifold structures be *manifold preserving*. That is, the operations should result in valid 2-manifold structures.

Akleman and Chen recently introduced a topologically robust mesh modeling approach [1] by adopting topological graph theory [11, 19] to computer graphics and shape modeling. Their 2-manifold mesh modeling scheme is based on a minimal set of manifold preserving operators [1] that are simpler, more intuitive and more user-friendly when compared to previously proposed schemes.

The minimal set of fundamental operators that have been identified are : CreateVertex, which inserts a new vertex into the mesh, DeleteVertex, which

removes an existing vertex from the mesh, InsertEdge, which inserts an edge between two existing corners of the mesh and DeleteEdge, which deletes an existing edge from the mesh [1]. This set of operators is more uniform in comparison to other previously proposed operator sets, both from the point of view of modeling systems and end users. A modeling system using these operators only needs to deal with the internal representation and does not have to worry about the topological integrity of the mesh. Using these operators does not require understanding the internal implementation and only requires identifying corners in the mesh structure, which are the operands of the above mentioned operators.

II.5. Fundamental operators

The mesh modeling approach introduced by Akleman and Chen [1] will be followed in this research. In their approach, they make use of the *Doubly Linked Face List* (DLFL), originally developed by Chen [11]. This is used as the underlying data structure for representing meshes in this research.

The DLFL structure consists of a list of *vertices*, *edges* and *faces*. *Vertex*, *edge* and *face* refer to the internal representations of a point in three-dimensional space, a line segment connecting two points and a sequence of points respectively. For brevity and simplicity we will not make an explicit distinction between the internal representation and the actual geometric entity unless required. Thus, *vertex* will be used to refer to both the geometric entity as well as the topological entity (the internal representation) except where the reference is not clear from the context of the usage.

To simplify operations on the mesh, an additional entity, namely a *corner*, is also introduced. A *corner* is a vertex-face pair, $c = \{v, f\}$, where v is one of the vertices in f. Formally, a corner is a subsequence of the face boundary walk. That

is, if $f = (v_0, v_1, \ldots, v_{N-1})$ is a face, $c_i = \{v_{i-1}, v_i, v_{i+1}\}$ is the corner referring to the vertex v_i in f. A corner is associated with only one face, but several corners can refer to the same vertex.

Internally, each face is represented as an ordered sequence of corners each of which contains a pointer back to the face. Every corner also has a pointer to the vertex it refers to and every vertex contains a list of corners which refer to it. An edge contains pointers to two corners, one for each end of the edge. For each end of the edge there are two possible choices for the corners. The edge stores the corners at which the edge *originates*[1] in the two faces which are on either side of the edge. Each corner in turn has a pointer to the edge which originates there.

Based on the minimal set of operators mentioned above, high-level operators can be developed, which will allow the user to easily and intuitively perform topological and homeomorphic operations on a given 2-manifold mesh structure [2]. Before exploring the various high-level operators that have been developed, we will take a more detailed look at the minimal set of fundamental operators.

1. $(v, f) = \textsc{CreateVertex}(p)$ creates a 2-manifold surface with one vertex v and one face f which will be referred to as a *point sphere*. The geometric coordinates of the vertex v are given by p which is a point in three-dimensional space. The operation is the same as the Euler operation $MVFS$ [28] and effectively adds a new surface component to the current 2-manifold. The CreateVertex operator is essential in the initial stage of the creation of a new mesh and creates a new surface component in the given 2-manifold. In particular, this operator is necessary when a new surface component is to be created in an empty manifold.

[1]Since each face is an ordered sequence of corners, we can talk of an edge as originating at a particular corner in a face.

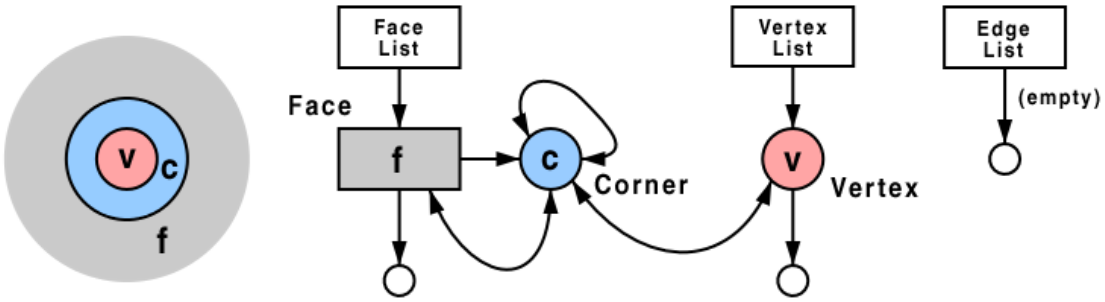


Fig. 6. Structure of a point sphere.

Figure 6 illustrates the structure of a point sphere. With respect to the formal definition of a corner given above, we can consider the preceeding and succeeding vertices to be the same as the lone vertex in the face. Thus if $c = \{v, f\}$ is the lone corner in the face, it can also be written as $c = \{v, v, v\}$.

2. DeleteVertex(v) is the complement of the CreateVertex operator. It removes a point sphere from the mesh structure. If v is not part of a point-sphere, the operator returns without making any changes to the mesh. The operation is the same as the Euler operation $KVFS$ [28] and effectively removes an existing surface component from the current 2-manifold. The DeleteVertex operator is essential for cleaning up the mesh structure to prevent unwanted visual artifacts from appearing.

3. e = InsertEdge(c_1, c_2) inserts a new edge e into the mesh structure between two corners c_1 and c_2 as shown in Figure 7.

   If InsertEdge inserts an edge between two corners of the same face, the new edge divides the face into two faces without changing 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

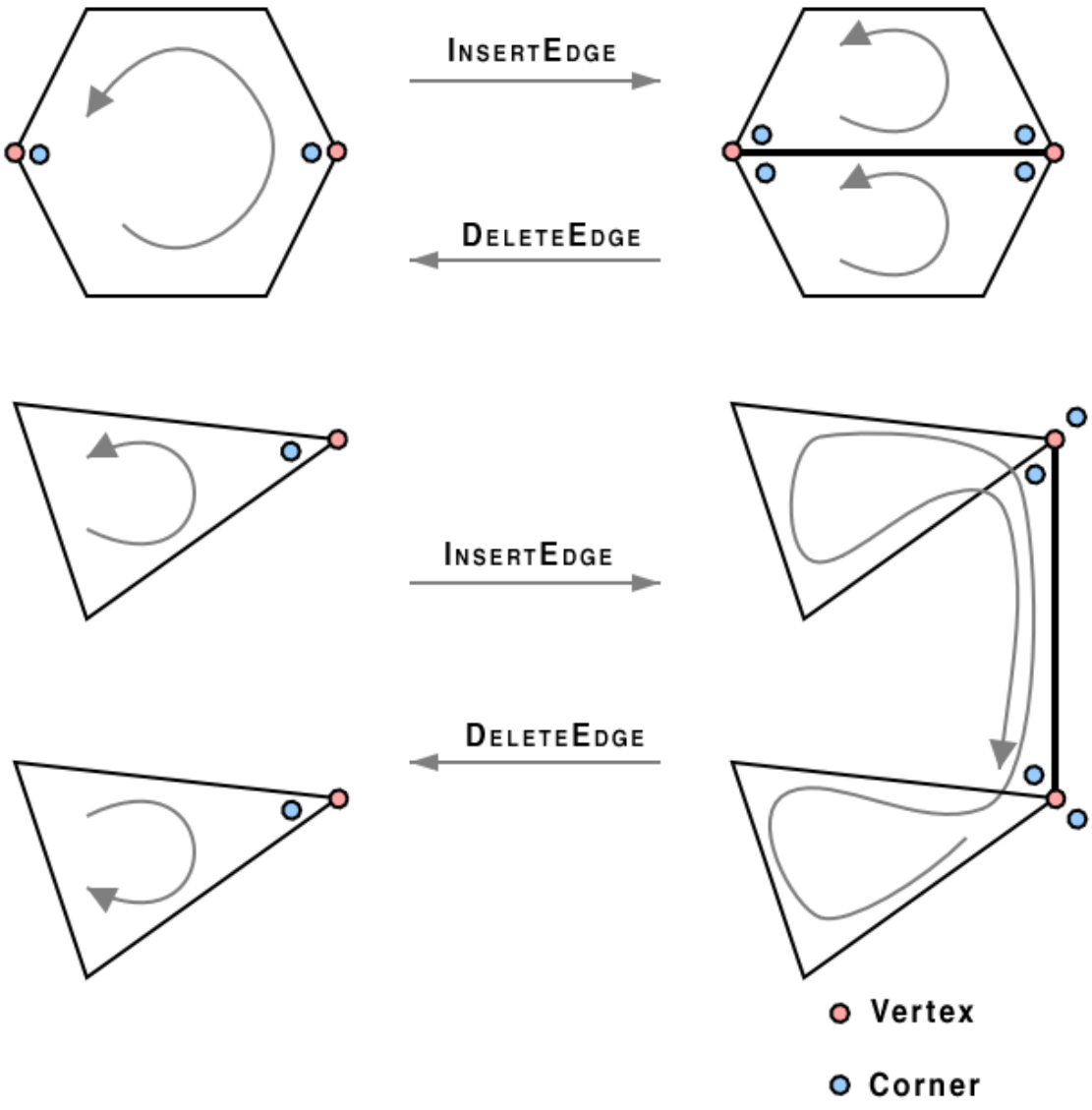


Fig. 7. InsertEdge and DeleteEdge operators.

edge correspond to point spheres), the new edge merges the two faces into one and changes the topology of the 2-manifold.

4. DeleteEdge(e) deletes the edge e from the mesh structure.

   This is the inverse of the InsertEdge operator. In general, if f_1 and f_2 are the faces on either side of the edge e, then deleting e combines f_1 and f_2 into a single face. But if f_1 and f_2 refer to the same face f (as will be the case if e is the result of an InsertEdge operation between corners of two different faces), then deleting e separates f into two faces, thereby changing the topology of the mesh. Figure 7 illustrates this operator.

The various high-level operators developed using these fundamental operators are described in the next chapter.

CHAPTER III

HIGH-LEVEL MESH MODELING OPERATORS

The fundamental operators introduced in the previous chapter form the core of the topological mesh modeling system. All operations on the mesh can be executed as a sequence of the four fundamental operators, CreateVertex, DeleteVertex, InsertEdge and DeleteEdge.

High-level operators combine a sequence of the fundamental operators, usually along with one or more non-topological operations (such as loops or mathematical calculations) into a single unit. They simplify the user interface since most operations on a mesh involve more than just application of the basic operators. They also simplify the development of more complex operations on a mesh such as those that will be introduced in later chapters.

III.1. DeleteEdgeWithCleanup(e)

As described earlier, the DeleteEdge operator removes an edge from the mesh. If the faces on either side of the edge are the same, then the operator splits the face into two. In the special case when one end of the edge is a valence-1 vertex[1], one of those faces (corresponding to the valence-1 vertex) becomes a point sphere. If both ends of the edge are valence-1 vertices, then deleting the edge creates two point-spheres. In either situation, the point-spheres can cause visual artifacts, especially when smoothing the mesh.

The point-spheres can be automatically cleaned up since all the necessary information is readily available. The DeleteEdgeWithCleanup operator does exactly

[1]The *valence* of a vertex is the number of edges incident to that vertex.

that. It deletes the specified edge and if there are any point-spheres created by the deletion, it cleans them up using the DeleteVertex operator. Figure 8 illustrates its operation. The red edges in the first image are to be deleted. Using only the DeleteEdge we get the mesh in the bottom right image in which a point sphere still remains. The DeleteEdgeWithCleanup operator performs the additional step of removing the point sphere to produce the mesh shown in the final image.

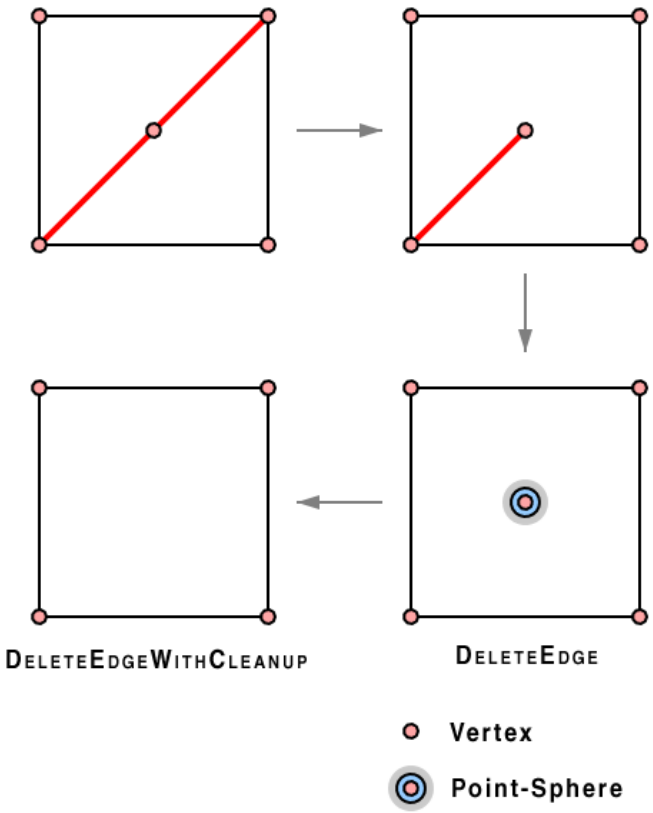


Fig. 8. The DeleteEdgeWithCleanup operator.

The algorithm for DeleteEdgeWithCleanup proceeds as follows:

1. Find the two faces on either side of the edge e, f_1 and f_2.
2. DeleteEdge(e).

3. If f_1 is a point-sphere

   (a) Find the vertex v_1 corresponding to the point sphere f_1.

   (b) DeleteVertex(v_1).

4. If $f_2 \neq f_1$ and f_2 is a point-sphere

   (a) Find the vertex v_2 corresponding to the point sphere f_2.

   (b) DeleteVertex(v_2).

III.2. $(\mathbf{e_1}, \mathbf{e_2})$ = SubDivideEdge(e)

The SubDivideEdge operator, as the name implies, subdivides the given edge into two equal halves. The operation introduces a new vertex at the middle of the original edge and increases the number of edges in the mesh by one. The topology of the mesh remains unchanged. This operator is very useful for remeshing the faces of a mesh. Figure 9 illustrates the working of this operator.

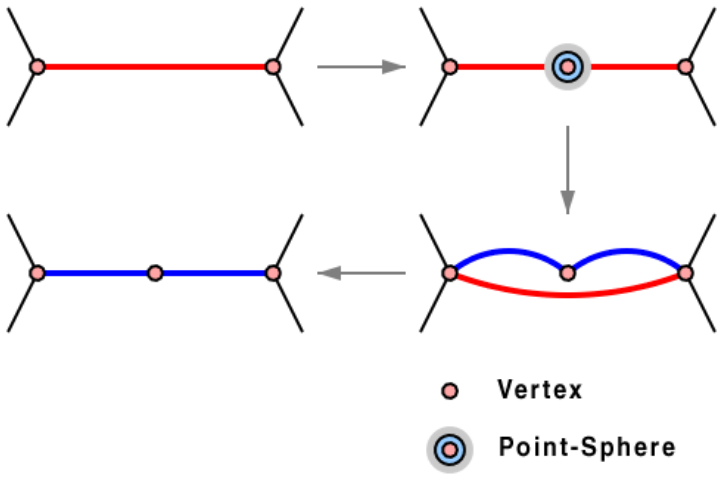


Fig. 9. The SubDivideEdge operator. In the third step, the original edge is shown curved for clarity.

The algorithm proceeds as follows:

1. Let v_1 and v_2 be the vertices connected by e. Compute the mid-point of the edge $p_m = \frac{v_1+v_2}{2}$.

2. $(v_m, f_m) = \textsc{CreateVertex}(p_m)$. This creates a point-sphere at p_m. Let $c_m = \{v_m, f_m\}$.

3. Let f be one of the faces adjacent to e. Let c_1 and c_2 be the corners in f corresponding to the vertices v_1 and v_2.

   (a) $e_1 = \textsc{InsertEdge}(c_1, c_m)$. This creates an edge between v_1 and the newly created vertex, v_m.

   (b) $e_2 = \textsc{InsertEdge}(c_2, c_m)$. This creates an edge between v_2 and v_m.

   After the first InsertEdge operation, there will still be only one corner referring to v_m, namely c_m. Thus, for the second InsertEdge operation, we can use the same corner.

4. $\textsc{DeleteEdge}(e)$. This removes the original edge from the mesh.

III.3. CreatePipe($\mathbf{c_1}, \mathbf{c_2}$)

The InsertEdge operator inserts an edge between two corners. In the situation when the corners belong to different faces, it connects the two faces with an infinitely thin handle or pipe, thereby increasing the genus of the surface by one. The CreatePipe operator is a natural extension to the InsertEdge operator, where every matching corner of two different faces is connected to produce a solid handle or pipe as shown in Figure 10. This operation still increases the genus of the surface only by one, but produces cleaner geometry which is better suited for subdivision and remeshing.

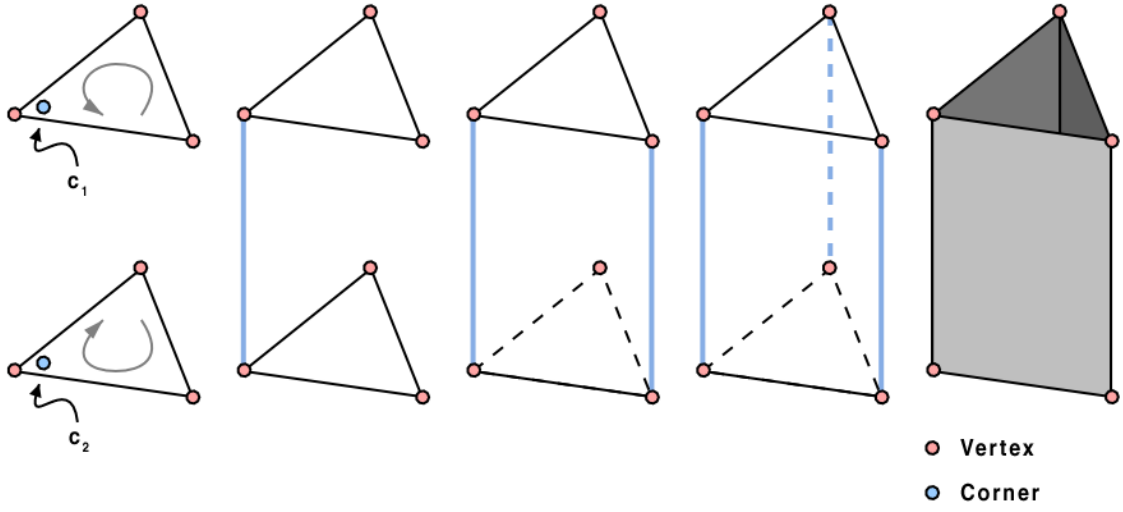


Fig. 10. The CreatePipe operator.

Formally, the CreatePipe operator connects the two faces which contain the corners c_1 and c_2 such that there is an edge between c_1 and c_2 and there is an edge between other matching corners (with reference to c_1 and c_2) of the two faces.

It is assumed that the two corners belong to topologically distinct faces with equal number of corners. The latter assumption is not really a restriction, since a pre-processing step can be used to make the face-valences[2] equal by sub-dividing the edges of the face with smaller number of corners [3].

III.4. $(\mathbf{f_f}, \mathbf{f_b}) = \mathbf{CreateFaceManifold}(\mathbf{p_0}, \mathbf{p_1}, \ldots, \mathbf{p_{N-1}})$

The CreateFaceManifold operator creates a manifold surface consisting of two faces which share the same vertices but are turned in opposite directions as shown in Figure 11. The vertex coordinates of the faces are given by points (p_i) specified as input. Of the two faces created, one will contain the input points in the given order,

[2]The term "face-valence" is used to refer to the number of corners in a face

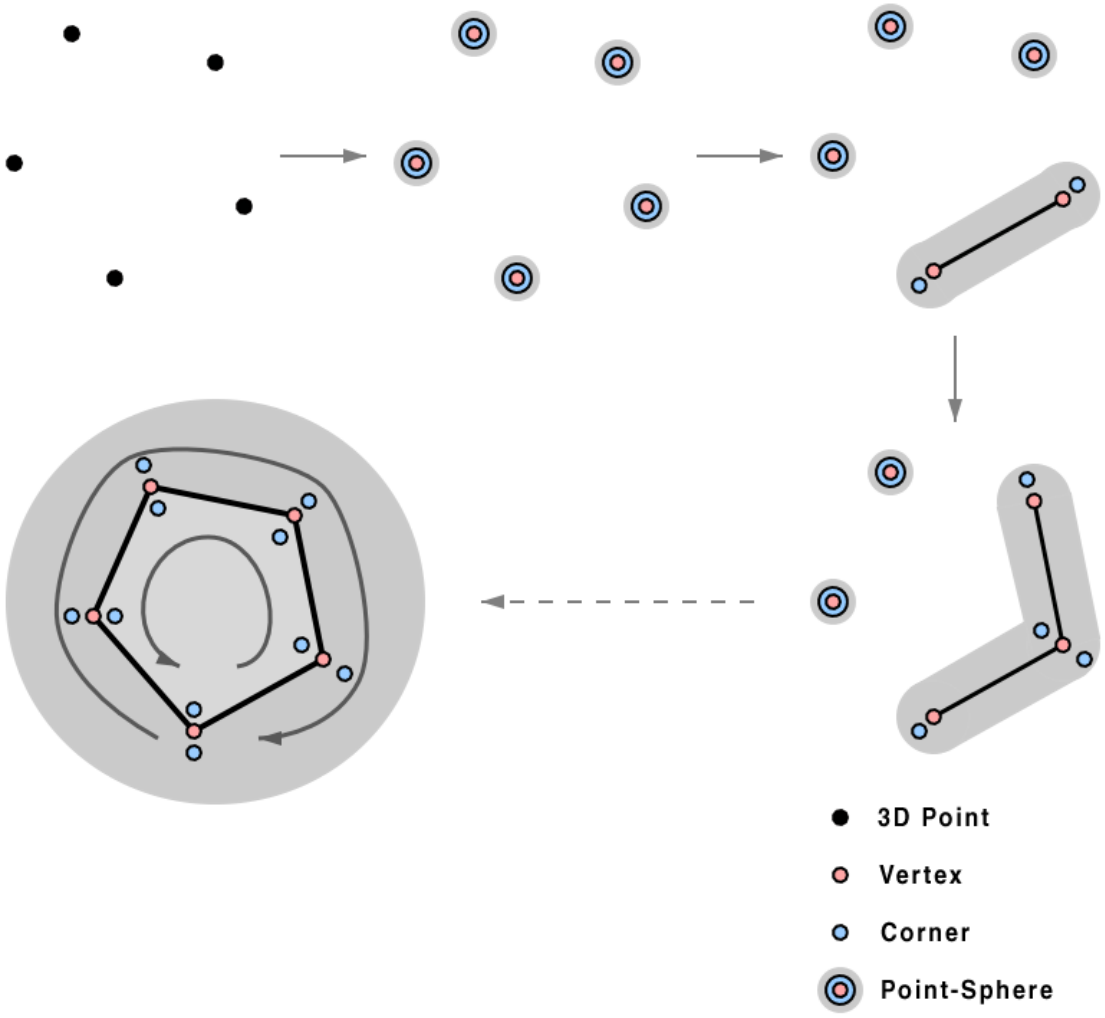


Fig. 11. The CreateFaceManifold operator.

$(p_0, p_1, \ldots, p_{N-1})$, and the other will contain the same points in the reverse order. We can think of the first face as the *front face* (f_f) and the second one as the *back face* (f_b).

1. for $i = 0$ to $N - 1$ do

   $(v_i, f_i) =$ CreateVertex(p_i);

   Let $c_i = \{v_i, f_i\}$.

2. for $i = 0$ to $N - 1$ do

   $e =$ InsertEdge$(c_i, c_{(i+1) \bmod N})$.

III.5. ConnectEdges$(\{\mathbf{e_1}, \mathbf{f_1}\}, \{\mathbf{e_2}, \mathbf{f_2}\})$

The ConnectEdges operator is another extension to the InsertEdge operator. Given two half-edges[3], the operator inserts two edges into the mesh as illustrated in Figure 12.

If $\{e_1, f_1\}$ and $\{e_2, f_2\}$ refer to the two half-edges, the first edge is inserted between the corner in f_1 where e_1 *starts* and the corner in f_2 where e_2 *ends*. The starting and ending corners are determined according to the rotation order in the respective faces. If $f = (v_0, v_1, \ldots, v_{N-1})$ and e is an edge in the face between the vertices v_i and v_{i+1}, then e is said to *start* at c_i and *end* at c_{i+1} where $c_i = \{v_i, f\}$ and $c_{i+1} = \{v_{i+1}, f\}$.

The operator creates a new face in the mesh bounded by e_1, e_2 and the two newly inserted edges. If f_1 and f_2 refer to the same face as shown in Figure 12, the operator effectively splits the face into 3 smaller faces, one of which will be the new face. If f_1 and f_2 refer to different faces (not shown), then the first InsertEdge operation will

[3]The term "half-edge" is used to refer to an {edge, face} pair

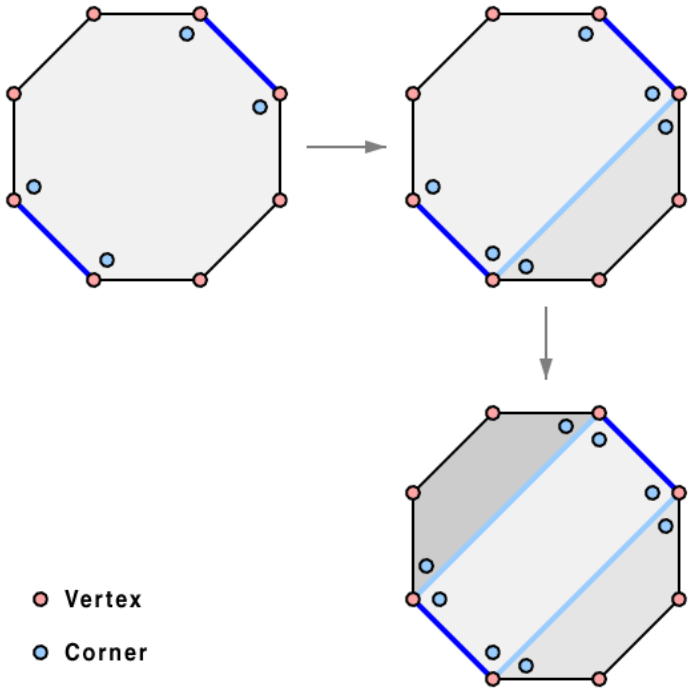


Fig. 12. The ConnectEdges operator.

introduce a handle and combine the two faces into one, thus changing the topology of the mesh. The second InsertEdge operation will split this face into two. The number of faces in the mesh will thus remain the same as before.

III.6. CollapseEdge(e)

The CollapseEdge operator removes an edge from the mesh and merges the two vertices at the end of the mesh into a single vertex. The merged vertex is positioned at the mid-point of the edge.

In the following implementation, one of the end points is retained and the other end point is merged into the first one. The merged vertex is then repositioned at the mid-point of the edge.

1. Let v_1 and v_2 be the end points of e. Compute the midpoint of e, $v_m = \frac{v_1+v_2}{2}$.

2. Let E_2 be the list of edges pointing to v_2. Let C_1 and C_2 be the list of corners pointing to v_1 and v_2 respectively.

3. Among the corners in C_1 find the corner c_1 at which e starts.

4. For every corner $c_{2,i} = \{v_2, f_i\}$ in C_2

   (a) Let $c'_{2,i}$ be the corner following $c_{2,i}$ in f_i.

   (b) if $c'_{2,i}$ does not point to v_1, InsertEdge$(c'_{2,i}, c_1)$.

5. For every edge $e_{2,i}$ in E_2, DeleteEdgeWithCleanup$(e_{2,i})$.

   Since the edge e will also be in E_2 we do not have to delete it separately.

6. $v_1 = v_m$. This re-positions v_1 at the mid-point of e.

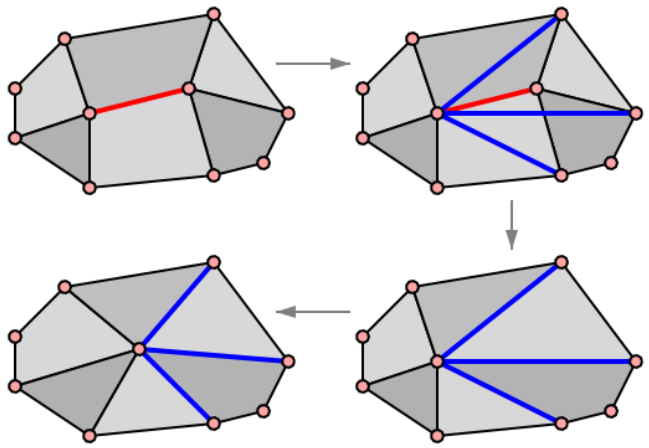


Fig. 13. The CollapseEdge operation.

Figure 13 illustrates the above sequence of operations. The edge shown in red is the edge that is to be collapsed. The edges inserted in step 4 are shown in blue color. The vertex v_2 does not have to be separately deleted – the DeleteEdgeWithCleanup operator automatically deletes it when the last edge pointing to it is deleted.

CHAPTER IV

MULTI-SEGMENT CURVED HANDLES

This chapter presents the first of the two interactive high-genus modeling tools that have been developed, namely creation of multi-segment, curved handles. The tool allows the creation of such handles between two faces belonging to two orientable 2-manifold meshes. Figure 14 shows an example of an object created using this tool.

Fig. 14. Unity knot. The image on the right was created using the multi-segment handle tool.

Creation of a handle between two faces of the same mesh increases the genus of the mesh by one. On the other hand, if the two faces belong to distinct mesh surfaces, then the two surfaces are connected but the genus does not increase.

It needs mentioning that handle creation is different from a loft between two faces, although visually they produce similar results. A loft creates an additional surface component, topologically equivalent to a 2-manifold with a boundary, which then has to be "stitched" to the original mesh. Handle creation is a topological operation and directly alters the topology of the mesh. Thus topological consistency becomes important for handle creation. The present approach not only guarantees the 2-manifold property of the final mesh, but the constructed meshes preserve the 2-manifold property at every stage of the handle creation process.

The CreatePipe operator, described in Section III.3, has emerged as one of the most useful high-level operators developed. However, a drawback of this operator is that the length of the edges in the handle can be much longer than other edges in the mesh. A straightforward solution to this problem is to split the handle into multiple segments, such that the edges in each individual segment are similar in length to the edges in the rest of the mesh. A useful side effect of this solution is that it allows the creation of a handle with an overall curvature, although individual segments of such a handle will still be straight.

An algorithm to create multi-segment handles followed by a modification which allows curved handles are presented in the following sections. For simplicity of explanation it is assumed that the two faces between which the handle is to be created have the same number of vertices. However, as in the case of the CreatePipe operator, this is not a restriction and faces with different number of vertices can easily be handled by a pre-processing step.

IV.1. Creating multi-segment handles

The algorithm to create multi-segment handles, shown graphically in Figure 15, proceeds as follows for a given mesh $\mathcal{M}$ and corners $c_1 = \{v_1, f_1\}$ and $c_2 = \{v_2, f_2\}$:

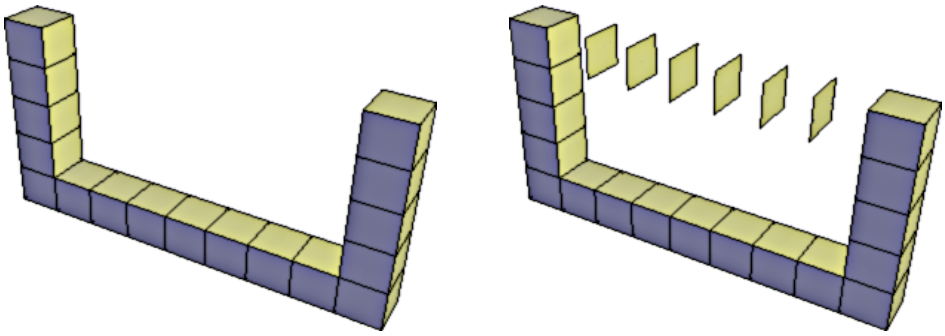
1. for $i = 1$ to $k - 1$ where k is the number of segments required in the handle

   (a) Compute the positions $(p'_{i,0}, p'_{i,1}, \ldots, p'_{i,N-1})$ of the vertices in face f'_i based on a linear interpolation between the original faces f_1 and f_2. The correspondence between vertices in the two faces is determined by the corners c_1 and c_2 and the rotation system defining the orientation of the faces.

   (b) $(f'_{i,f}, f'_{i,b})$ = CreateFaceManifold$(p'_{i,0}, p'_{i,1}, \ldots, p'_{i,N-1})$. This creates the two faces which form the boundary between segment i and segment $i + 1$ in the handle.

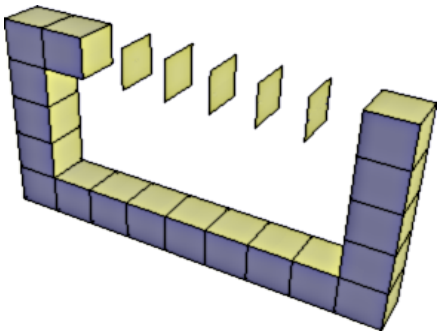
   This step creates a sequence of face pairs starting from $(f'_{1,f}, f'_{1,b})$ and ending at $(f'_{k-1,f}, f'_{k-1,b})$ as shown in Figure 15B. One vertex in each of the intermediate faces will lie on the line segment connecting v_1 and v_2.

2. Starting from $f'_{0,f} = f_1$ and ending at $f'_{k,b} = f_2$, going through the sequence of face pairs $(f'_{1,f}, f'_{1,b}), (f'_{2,f}, f'_{2,b}), \ldots (f'_{k-1,f}, f'_{k-1,b})$ created above, apply the CreatePipe operator to each pair of adjacent faces whose normals point towards each other. The *front* face of one pair $(f'_{i,f})$ and the *back* face of the next pair $(f'_{i+1,b})$ will satisfy this requirement. The corners chosen for the CreatePipe operation will be the ones corresponding to the vertices which lie on the line segment between v_1 and v_2. Figures 15C – 15F illustrate this step.

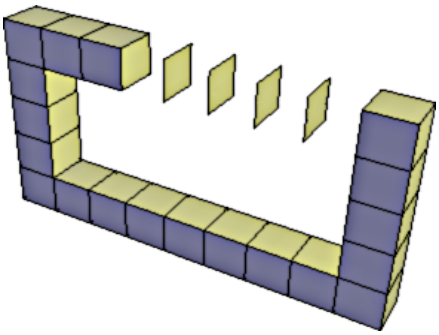


A

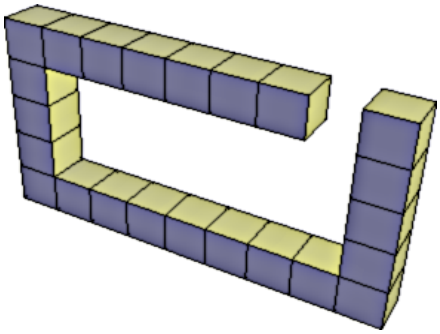
B



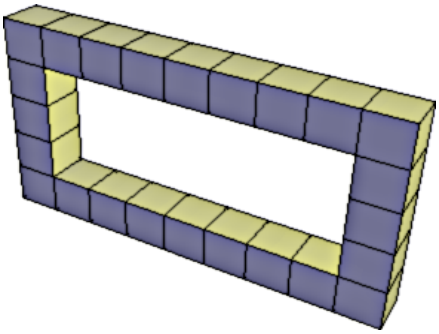
C



D



E



F

Fig. 15. Multi-segment handle creation.

IV.2. Curved handles

Now that we have a handle with multiple segments, one is naturally inclined to explore the possibility of creating a *curved* handle made of piecewise linear segments. An extension to the algorithm presented in the previous section which allows the creation of a curved handle is described in this section. A simple approach, using readily available geometric information, is used to create the handle. Essentially the only change in the algorithm described earlier is in the way the positions of the vertices in the intermediate faces are computed.

As before, two corners $c_1 = \{v_1, f_1\}$ and $c_2 = \{v_2, f_2\}$ are assumed to be given. In addition two weights w_1 and w_2, both of which can be specified by the user, are also defined. If f_1 and f_2 are star shaped faces with the same number of vertices, the goal is to create a multi-segment curved handle between them.

A Hermitian curve is used to determine the overall shape of the handle. The curve is defined by the following equation:

$$\mathcal{H}(t) = p_1 h_1(t) + p_2 h_2(t) + w_1 n_1 h_3(t) - w_2 n_2 h_4(t). \tag{4.1}$$

where t is the independent parameter which typically varies from 0.0 at the starting point of the curve (p_1) to 1.0 at the end point of the curve (p_2).

Here p_1 and p_2 are the centroids and n_1 and n_2 are the average unit normals of f_1 and f_2 respectively. $h_1(t), h_2(t), h_3(t)$ and $h_4(t)$ are the Hermite basis functions. This Hermitian curve starts from the centroid of face f_1 in the direction of the face normal (n_1) and ends at the centroid of face f_2, in a direction opposite to the face normal ($-n_2$).

The weights w_1 and w_2 are applied to the face normals n_1 and n_2 respectively. The weights provide limited control over the shape of the handle, since they affect

the shape of the Hermitian curve. Note that w_1 and w_2 can also be negative, which can be used to create holes instead of handles. Figure 16 shows two Hermitian curves illustrating how the normals affect the shape of the curve.

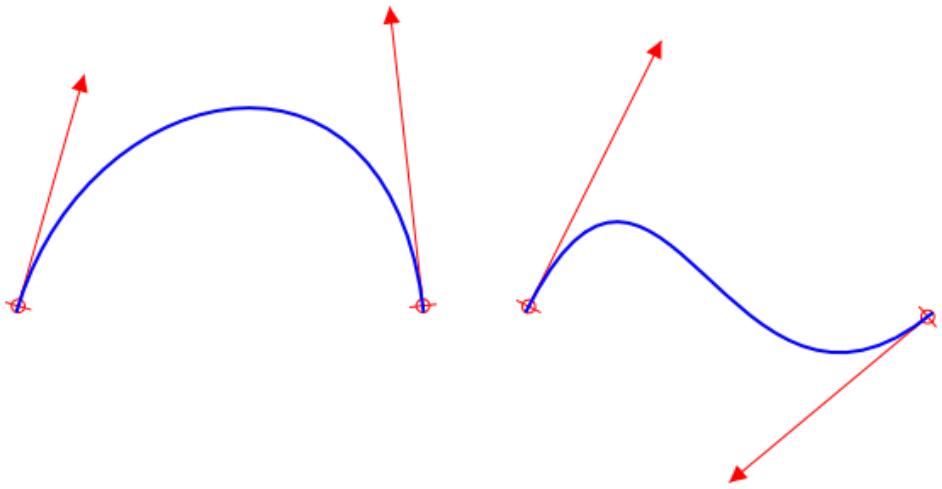


Fig. 16. Hermitian curves.

IV.3. Face morphing

The Hermitian curve $\mathcal{H}(t)$ determines the overall shape of the handle. To compute the actual positions of the vertices in the intermediate faces, the shape of the cross-section of the handle at each segment as it goes from f_1 to f_2 needs to be determined.

This problem strongly resembles the problem of 2D shape blending [34] (more widely known as 2D morphing). Therefore, in this approach, the problem is first simplified into a 2D problem. To morph the shape of the faces from f_1 to f_2 in 2D, a reference plane $\mathcal{R}$ is first determined and both faces are rotated to that reference plane. The choice of $\mathcal{R}$ is arbitrary and does not by itself affect the algorithm. However care must be taken to ensure that no degeneracies occur when performing the rotation. For example if the normal to the reference plane is exactly opposite to

the face normal, the rotation vector to rotate the face onto the reference plane cannot be determined, since the cross product between the two normals becomes zero.

The vector from the centroid of the first face to the centroid of the second face is used to determine the unit normal vector $n_{\mathcal{R}}$ to the reference plane $\mathcal{R}$. f_1 is rotated onto $\mathcal{R}$ around its centroid using the rotation axis $n_1 \times n_{\mathcal{R}}$. f_2 is similarly rotated.

Each vertex of the rotated f_1 and f_2 is then moved to the reference plane $\mathcal{R}$, to ensure that the transformed faces are planar. The faces are then translated to make their centroids coincide with the origin of the coordinate system. Thus, after applying the above operations, the two faces will be co-planar and both of their centroids will be at the origin.

As mentioned above, there is a possibility that the rotation vector becomes zero, namely when either $n_1 \bullet n_{\mathcal{R}} = -1$ or $-n_2 \bullet n_{\mathcal{R}} = -1$. This special case, where there is more than one possible solution for the overall shape of the handle, has been ignored.

IV.3.1. Face morphing in 2D

Under the assumption that these faces are star shaped and their centroids are star centers (i.e. any ray emanating from the centroid intersects with the face at most once), the morphing problem is simpler than the general 2D shape blending problem [34, 33]. However, care still needs to be taken to avoid self intersections. For instance, if a linear interpolation is performed directly between the vertices of the two faces, self intersections can occur.

Instead of linear interpolation between the vertex coordinates, the interpolation is done in polar coordinates. This interpolation guarantees that intermediate faces do not self-intersect. Using a reference axis system on the plane $\mathcal{R}$, every vertex v of f_1 and f_2 is resolved into a distance-angle pair (r, θ), where r is the distance of v from

the centroid of the face (which is now the origin) and θ is the angle that v (treated as a vector) makes with the X axis. The choice of the reference axis system is arbitrary and has no influence on the final results. Without loss of generality, the vector from the origin to the mid-point of the last edge in f_1 is taken to be the X axis. The Y axis is then defined by the cross product between $n_{\mathcal{R}}$ and the X axis.

Care is taken to ensure that the angles within the same face always increase monotonically, going from the first vertex to the last one. This is necessary to avoid self intersections as we transition from f_1 to f_2, and also gives a smooth transition. The difference in angles for the first vertices is also restricted to be less than 180 degrees. This prevents unwanted twists in the pipe (along the axis of the pipe).

Let

$$\begin{aligned} f_1 &= (v_{1,0}, v_{1,1}, \ldots, v_{1,N-1}) \\ f_2 &= (v_{2,0}, v_{2,1}, \ldots, v_{2,N-1}) \\ v_{1,j} &= (r_{1,j}, \theta_{1,j}),\ j = 0 \quad \text{to} \quad N-1 \\ v_{2,j} &= (r_{2,j}, \theta_{2,j}),\ j = 0 \quad \text{to} \quad N-1 \end{aligned} \tag{4.2}$$

be the resolved representations of the two transformed faces f_1 and f_2, where N is the number of vertices in each face.

A linear interpolation of the distance-angle pairs is then performed using the same parameter t as used for the Hermitian curve, such that $t = 0$ corresponds to the transformed f_1 and $t = 1$ corresponds to the transformed f_2. Let

$$f(t) = f'_i = (v'_{i,0}, v'_{i,1}, \ldots, v'_{i,N-1}) \tag{4.3}$$

be an interpolated face corresponding to the parameter t, where

$$\begin{aligned} v'_{i,j} &= (r'_{i,j}, \theta'_{i,j}),\ j = 0 \quad \text{to} \quad N-1 \\ r'_{i,j} &= (1-t)r_{1,j} + tr_{1,j} \\ \theta'_{i,j} &= (1-t)\theta_{1,j} + t\theta_{2,j} \end{aligned} \tag{4.4}$$

Figure 17 illustrates how the face morphing is done.

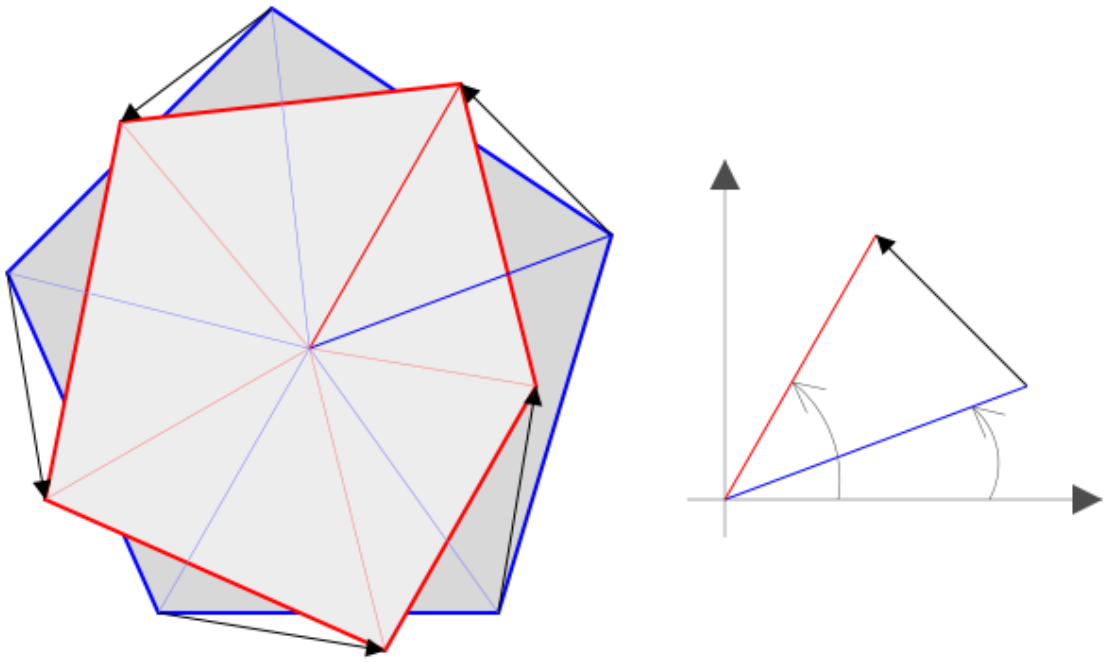


Fig. 17. Face morphing in 2D.

Using the Hermitian curve $\mathcal{H}(t)$, a point $p(t)$ and a tangent vector $n(t)$ are obtained for the given parameter t. $f(t)$ is then rotated such that its normal points in the same direction as $n(t)$ and is translated so that its centroid coincides with $p(t)$.

From this point, the process is identical to the second half of the multi-segment handle creation algorithm described in Section IV.1.

IV.4. Implementation

An outline of the implementation of the algorithm described above is given below. Two corners $c_1 = (v_1, f_1)$ and $c_2 = (v_2, f_2)$ and two weights w_1 and w_2 are the inputs to the algorithm. Figure 18 illustrates the operation of the algorithm.

1. Let N be the number of vertices in the f_1 and f_2.

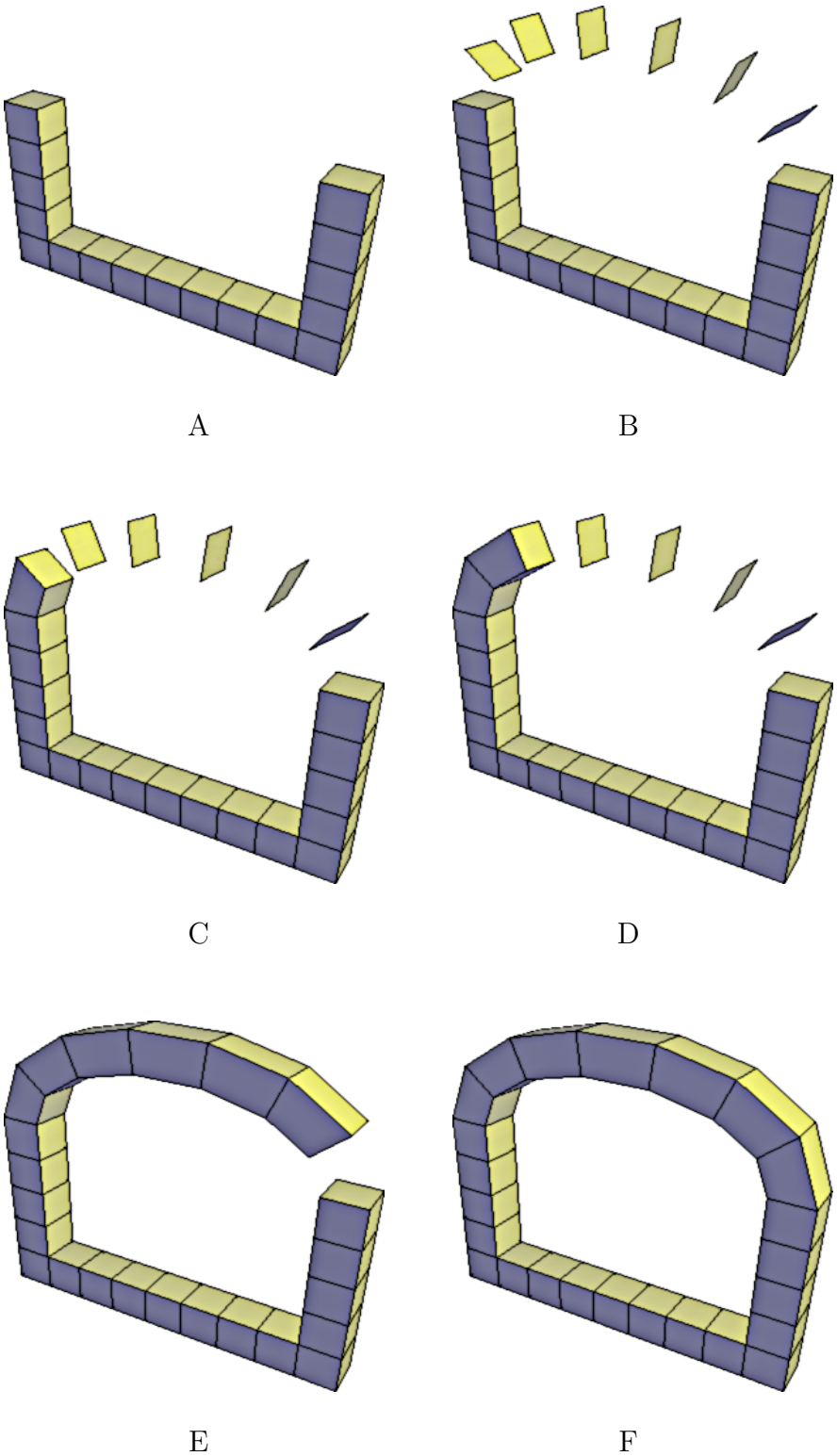


Fig. 18. Multi-segment curved handle creation.

2. (a) Reverse face f_2.

   (b) Using the centroids p_1 and p_2 and weighted normals n_1 and n_2 of the two faces, compute a Hermitian curve $\mathcal{H}$

3. (a) Compute the reference plane $\mathcal{R}$ with normal $n_{\mathcal{R}}$.

   (b) Transform f_1 and f_2 into f_1' and f_2' such that the normals of f_1' and f_2' are in the same direction as $n_{\mathcal{R}}$ and their centroids are at the origin.

   (c) Resolve f_1' and f_2' into distance-angle pairs. (Equation 4.2).

4. for $i = 1$ to $k - 1$ where k is the number of segments required in the handle

   (a) Compute $t = i/k$

   (b) Compute a point p_i and tangent n_i on the curve $\mathcal{H}$ for t

   (c) Perform a linear interpolation between f_1' and f_2' using t as the parameter to obtain $f_i' = (p_{i,0}', p_{i,1}', \ldots, p_{i,N-1}')$ (Equations 4.3 and 4.4)

   (d) Transform f_i' to $f_i'' = (p_{i,0}'', p_{i,1}'', \ldots, p_{i,N-1}'')$ such that the normal to f_i'' points in the direction of n_i and its centroid coincides with p_i.

   (e) $(f_{i,f}'', f_{i,b}'') = \textsc{CreateFaceManifold}(p_{i,0}'', p_{i,1}'', \ldots, p_{i,N-1}'')$

5. Starting from $f_{0,f}'' = f_1$ and ending at $f_{k,b}'' = f_2$, going through the sequence of face pairs $(f_{1,f}'', f_{1,b}''), (f_{2,f}'', f_{2,b}''), \ldots (f_{k-1,f}'', f_{k-1,b}'')$ created above, apply the CreatePipe operator to each pair of adjacent faces whose normals point towards each other. The *front* face of one pair $(f_{i,f}'')$ and the *back* face of the next pair $(f_{i+1,b}'')$ will satisfy this requirement. The corners chosen for the CreatePipe operation will be the ones corresponding to the vertices which lie on the curve between v_1 and v_2.

IV.5. Examples

In the following figures, the faces to be connected have a darker shade and the selected corners in those faces are indicated as white dots.

Figure 19 shows two examples of creating a handle between two faces which originally have different number of vertices. Three examples of face morphing for different choices of corners are shown in Figure 20. Notice that by varying the choice of corners, a twist is automatically introduced into the handle.

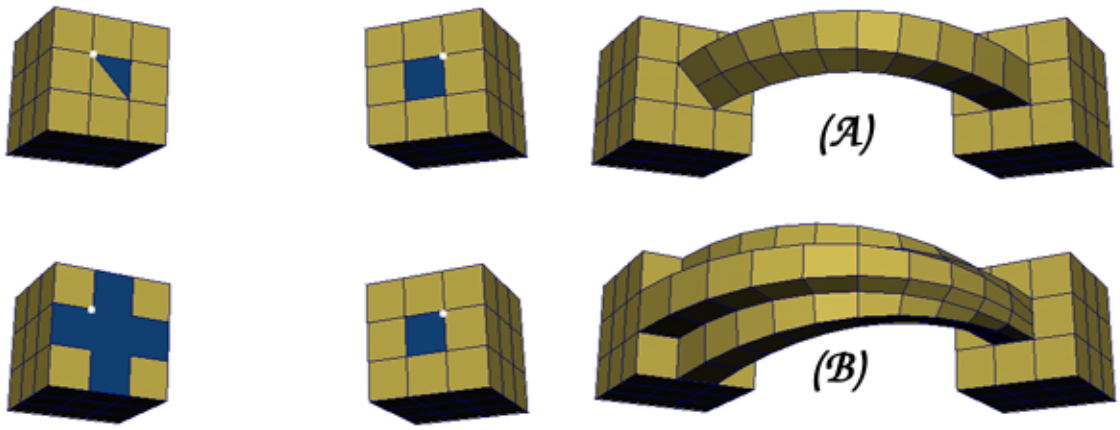


Fig. 19. Two examples of creating handles between faces with different number of vertices.

The morphing algorithm used in the handle creation cannot guarantee absence of self-intersections for handles between non-star faces. However, it can still be used in some situations as shown in Figure 21, where the handles do not self-intersect inspite of the end faces being non-star shaped.

To create the segments of a handle, the Hermitian curve is sampled using $t = i/k$ where k is the number of segments in the curve and i is the index of the current segment and varies from 0 to $k - 1$. As can be expected this produces segments of unequal length, with shorter segments in regions of high curvature and longer

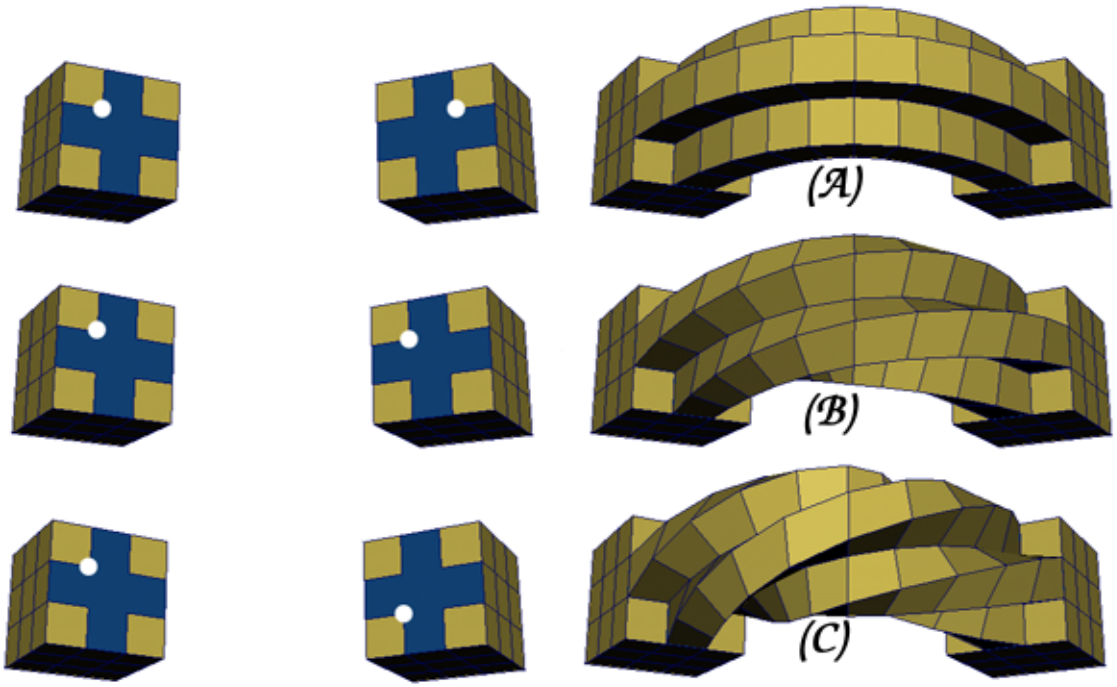


Fig. 20. Examples that show face morphing for different choice of corners.

segments in other regions. To improve the quality of the handles alternative sampling strategies were tested, such as using an appropriate step length with respect to the curvature of the Hermitian curve. The results were not significantly different in terms of visual appearance.

The main problem occurs when the handle intersects itself. However, if the Hermitian curve intersects with itself for the user selected weight values w_1 and w_2, there is no solution to this problem. Since both high curvature and curve self-intersection occur for larger values of w_1 and w_2, it is easier to reduce the values of these parameters. A very small number of segments is usually sufficient to construct handles. Unless the curvature is extremely high, a small number of segments do not intersect each other. In interactive applications users can easily avoid self intersection by either decreasing the values of w_1 and w_2 or using less segments.

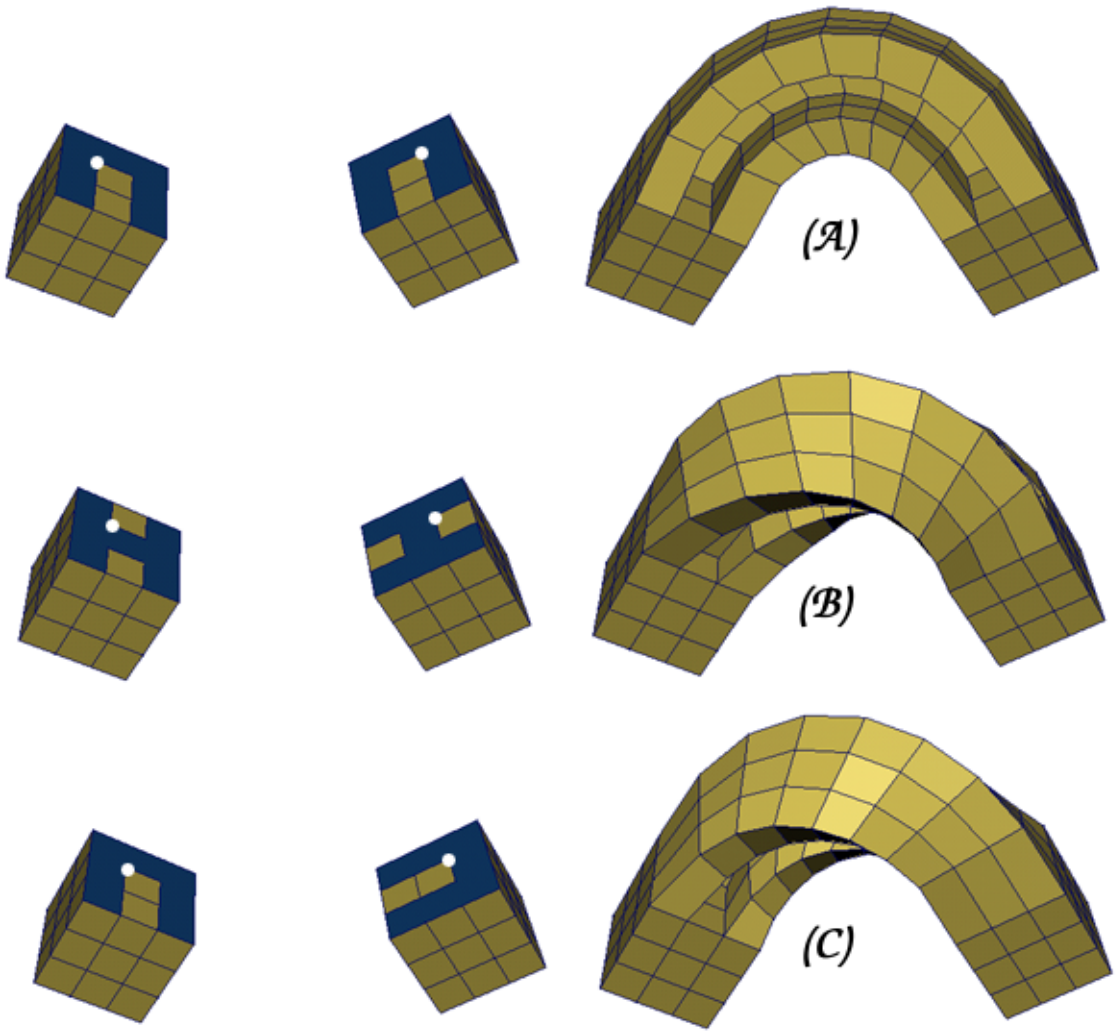


Fig. 21. Examples that show face morphing between non-star shaped faces.

CHAPTER V

RIND MODELING

For the second interactive approach to high-genus modeling, we return to the sculptures that provided the inspiration for this research. In this chapter, we will look at a tool to create high-genus rind shapes of the kind shown in Figure 22 below and Figures 1, 2, 3 and 5 in Chapter I. Figure 22 also shows a high-genus model (the floating head) created using the rind modeling tool. A *rind shape* can be thought of as a shell enclosing a volume. When we think of a rind shape as an orientable surface, the interior of the surface is the region *between* the two surfaces which make the shell and not the volume enclosed by the shell itself.

Fig. 22. High-genus head meets high-genus elephant. The head model was created using the rind modeling tool.

Among Escher's drawings are three rind shaped figures, titled *Study for Rind*, *Rind* and *Bond of Union* [8]. All three images actually depict genus zero surfaces,

but their structure is such that they can be modeled using the same approach as used for high-genus rind models. Figure 23 shows an example of a genus zero rind surface created using the rind modeling tool.

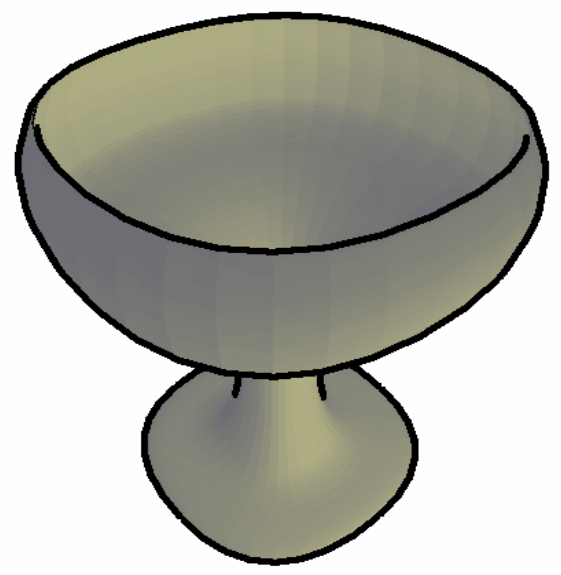


Fig. 23. A genus zero rind surface created using the rind modeling tool.

Rind shapes are also extremely common in the world of man-made objects. Examples include bottles, teapots, masks, boxes and even houses, many of which are not high-genus objects, but readily suggest an approach based on high-genus modeling.

The process of creating a rind shape starting from a 2-manifold mesh surface can be split into two steps. In the first step an offset surface is created for the given mesh. In the second step holes are punched in the shell or crust produced in the first step. Punching a series of holes adjacent to each other can be considered as a form of peeling, similar to peeling the rind of an orange.

Although the process appears to be a simple one, there are several issues to consider, especially when we consider topological consistency of the mesh surface and user interactivity. These issues are discussed later in this chapter.

V.1. Simple algorithm for rind modeling

At first glance, it appears that creating a rind for any given shape is an easy process. A naive algorithm for the construction of high-genus rind surfaces might proceed as given below. For the second stage, the algorithm makes use of the CreatePipe operator described in Section III.3.

Stage 1: Offset surface creation. This stage consists of two steps.

1. Duplicate the initial mesh and scale the vertices of the new mesh about their centroid so that they lie completely inside the first mesh. This operation creates two nested surfaces, the *initial mesh* and the newly created *offset mesh* with each face in the outer surface having a matching face in the inner surface.
2. Reverse the normals of the faces of the offset mesh. This operation changes the inside and outside of a 2-manifold mesh by changing the rotation orders of faces. This step is necessary to produce a topologically valid rind surface.

Stage 2: Hole punching and peeling. From the user's point of view, peeling is identical to hole punching. However the two are topologically different as described below.

Hole punching This involves connecting a face in the initial mesh to its corresponding face in the offset mesh using the CreatePipe operator. The process can be repeated to produce as many holes as desired.

The first hole that is punched combines the inner and outer surfaces into a single surface. Formally, the first CreatePipe operation changes the topology of the object, but it does not increase the genus of the object.

Subsequent application of the CreatePipe operator changes the topology as well as increases the genus by one [3].

Peeling As explained earlier, punching a series of holes adjacent to each other is essentially a peeling operation. But punching holes as described above does not automatically create a peeled effect.

Punching holes in two neighboring faces leaves an infinitely thin wall or slab (like a thin membrane) between two neighboring holes as shown in Figure 24. Users have to delete such walls to create a peeled look.

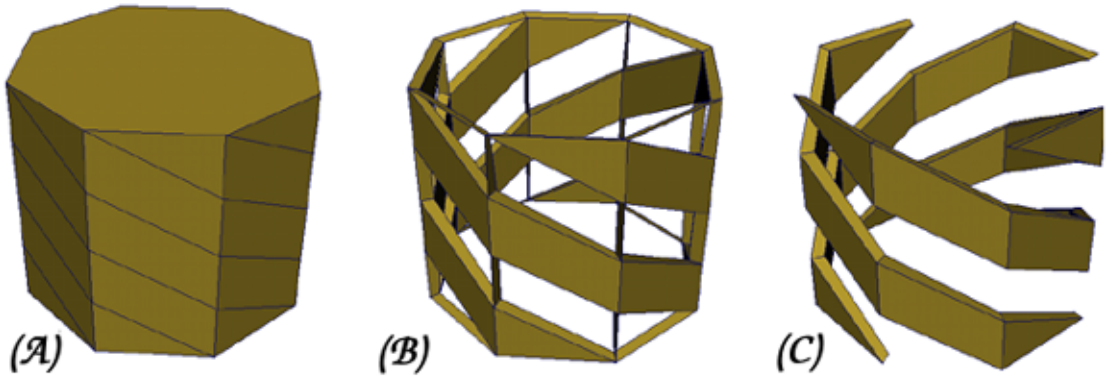


Fig. 24. An example of the need for peeling. (A) shows an initial mesh and (B) shows infinitely thin walls left over by hole punching. The mesh shown in (C) is obtained after deleting these infinitely thin walls.

V.2. Problems with the naive algorithm

Theoretically, it is possible to create almost any high-genus manifold rind shape with algorithm described above. But there are several usability problems with the above approach as explained below.

V.2.1. Offset surface creation

For simple convex meshes, the naive algorithm can produce an offset mesh which is completely inside the initial mesh. But if the object is not convex or star-like, it may be necessary to move different vertices by different amounts to produce the offset mesh just to ensure that the offset surface is completely inside the initial mesh. This can be extremely cumbersome and time consuming even for simple objects with a large number of faces.

The problem becomes more complicated when we want to create a rind of uniform thickness. For high-genus initial meshes, using the above approach can result in intersections between the inner surface and the outer surface as shown in Figure 25.

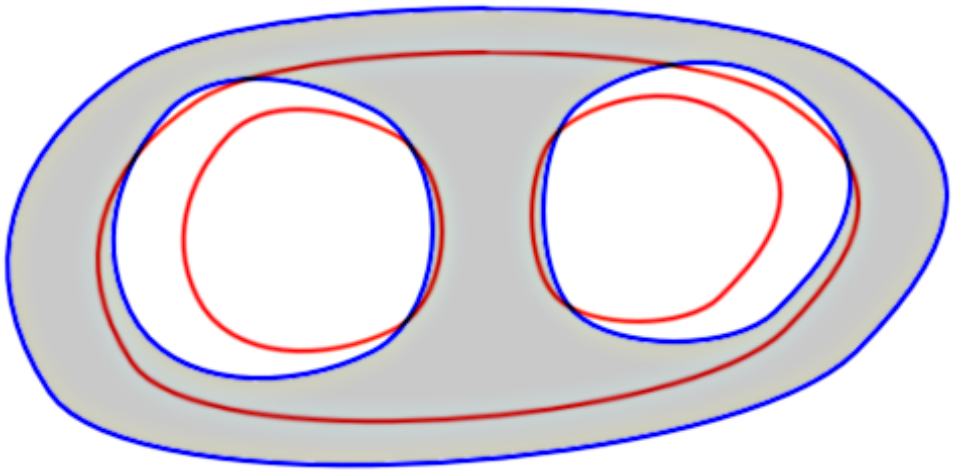


Fig. 25. Problem of creating the rind using the naive algorithm for a high-genus surface.

Figure 26 shows an x-ray view of a teapot model created using the naive algorithm. Although the model is a valid manifold, it is far from satisfactory since the thickness of the rind is highly non-uniform, as is evident from the x-ray image.

Fig. 26. X-ray view of a rind teapot model created using the naive algorithm.

V.2.2. Hole punching

As mentioned above, hole punching makes use of the CreatePipe operator. Recall that this operator requires two corners of the mesh to be specified. For hole punching, one corner will be from a face in the outer mesh and the other will be from the matching face in the inner mesh.

Selecting the correct corner from the inner surface is not easy, since the the inner surface is only partially visible even through holes which have already been punched. For punching the first hole the difficulty is even more pronounced since the inner surface is completely invisible from outside.

V.2.3. Peeling

As mentioned earlier, punching holes in two neighboring faces leaves an infinitely thin wall. Users have to delete such walls to create a peeled look. Visually, it appears as though the wall is made up of a single two-sided quadrilateral. But a two sided quadrilateral is not a valid entity in an orientable 2-manifold surface and cannot be created using the four fundamental topology change operators. In terms of the mesh structure, the wall actually consists of two coincident quadrilaterals. Since all the

edges are straight, the existence of more than one edge in the same location is not visually apparent.

To illustrate the problem with deleting the thin wall, the edges of one of the quadrilaterals is assumed to be curved as shown in Figure 27A. Although this figure does not make sense geometrically, it helps to conceptually visualize the structure of the infinitely thin wall.

From the figure it can be seen that the infinitely thin wall is in fact a handle. This handle can be deleted by deleting the two edges e_1 and e_2 as shown in Figure 27. Deleting e_1 combines the two quadrilaterals and creates a hexagonal face [3]. This hexagon is still a handle, which is eliminated after the deletion of e_2, thus separating the hexagon into two two-gons.

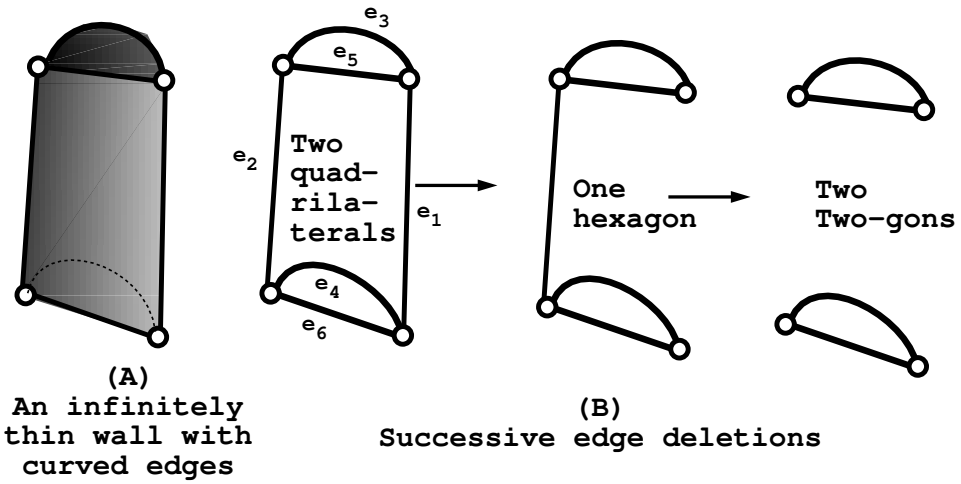


Fig. 27. Deletion of a an infinitely thin pipe by deleting only two edges. Initially this infinitely thin pipe consists of two quadrilaterals.

Deleting the two edges is not an easy process from the user interface point of view. The first edge (e_1 or e_2) can be easily deleted. However, it can be difficult to delete the second edge, since the first edge deletion creates a non-planar hexagon which cannot be properly rendered. After the first edge has been deleted, the second

edge becomes visible only from some orientations.

After the two edges have been deleted, we are left with two two-gons. These two-gons do not carry any extra visual or geometric information, since they are identical (visually and geometrically) to a single straight edge. Although these two-gons do not cause any apparent problems for creating rind shapes, they do cause visual artifacts when subdivision schemes are applied to the mesh as can be seen in Figure 28B. Therefore it is desirable to delete these two-gons from the mesh.

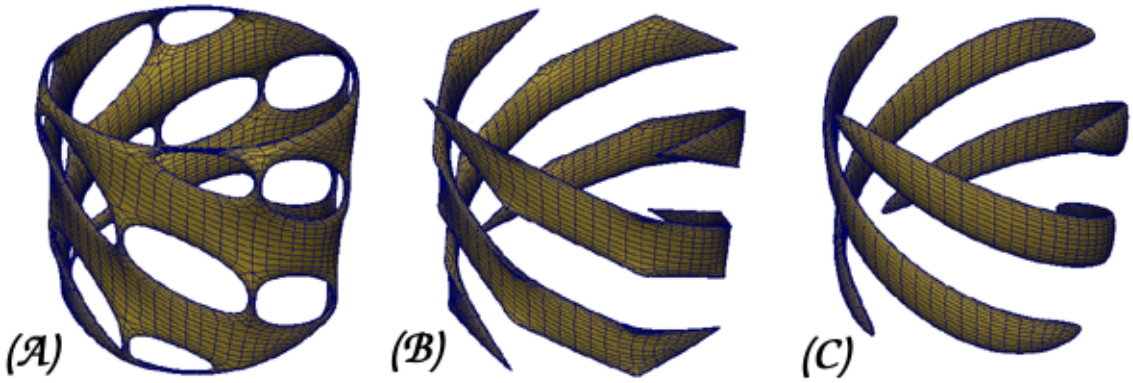


Fig. 28. The effect of two-gons in smoothing with subdivision. (A) shows a subdivided version of 24B and (B) and (C) show subdivided versions of 24C with and without two-gons. Note that the version with two-gons shows tangent discontinuities. In all cases, Catmull-Clark [10] subdivision was applied twice.

Figure 29 illustrates how deletion of one edge of a two-gon eliminates the two-gon. With reference to Figure 27, deleting e_3 and e_4 will eliminate the two-gons. However, this operation is not easy since their existence cannot be visually inferred and deleting one edge of a two-gon does not produce any visual change. Moreover, the edges of two-gons are usually very short and therefore it is very difficult to select and delete them by hand. In other words, even if the user tries to delete all such edges, some may still be overlooked.

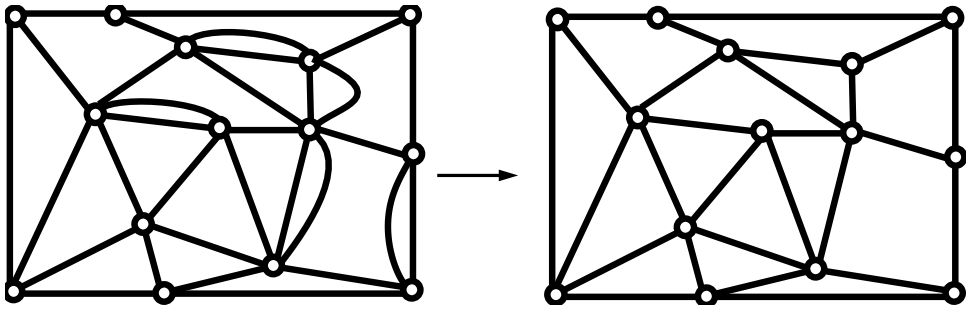


Fig. 29. Two-gons are removed by deleting one of their edges. In order to show the actual mesh structure, one of the edges of every two-gon is drawn curved.

V.3. Improved rind modeling algorithm

In this section, an approach which addresses all of the above mentioned problems and provides a simple interactive method is presented. The usability problems described in the previous section are resolved through a semi-automatic approach as outlined below.

1. The offset surface is created automatically by the system based on a user specified thickness parameter. This addresses the problem of non-uniform rind thickness obtained using the naive algorithm.

2. The system automatically establishes a correspondence between the faces of the outer mesh and the inner mesh. Thus punching and peeling only require the selection of a face in the outer mesh. The invisibility or partial visibility of the inner mesh becomes a non-issue since the user no longer has to select matching corners in the inner mesh and the outer mesh.

   It must be mentioned that the problem of invisibility of the inner surface can be addressed by using transparency when rendering the mesh. Although this alleviates the problem to some extent, this is not an effective solution in very dense meshes where transparency can create difficulties in distinguishing faces

of the inner mesh from those of the outer mesh.

3. Since the system has access to the internal structure of the mesh, the infinitely thin wall between neighboring holes is automatically eliminated. The system also cleans up the two-gons to achieve a smooth, peeled rind.

A more detailed description of the above approach follows. The algorithm can be split into two parts - an automatic step and an interactive step.

V.3.1. Automatic step: Offset surface creation

This step is automatically initiated when the system enters the *rind modeling mode* and proceeds as follows:

1. Duplicate the initial mesh to create an offset mesh, reverse the normals and compute average unit normals for each vertex.

2. Create a *correspondence table* which maps each face in the initial mesh to its matching face in the offset mesh. The face correspondence also establishes matching corners in the two faces.

3. Move each vertex of the offset mesh a distance equal to the thickness value in the direction of its average normal vector.

The offset surface construction problem for 2-manifold meshes is very different from offset surfaces in solid modeling [30, 18] since the shapes of faces of 2-manifold meshes do not have to be well-defined. For 2-manifold meshes a distance function cannot be defined. Therefore a "correct" offset surface cannot be rigorously defined. Thus, any procedure to create offset surfaces for 2-manifold meshes must be somewhat ad hoc.

The above procedure may result in self intersection for high values of the thickness parameter. It is possible to accommodate the self intersection by changing the topology and mesh structure as shown in Figure 30. However such a change in the topology of the rind surface means that some metric (such as distance between faces and their relative orientation) will have to be used to find matching faces between the initial mesh and the offset mesh. This could potentially result in a one-to-many correspondence between faces in the initial mesh and those in the offset mesh. It also means that there will be some faces in the initial mesh for which a matching face in the offset mesh doesn't exist and vice versa.

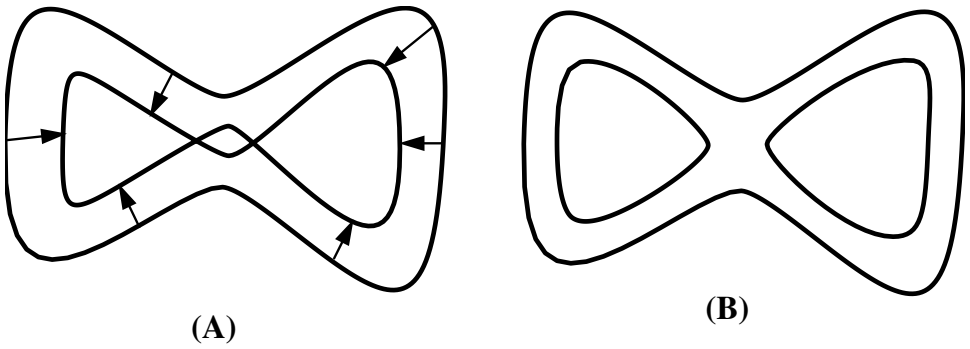


Fig. 30. Avoiding self intersection for high thickness values (A) requires a change in topology as shown in (B).

The procedure is greatly simplified if the initial and the offset surfaces have the same mesh structure and a one-to-one correspondence exists between faces in the initial mesh and the offset mesh. In this approach, self intersections can be avoided simply by using smaller thickness values. This procedure is practically useful and successfully creates acceptable offset surfaces for a wide range of initial meshes. An advantage of this procedure is that it is extremely simple, and therefore, suitable for interactive applications.

V.3.2. Interactive step: Hole punching and peeling

Once the offset surface has been created, the interactive step is initiated. During the interactive step, the system can be in two modes: hole punching or peeling. The mode can be changed during the interactive process. In either mode, the user can punch holes or peel simply by selecting faces of the mesh. Selecting a face which was created by a previously punched hole does not have any effect.

Hole punching mode The algorithm for punching a hole is extremely simple. Let f_1 be the face selected by the user and $\mathcal{T}$ be the correspondence table created in the automatic step.

1. If $\mathcal{T}$ contains f_1 find the matching face f_2 and the matching corners in the two faces c_1 and c_2.

   If $\mathcal{T}$ does not contain f_1 the selection is ignored and the hole punching procedure stopped. Control is returned to the user to select another face.

2. CreatePipe(c_1, c_2)

Peeling mode The first part of the algorithm for peeling is identical to the algorithm for hole punching. In the second part, the infinitely thin wall and the two-gons are cleaned up. The second part makes use of the fact that for every edge in the selected face, there is a matching edge, in the matching face obtained through the correspondence table.

1. If $\mathcal{T}$ contains f_1 find the matching face f_2 and the matching corners in the two faces c_1 and c_2.

   If $\mathcal{T}$ does not contain f_1 the selection is ignored and the hole punching procedure stopped. Control is returned to the user to select another face.

2. CreatePipe(c_1, c_2)
3. For every edge $e_{1,i}$ in f_1 find the matching edge $e_{2,i}$ in f_2.
4. If the two faces adjacent to $e_{1,i}$ are the same as the two faces adjacent to $e_{2,i}$, then we have a thin wall which has to be cleaned up. This merely involves deleting all the edges of one of the faces adjacent to either $e_{1,i}$ or $e_{2,i}$.

V.4. Examples

Figure 31 shows examples of rind shapes created using this approach. The models have been smoothed by applying a subdivision scheme to the rind surface.

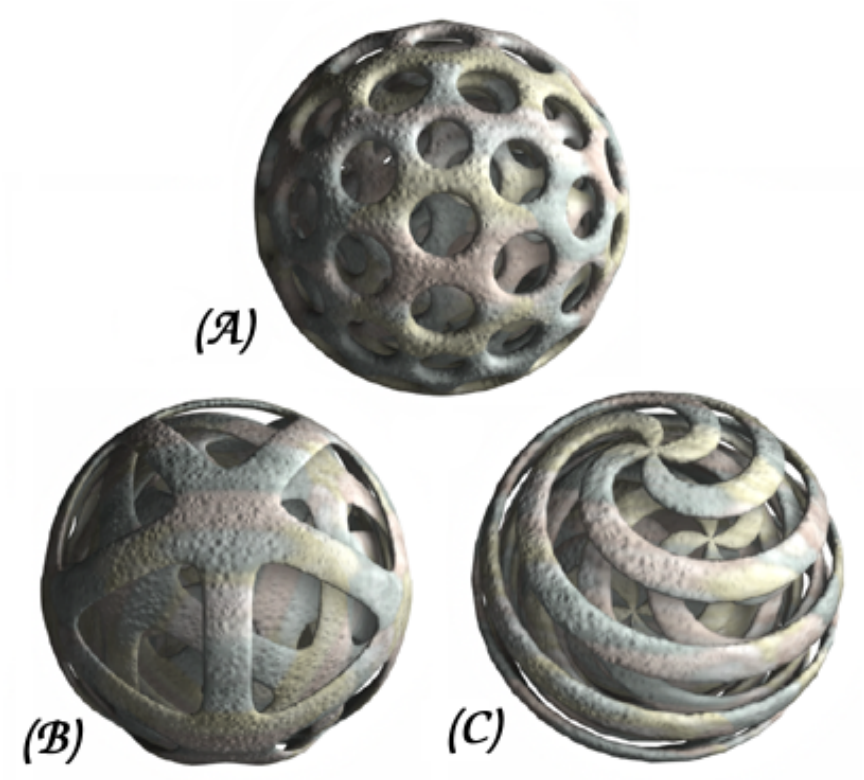


Fig. 31. Examples of spherical rind shapes. The nested structures are obtained by duplicating, scaling and rotating the outer rind shapes.

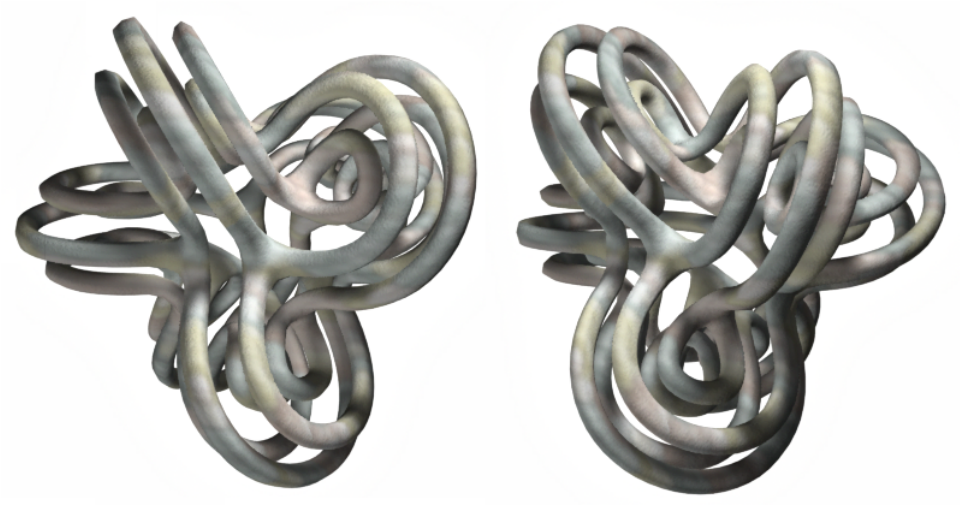


Fig. 32. Two views of a shape that can be created by the rind modeling approach, but does not look like a rind shape.

The rind modeling method can also be used to create shapes that do not necessarily look like rind shapes. An example of such a shape is shown in Figure 32. The surface in this figure looks more like an extruded surface than a rind surface, but it cannot be created as a simple extrusion because of its branching structure.

As mentioned earlier, manifold models are also functional models. The teapot shown in Figure 33 is an example of a functional model that can be created by the rind modeling tool. As can be seen from the x-ray and cut away images, this teapot has a *real* not just an apparent hole to let the water pour from the spout, allowing the model to be used in physical simulations.

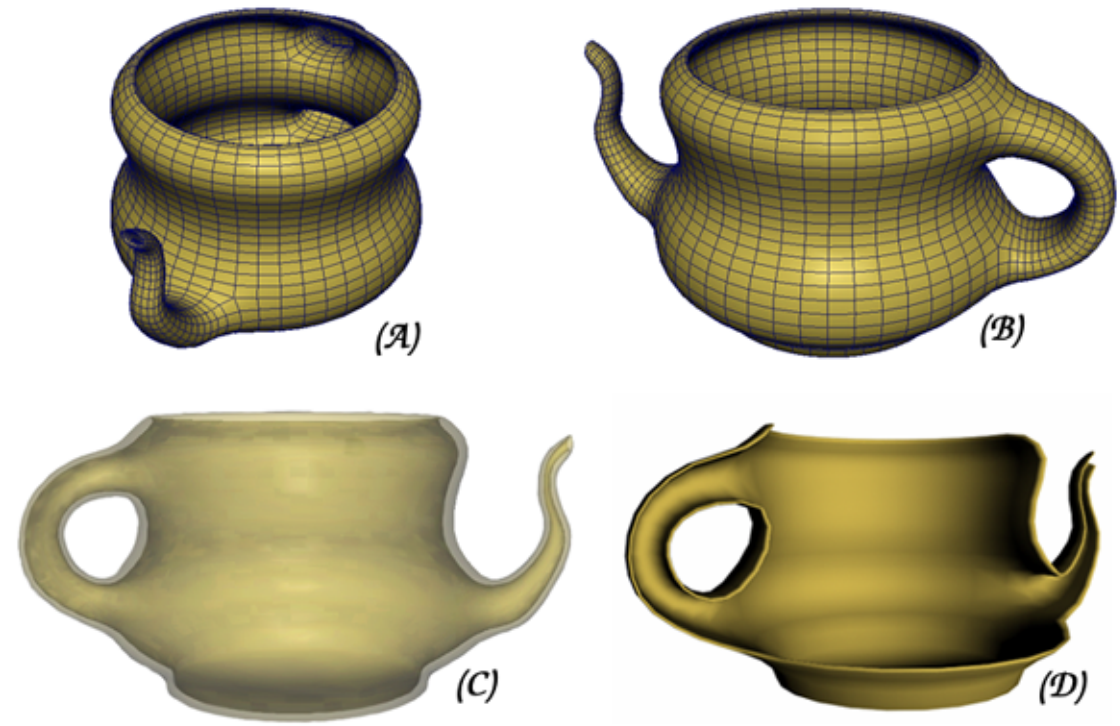


Fig. 33. A rind-shaped teapot. (A) and (B) show two different views of the manifold mesh. (C) is an X-ray image using transparency. (D) is sliced to show the interior.

CHAPTER VI

GENERALIZED SIERPINSKI POLYHEDRA

The Sierpinski tetrahedron is the three-dimensional version of the Sierpinski gasket (Figure 34). A Sierpinski *polyhedron* is a generalization of the Sierpinski tetrahedron where the overall shape can be any polyhedron.

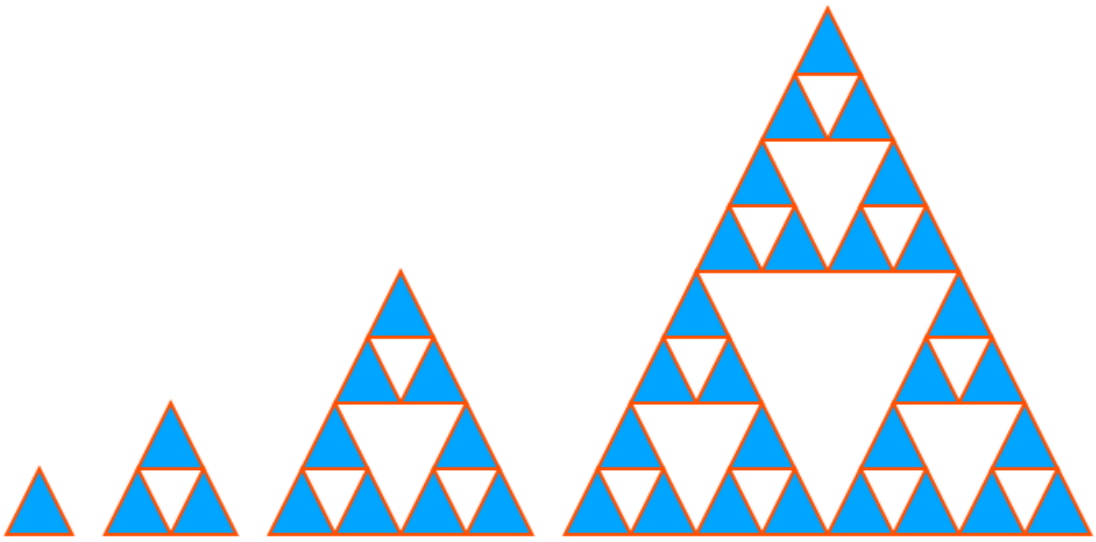


Fig. 34. The Sierpinski gasket. The sequence shows the construction up to 4 iterations using the union operation.

Most shape construction algorithms for fractal geometry are given by a set of rules that are applied to an initial shape [6]. However, the limit surfaces are often independent of the 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, subdivision schemes have three advantages: (1) their underlying rules (remeshing schemes) are mesh topological in nature, (2) the rules can be easily applied to any manifold

polygonal mesh, (3) the limit shapes depend on the initial shapes.

In this chapter a scheme to construct generalized Sierpinski polyhedra is presented. Most current Sierpinski tetrahedron construction schemes create either an infinite set of disconnected tetrahedra or a non-manifold polyhedron. In contrast, the scheme described here creates one connected and manifold polyhedron. The new scheme can be applied to any manifold polyhedral mesh. Depending on the shape of the initial polyhedron, the new scheme can construct a wide variety of Sierpinski polyhedra.

VI.1. Current construction approach

Most fractal geometrical shapes, including the Sierpinski tetrahedron are self-similar, i.e. parts of the shape have the same form as the entire shape. This self-similarity property [6] is exploited by algorithms for construction of such shapes. One of the most common approaches is to create the fractal shape by repetitively taking the union of geometrically transformed copies of an initial shape, as shown for the Sierpinski gasket in Figure 34. For example, if a self-similar fractal shape is given as

$$S = \bigcup_{k=0}^{K} A_k S$$

where A_k is a transformation matrix in a homogeneous coordinate system, then constructing such a shape involves creating a series of shapes starting from an initial surface S_0 as

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

.

In the actual implementation of such algorithms, the union operation is usually ignored, since it is impossible to visually distinguish between a shape constructed

using disconnected copies placed at the correct locations and one that makes use of the union operation. This results in the number of copies of the initial shape increasing geometrically – i.e. by a constant factor (K in the above equations) – in each iteration, giving a good approximation of the limit surface after a few iterations. This approach is widely used because of its simplicity. The procedure is also dimension independent, i.e. the same conceptual algorithm can be used in 2D as well as 3D. The algorithm is also independent of the actual representation of the shape, which can be a set of points, a polygonal mesh or an implicit surface.

VI.2. Alternative approaches

As mentioned earlier, algorithms based on the above approach are specific to the target (limit) shapes and they are independent of the shape of the initial object, i.e. each algorithm approaches its target shape whatever be the initial shape.

There exist alternative approaches for constructing fractal shapes. These are usually dimension dependent and are hard to implement in 3D. As such, they have not been widely used in 3D applications. However, using these alternative approaches, a variety of fractal shapes can be constructed 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 [27]. This particular approach is built upon in the following sections to develop a generalized Sierpinski polyhedron construction algorithm.

VI.3. Generalization of Mandelbrot's alternative Sierpinski triangle construction

Although not explicitly stated by Mandelbrot, the above construction scheme does not require that the initial shape be a uniform triangle. The scheme can be applied to any 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*" (see Figure 35). Notice that, after the first application of this modified algorithm, all polygons become triangles, and the original scheme can be applied without modification.

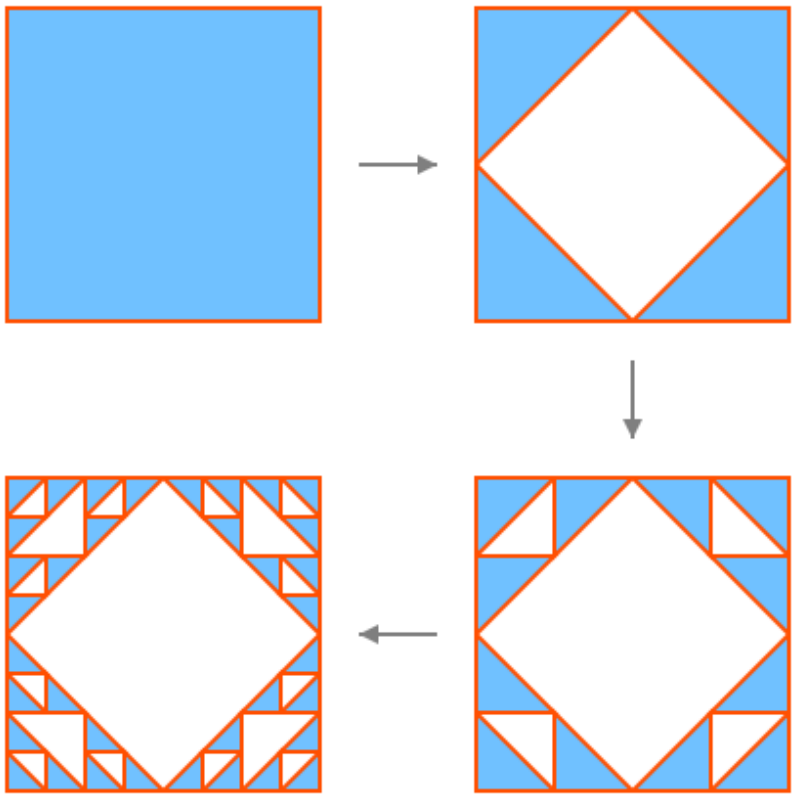


Fig. 35. Generalization of Mandelbrot's alternative Sierpinski triangle construction to convex polygons.

VI.3.1. Extension to 3D

To be able to generate polyhedra, it will be useful if this algorithm can be extended to three dimensions. However, it is difficult to extend this algorithm to three dimensions using set operations. Construction of a generalized Sierpinski polyhedron requires a set-difference of the initial polyhedron with a polyhedron that is constructed by connecting midpoints of each edge in the original polyhedron. This poses 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 [24].

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 in which the faces are triangular and therefore planar. But for the general case, the faces may not be triangular and hence may not be planar, complicating the set-difference procedure even further.

VI.4. 3D version of generalized Sierpinski triangle construction

This section presents an overview of an algorithm that provides a three-dimensional version of the generalized Sierpinski triangle construction. The scheme is very similar to subdivision schemes [43, 31, 12, 10]:

- Like every subdivision scheme, the algorithm is based on a simple remeshing scheme.

- The algorithm can be applied to any polygonal manifold mesh.

- Subdivision schemes creates regular regions that approach parametric surfaces such as cubic B-Spline surfaces. This algorithm also creates regular regions that approach Sierpinski tetrahedra.

- Similar to subdivision surfaces, this 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.

The algorithm differs from subdivision schemes in that it changes the topology of the initial mesh. However, unlike algorithms for self-similar fractals, this algorithm does not create disconnected surfaces. Each connected component in the initial manifold mesh remains connected after application of the algorithm. It must be mentioned that this property is important to be able to produce topological high genus surfaces – constructing a polyhedron with disconnected components does not increase the genus of the resulting object, although visually it does appear to have high genus. With the new scheme, after the first iteration, the genus of the mesh increases by a factor of four with every subsequent iteration.

VI.5. 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. An edge whose end points (vertices) are the same is referred to as a *self-loop*. Figure 36 provides a visual representation of the algorithm described below.

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

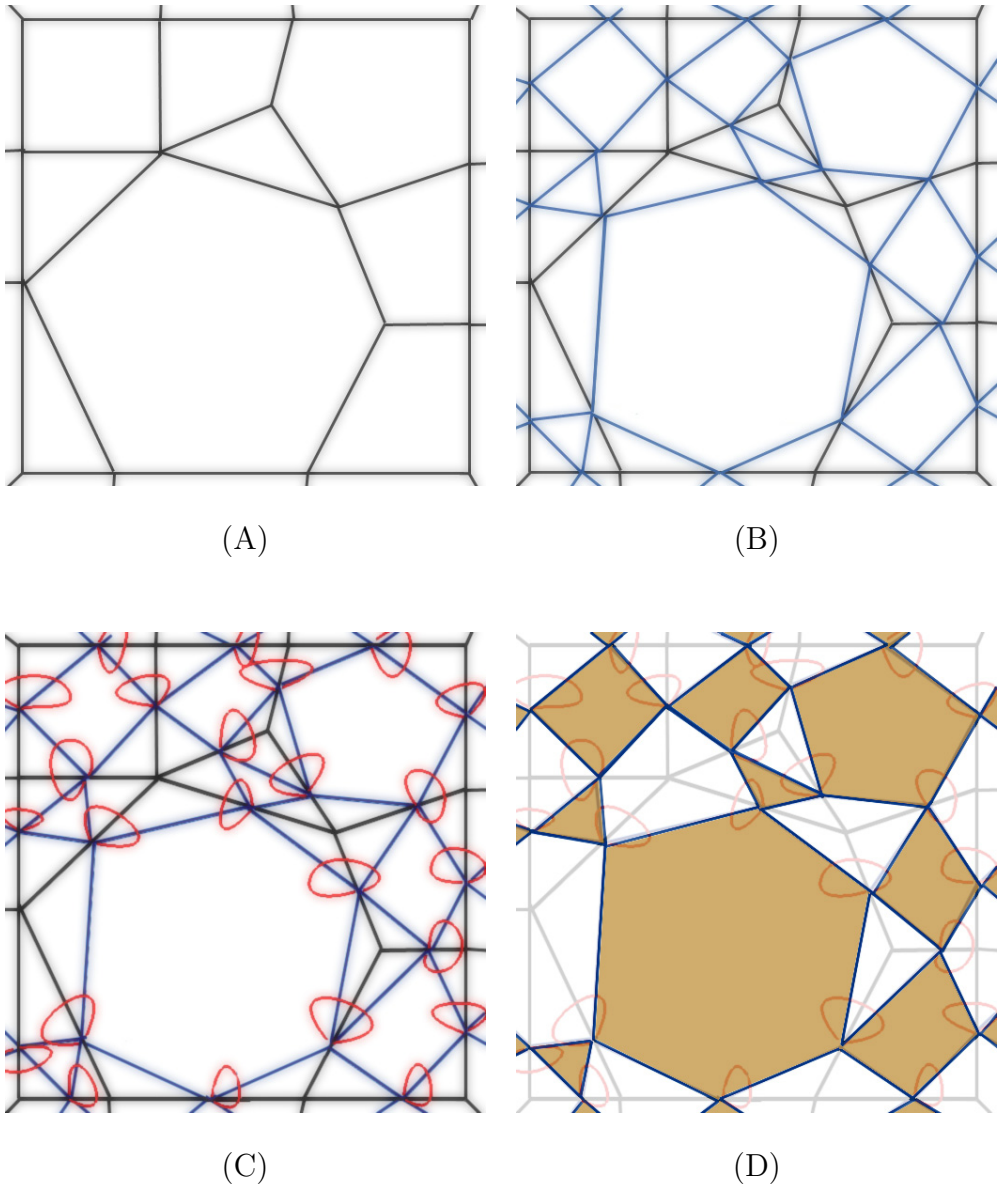


Fig. 36. The Sierpinski subdivision algorithm. (A) is the initial mesh, in (B), each edge is subdivided and midpoints are connected by inserting edges (shown as blue) to create new faces. In (C), an edge (shown as red) is inserted between two corners of each midpoint vertex. The yellow faces are automatically eliminated and new (white) faces are created.

(a) For every corner $c_i = \{a, v_i, b\}$ which points to v_i, insert an edge (shown as blue edges in Figure 36B) 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 36D) 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 36C) 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.

Notice that the back faces (shown as white faces Figure 36.D) are automatically created in this algorithm. Each one of these faces has one self-loop on each one of its vertices. Since (in practice) the self-loop edges have zero length and are not visible, the resulting faces look like they have n number of sides instead of $2n$. For instance a face which looks like a triangle is actually a hexagon as shown in Figure 37.

The self-loops act like boundaries that allow the connection of adjacent tetrahedra. The topological structure of one such connection between two hexagons (that look like triangles) is illustrated in Figure 38. Although the resulting shapes are valid manifolds, they appear to be non-manifold since the self-loops cover zero area.

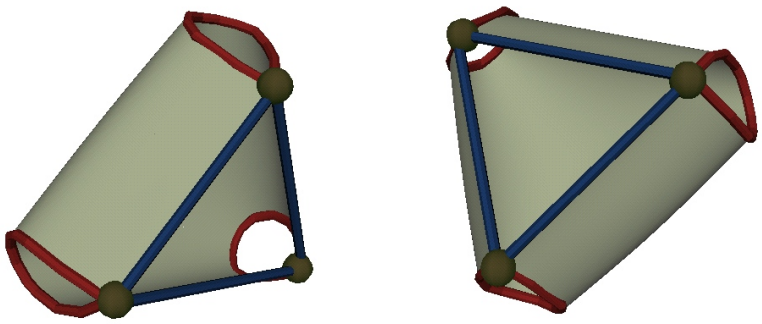


Fig. 37. Two topological renderings of a hexagonal face that looks like a triangle.

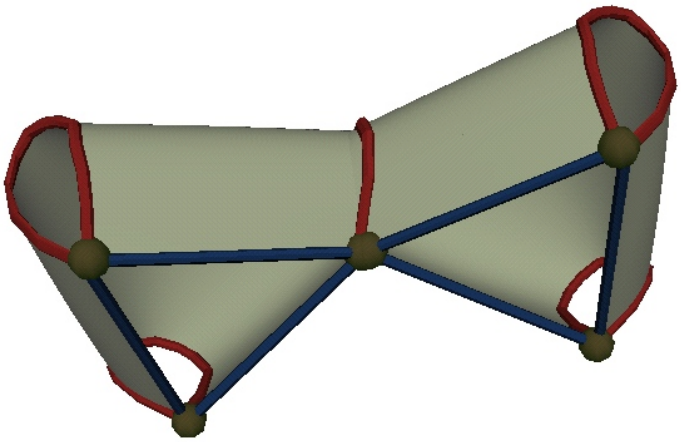
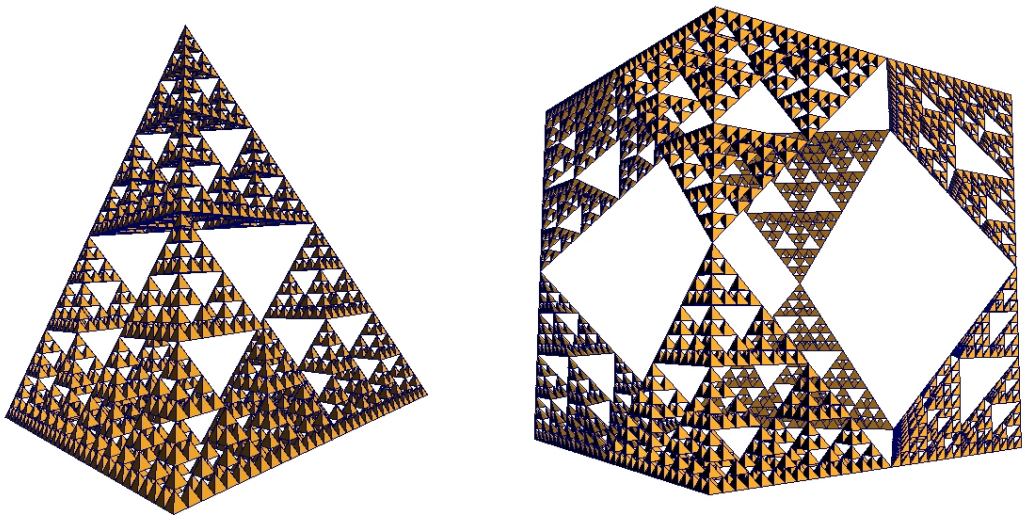


Fig. 38. Connection of the hexagonal faces shown in Figure 37 allows the creation of non-manifold looking manifold structures.

VI.6. Examples

Figure 39 shows two shapes created using the new algorithm, one of which is the familiar Sierpinski tetrahedron.



A B

Fig. 39. Two shapes created using the generalized algorithm. Initial shapes are a tetrahedron (A) and a cube (B).

In the following discussion of the examples, self-loops are ignored, since they are only useful for simplification of the algorithm and are otherwise invisible to viewers. For instance, a hexagon with three self-loops is referred to as a triangle for ease of illustration.

For evaluation of the results, the vertices have been classified into 5 categories. The classification is based on the pyramid created by the straight edges (ignoring self-loops) that share the vertex in question:

1. *Convex vertex.* The tip of the pyramid is convex.

2. *Star vertex*. The tip of the pyramid is star.

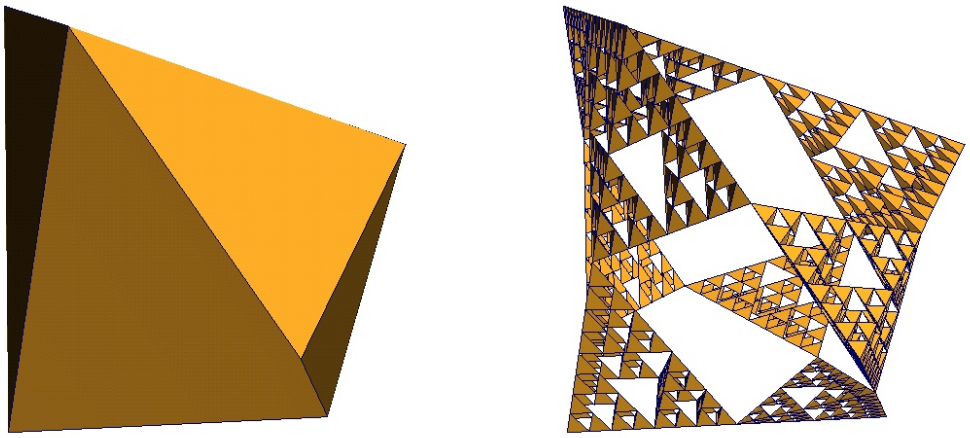
3. *Concave vertex*. The tip of the pyramid is concave.

4. *Planar vertex*. The tip of the pyramid is flattened.

5. *Saddle vertex*. The tip of the pyramid is a saddle point.

Based on this classification, the following cases have been identified.

- If the initial mesh consists of only 3-valence convex vertices such as in a dodecahedron or a cube [41], after the first iteration, the resulting mesh consists of only tetrahedral shapes. Since in a tetrahedral shape, each face is a triangle there is no problem in rendering. Note that the faces of the convex polyhedron with 3-valence vertices do not have to be planar as shown in Figure 40.



A B

Fig. 40. Sierpinski algorithm applied to a shape with only 3-valence convex vertices and non-planar faces. The resulting shape after 4 iterations is shown in B.

- If the initial mesh includes some non-3-valence convex vertices such as in an icosahedron or an octahedron [41], the resulting mesh always includes non-triangular faces. The planarity of these faces depends on the vertex positions in the initial mesh, although non-planar faces become more and more planar with each iteration.

- If the initial mesh includes some star vertices, the resulting mesh always includes star shaped faces as shown in Figure 41. Even if these faces are planar, hardware rendering can sometimes create visual artifacts when they are converted to triangles.

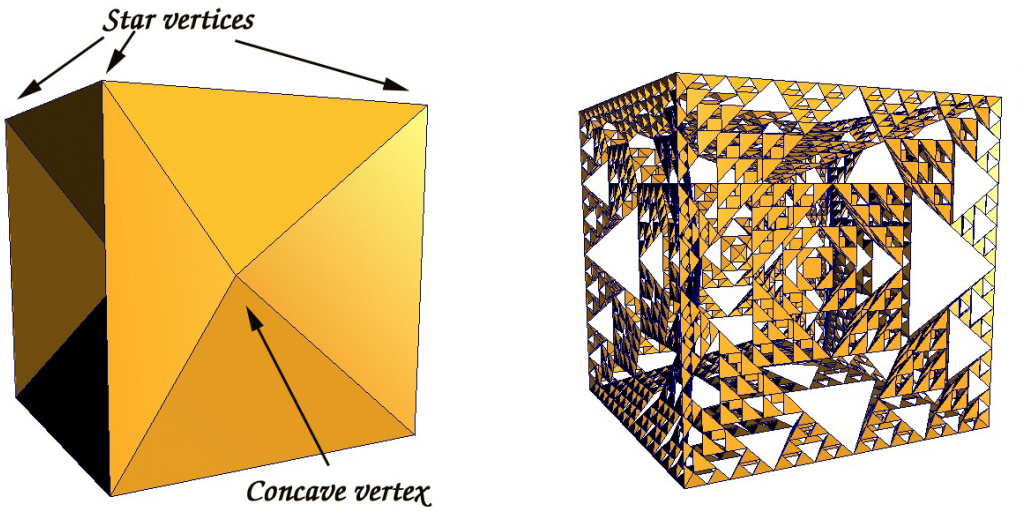
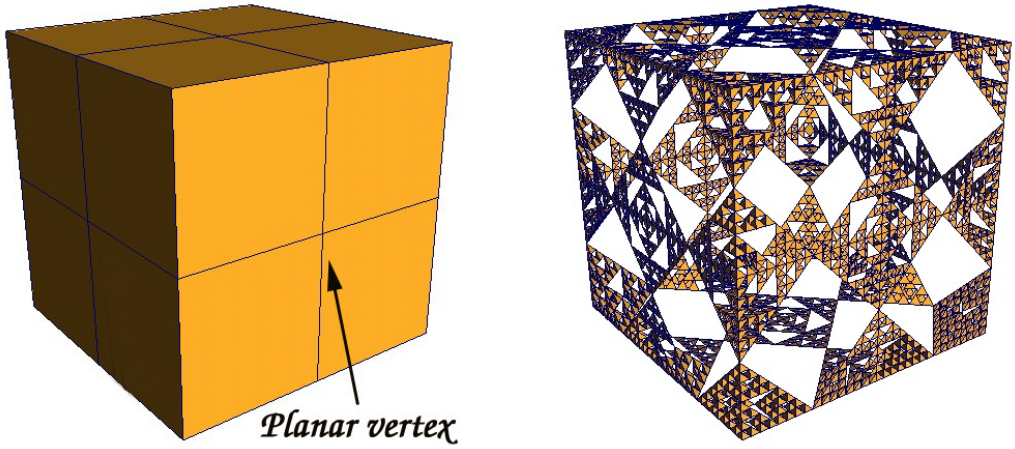


Fig. 41. Generalized Sierpinski algorithm applied to a mesh with star and concave vertices. The resulting shape after 4 iterations is shown in B.

- If the initial mesh includes some concave vertices, each one of these concave vertices creates a geometrically inverted pyramid (Figure 41), i.e. normal vectors points inside of the pyramid instead of outside. This problem is not easily

visible and can be corrected easily by inverting the normals.

- If the initial mesh includes some planar vertices, each one of these planar vertices creates a flattened pyramid, as shown in Figure 42. This problem can be visually annoying but cannot be corrected. The number of flattened pyramids increases with each iteration.



(A) (B)

Fig. 42. Generalized Sierpinski algorithm applied to a mesh with planar vertices. The resulting shape after 4 iterations is shown in B.

- 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 visually less annoying.

The algorithm produces connected and manifold polyhedra. Thus, applying a smoothing subdivision scheme such as Doo-Sabin or Catmull-Clark [12, 10], produces a smoothed shape which remains connected as shown in Figures 43 and 44.

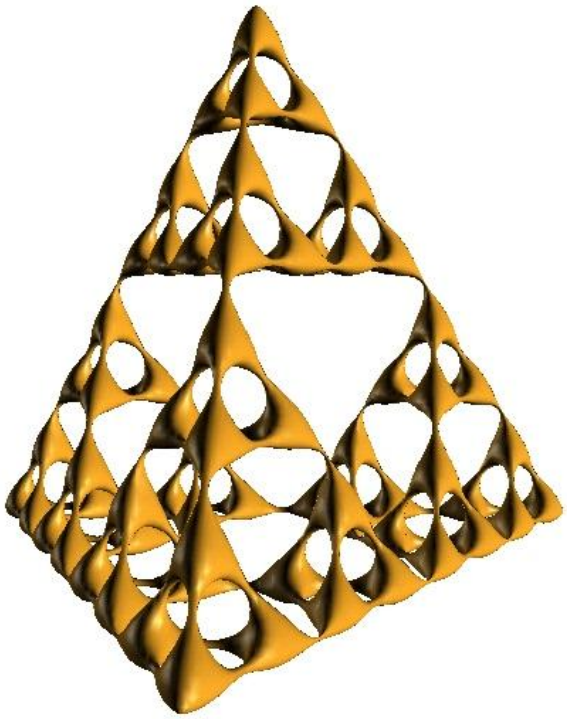


Fig. 43. Smoothed Sierpinski tetrahedron. 3 iterations of Sierpinski subdivision were applied to a tetrahedron and the resulting mesh smoothed using Doo-Sabin subdivision.

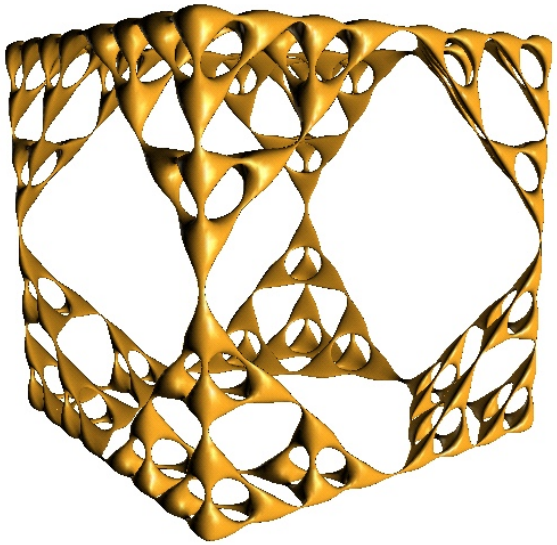


Fig. 44. Smoothed Sierpinski cube. 3 iterations of Sierpinski subdivision were applied to a cube and the resulting mesh smoothed using Doo-Sabin subdivision.

CHAPTER VII

GENERALIZED MENGER SPONGES

The Menger sponge is the three-dimensional analog of the two-dimensional Sierpinski carpet (Figure 45). Both are examples of self-similar fractal shapes as described in the previous section. Schemes for their construction are also similar to those that were described earlier (page 55).

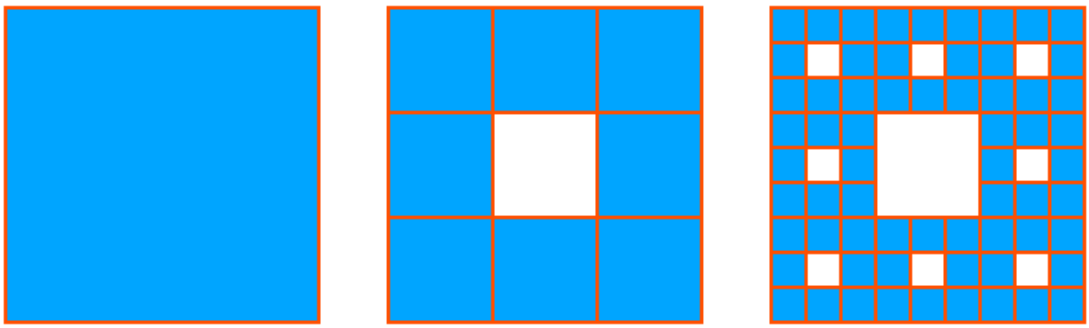


Fig. 45. The Sierpinski carpet. The sequence shows the construction up to 2 iterations using the set difference operation.

In this section, a scheme to construct generalized Menger sponge-type shapes is presented. For brevity, such shapes will be referred to as *generalized Menger sponges*. Along the lines of the algorithm for generalized Sierpinski polyhedra described earlier, the algorithm described here produces connected and manifold polyhedra and can be applied to any manifold polyhedron.

VII.1. Current construction approaches

The Menger sponge can be constructed using an algorithm that exploits the self-similarity property. For the Sierpinski carpet and the Menger sponge, schemes based

on set operations are common. An example of such an algorithm for creating the Menger sponge, starting from a cube is given below.

1. Start with any given shape.
2. Duplicate the shape 27 times and translate the copies to form a $3 \times 3 \times 3$ cube.
3. Discard the copies at the center of each side as well as the one at the center of the cube..
4. Take the union of the remaining objects.
5. Repeat the above steps with the new object.

In most implementations, the union is achieved simply by placing the shapes next to each other, which produces a set of disconnected polyhedra as shown in Figure 46 where the initial shape is a cube. The algorithm always produces the same limit shape (cubic Menger sponge) regardless of the initial shape.

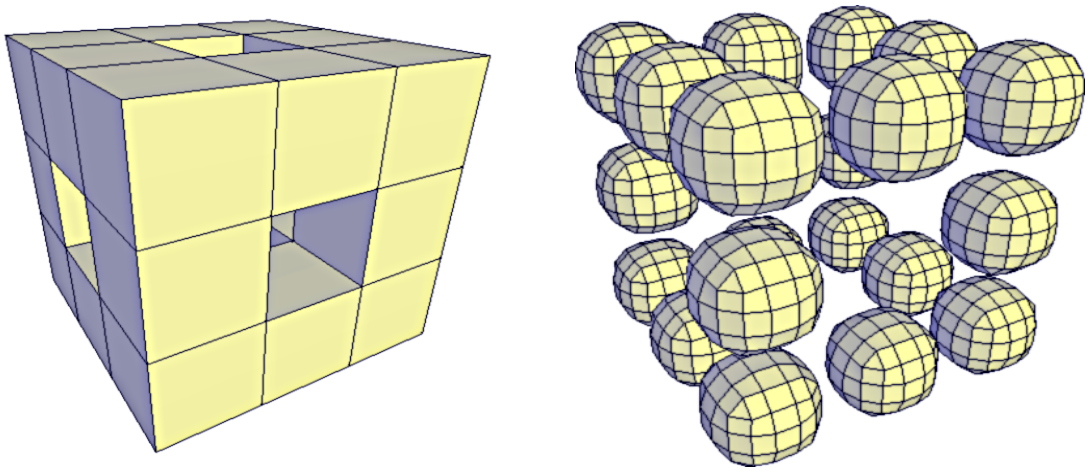


Fig. 46. Menger sponge after 1 iteration. The image on the right shows the mesh after applying a subdivision scheme.

VII.2. Construction approach based on set difference

In this section, a different approach to the construction of the Menger sponge is described. The algorithm uses a remeshing scheme, similar to subdivision schemes, but, unlike subdivision schemes, changes the topology of the initial mesh. The algorithm is similar to those based on set difference. However, it uses polygonal mesh modeling operators and can be easily generalized to any initial shape. It also produces connected and manifold polyhedra.

An overview of the algorithm for the case where the initial shape is a cube is given below.

1. Subdivide each face of the cube into 9 equal size squares.

2. Duplicate the center square in each face and offset it by a distance equal to one-third of the edge length of the cube, towards the interior of the cube.

3. Combine the offset faces to form a new cube at the center of the original cube. Once the inner cube has been created, reverse its normals to produce a correctly oriented object. The new cube will have an edge length which is one-third of the edge length of the original cube and every face of the new cube corresponds to a center face on the original cube.

4. Using the CreatePipe operator connect each center face on the original cube with its corresponding face in the inner cube.

Figure 47 illustrates the above algorithm. Note that there are no cubes in the resulting mesh. Obviously the above algorithm, as described, cannot be re-applied to the new mesh. Fortunately, the actual implementation of the algorithm does allow it to be applied recursively as will be shown in the next section. It also allows the algorithm to be generalized to initial shapes which are not cubes.

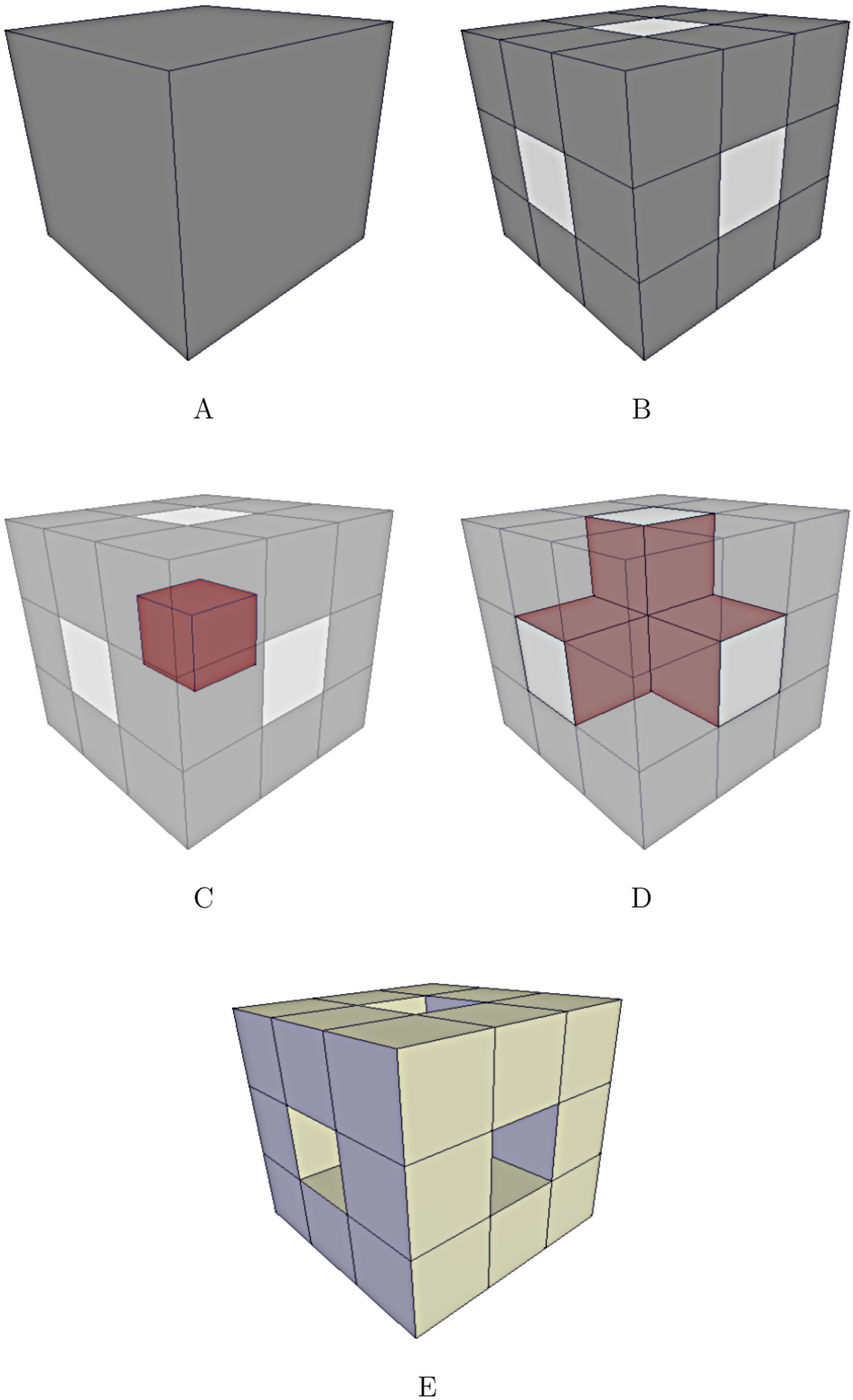


Fig. 47. Menger sponge algorithm based on set difference.

VII.3. Generalized Menger sponge algorithm

Let V be the list of vertices, E the list of edges and F the list of faces in the original mesh. Let D be a thickness parameter specified by the user.

For simplicity, all faces are assumed to be convex. We shall also assume that the initial shape to which the algorithm is applied is a convex polyhedron (it is not restricted to a cube). The algorithm proceeds as follows:

1. For every face f_i in F, create a new face pair $(f'_{i,f}, f'_{i,b})$ as follows:

   Let $(p_0, p_1, \ldots, p_{N-1})$ be the coordinates of the vertices in f_i, where N is the number of vertices in f_i. Let $(p'_0, p'_1, \ldots, p'_{N-1})$ be the coordinates of the vertices in the new faces. The new face pair will be referred to as the *remeshing face pair* since the faces will be used for remeshing the original faces in a later step. For simplicity, the modulus operator has been omitted when referring to the vertex index in the following steps.

   (a) for $j = 0$ to $N - 1$ do

      i. Compute two unit edge vectors originating at p_j and pointing towards the two vertices adjacent to p_j in f_i.

      $\vec{e}_1 = \frac{p_{j+1} - p_j}{|p_{j+1} - p_j|}$

      $\vec{e}_2 = \frac{p_{j-1} - p_j}{|p_{j-1} - p_j|}$

      ii. $p'_j = p_j + D(\vec{e}_1 + \vec{e}_2)$.

   (b) Create the remeshing face pair using the CreateFaceManifold operator and the coordinates computed above.

   $(f'_{i,f}, f'_{i,b}) = \text{CreateFaceManifold}(p'_0, p'_1, \ldots, p'_{N-1})$.

   The remeshing faces will be similar in shape to the original face f_i and smaller.

The *front* face, $f_{i,f}$, points outward in the same direction as f_i and the *back* face, $f_{i,b}$, points into the object opposite to that of f_i.

2. For every remeshing face pair $(f'_{i,f}, f'_{i,b})$ created above, create an *offset face pair* $(f''_{i,f}, f''_{i,b})$ as follows:

   Let $(p''_0, p''_1, \ldots, p''_{N-1})$ be the coordinates of the points in the offset faces. Let $\vec{n}_i$ be the average unit normal vector for face f_i.

   (a) for $j = 0$ to $N - 1$ do

   $$p''_j = p'_j - D\vec{n}_i.$$

   (b) $(f''_{i,f}, f''_{i,b}) = \textsc{CreateFaceManifold}(p''_0, p''_1, \ldots, p''_{N-1})$.

   The *offset* faces will be identical to the *remeshing* faces but offset towards the interior of the object. $f''_{i,f}$ points outward, similar to $f'_{i,f}$ and f_i, while $f''_{i,b}$ points into the object, similar to $f'_{i,b}$.

   Every face f_i in the original mesh is now associated with four faces: the remeshing face pair, $(f'_{i,f}, f'_{i,b})$ and the offset face pair $(f''_{i,f}, f''_{i,b})$. Every edge in the original mesh has two matching edges (corresponding to the two faces it borders) among the remeshing faces as well as two matching edges among the offset faces.

3. Combine the offset faces into a single surface as follows:

   (a) For every edge e_i in E

      i. Find the two matching edges among the edges that make up the offset faces. Let the matching edges be e''_1 and e''_2.

      ii. Each edge has two faces adjacent to itself. In this case, among the two faces adjacent to e''_1 and e''_2, one points inwards and the other points

outwards. Let the *outward* pointing faces adjacent to e_1'' and e_2'' be f_1^O and f_2^O respectively.

iii. ConnectEdges($e_1'', f_1^O, e_2'', f_2^O$).

Note that the *outward* pointing faces are chosen for the ConnectEdges operation. This will produce a surface that points *inwards* in relation to the original surface, thus creating an orientable 2-manifold.

Also note that e_1'' and e_2'' may geometrically be in the same position. This depends on the angle that the two faces adjacent to e_i in the original mesh make with each other. If the angle is 90 degrees the two edges will be coincident, otherwise they will be geometrically separate.

(b) Collapse edges that were inserted by the ConnectEdges operation in the previous step, if necessary.

If the edge to be collapsed is a self-loop, the edge is simply removed from the mesh and there is no need for any vertices to be merged. This operation reduces the number of edges in the mesh by one. If the edge was not a self-loop, the number of vertices in the mesh also decreases by one.

The criteria for determining if an edge has to be collapsed are discussed in Section VII.4.

At this stage, only the inward pointing offset faces $f_{i,b}''$ will remain.

4. Remesh the original faces (the outer surface). This will make use of the remeshing faces created in Step 1.

(a) For every edge e_i in E

i. Subdivide e_i into three parts.

ii. Adjust the coordinates of the two new points created, such that they are at a distance D (the thickness parameter) from the ends of the edge.

At the completion of the above steps, every edge in the original mesh would have been trisected into three parts, with the length of the end segments equal to the specified thickness parameter D.

(b) For every face f_i in F

Let E_{f_i} be the list of edges in f_i.

For every edge e_j in E_{f_i}

i. If e_j is the middle segment of an original edge

A. Find the remeshing face pair $(f'_{i,f}, f'_{i,b})$ corresponding to f_i.

B. Find the edge e'_j in the remeshing face pair which corresponds to the original edge in f_i of which e_j is the middle segment.

C. ConnectEdges$(e_j, f_i, e'_j, f'_{i,b})$.

Note that the *inward* pointing remeshing face is used for the ConnectEdges operation. This ensures that we create a correctly oriented 2-manifold. This step subdivides every face of the original mesh into $2k$ quadrilaterals, where k is the number of vertices in the original face, as well as a central face which has the same number of vertices as the original face.

At this stage, only the outward pointing remeshing face $f'_{i,f}$ will remain.

5. Connect each outward pointing remeshing face $f'_{i,f}$ to the corresponding inward pointing offset face $f''_{i,b}$.

For every face f_i in F

(a) Find the remeshing face $f'_{i,f}$ and offset face $f''_{i,b}$ corresponding to f_i.

(b) Find one pair of matching corners c'_i and c''_i in $f'_{i,f}$ and $f''_{i,b}$ respectively. The matching corners are easily obtained because of the face correspondence established in Step 2.

(c) CreatePipe(c'_i, c''_i).

VII.4. Conditions for edge collapse

All the edges which were inserted in Step 3a are candidates for being collapsed. However, in some situations collapsing the edge will cause self-intersections and in some situations the edge will have to be collapsed to *avoid* self-intersections. In certain cases, the decision can be left to the user or be made using some local metric of the mesh.

Let e''_{12} be the edge under consideration. Let f''_1 and f''_2 be the two offset faces between which e''_{12} was inserted. Further, let f_1 and f_2 be the faces in the original mesh corresponding to f''_1 and f''_2 respectively and let e_{12} be the edge adjacent to both f_1 and f_2. Let ϕ be the interior angle between f_1 and f_2 at each corner of their intersection along e_{12}.

Figure 48 illustrates the various entities under consideration. Remeshing faces are not shown in this figure. A cross-sectional view along the blue edge, as shown in Figure 49 will be used to explain the various edge collapse situations.

The decision on whether an edge is to be collapsed or not is made based on the value of ϕ. The possible scenarios and the corresponding decisions are described below.

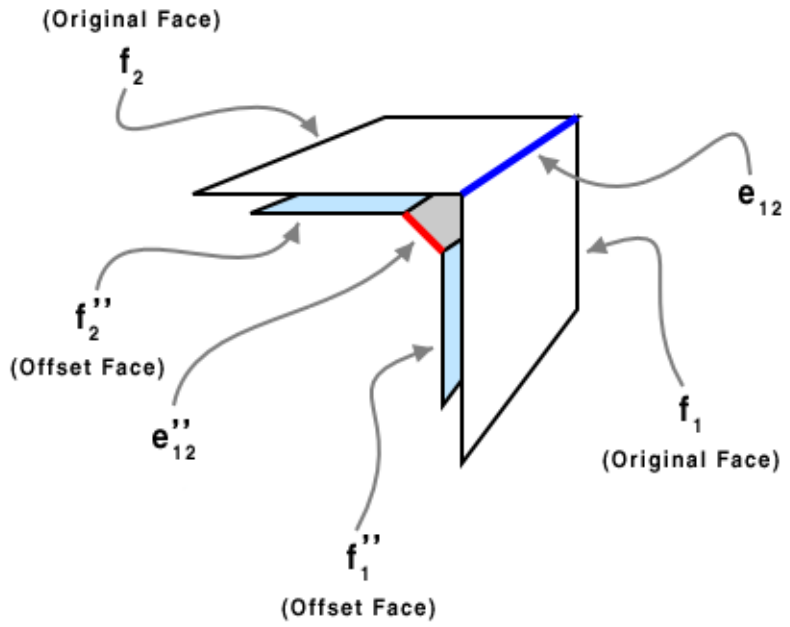


Fig. 48. Entities used for determining edge collapse conditions.

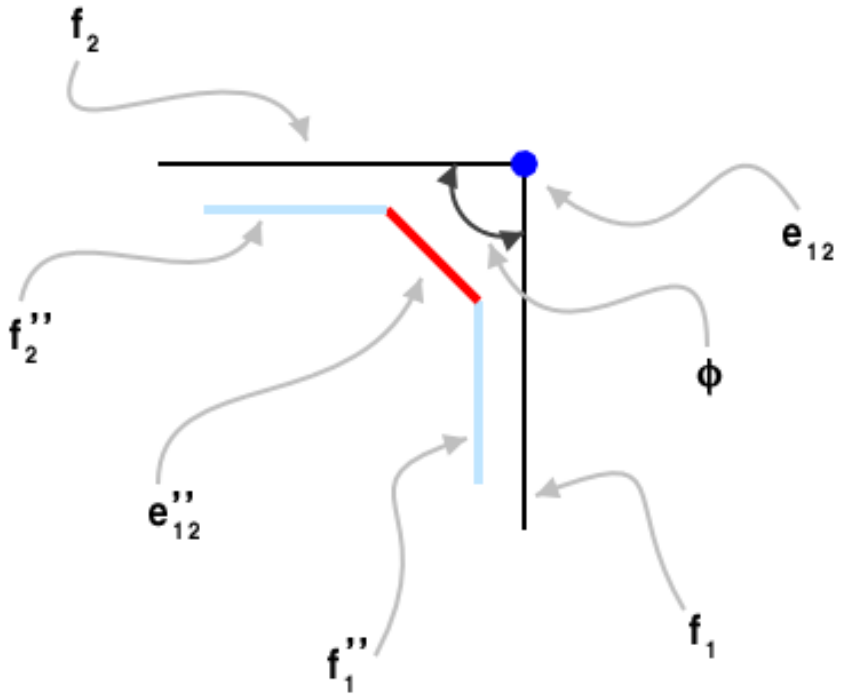


Fig. 49. Annotated cross-sectional view used for explaining edge collapse situations.

1. $\phi < 90°$.

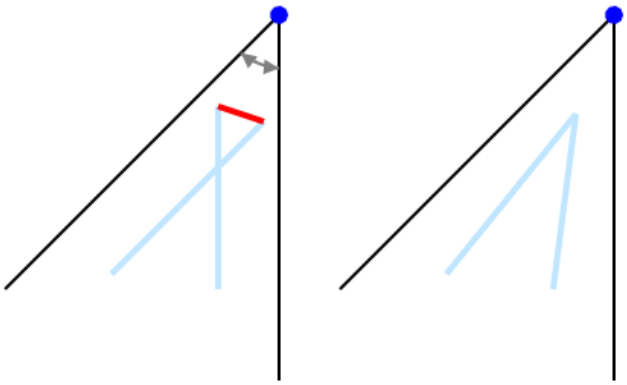


Fig. 50. Conditions for edge collapse, case 1: $\phi < 90°$.

In this situation f_1'' and f_2'' intersect causing a self-intersection in the interior surface. Collapsing e_{12}'' will eliminate this self-intersection, as shown in Figure 50.

2. $\phi = 90°$.

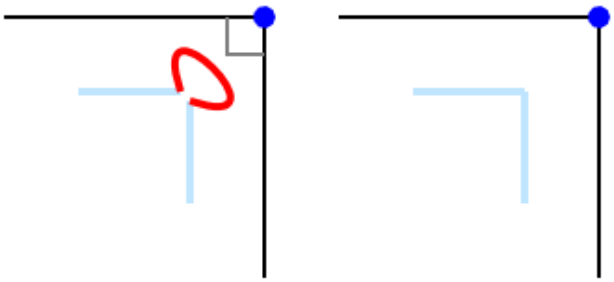


Fig. 51. Conditions for edge collapse, case 2: $\phi = 90°$.

In this case (Figure 51), f_1'' and f_2'' intersect at one of their edges. The end points of e_{12}'' will be geometrically identical, although they are shown as being geometrically distinct to emphasize that e_{12}'' it is not a self-loop. Since e_{12}'' does not add any visual or topological information to the mesh, it should always be

collapsed.

3. $90° < \phi < 180°$.

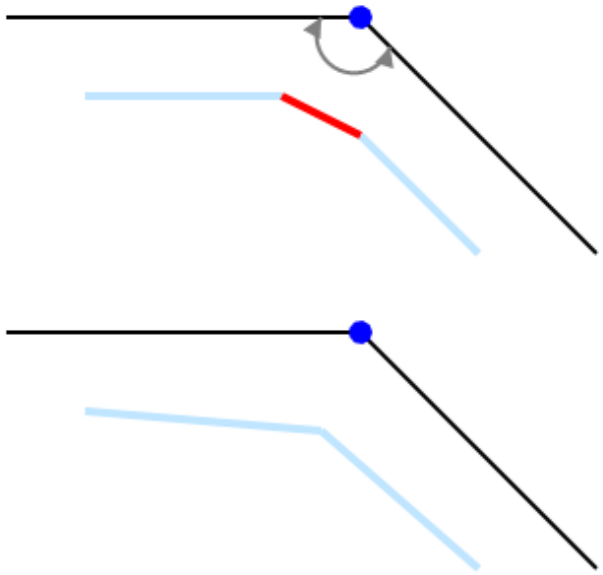


Fig. 52. Conditions for edge collapse, case 3: $90° < \phi < 180°$.

In this situation f_1'' and f_2'' do not intersect as shown in Figure 52. e_{12}'' does not have to be collapsed. However, in some situations it might be desirable to collapse the edge. The decision can be based on the length of e_{12}''. Or equivalently it can be based on the angle which the end points of e_{12}'' subtend at e_{12}, which is directly related to the angle between f_1 and f_2. The decision can be based on either of these criteria, with the end user deciding the threshold length or angle.

4. $\phi = 180°$.

f_1'' and f_2'' are co-planar as shown in Figure 53. As in the previous case, f_1'' and f_2'' do not intersect. In this situation, e_{12}'' does not have to be collapsed. In

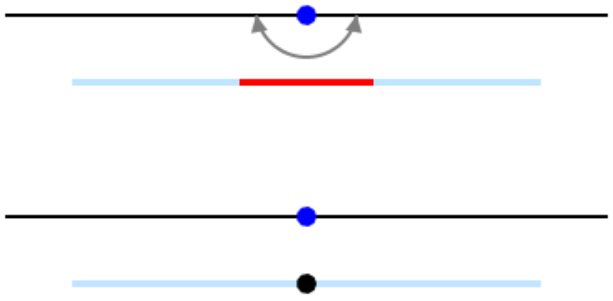


Fig. 53. Conditions for edge collapse, case 4: $\phi = 180°$.

most cases it is desirable to not collapse e''_{12}. Specifically for creation of a cubic Menger sponge e''_{12} should not be collapsed.

5. $180° < \phi < 270°$.

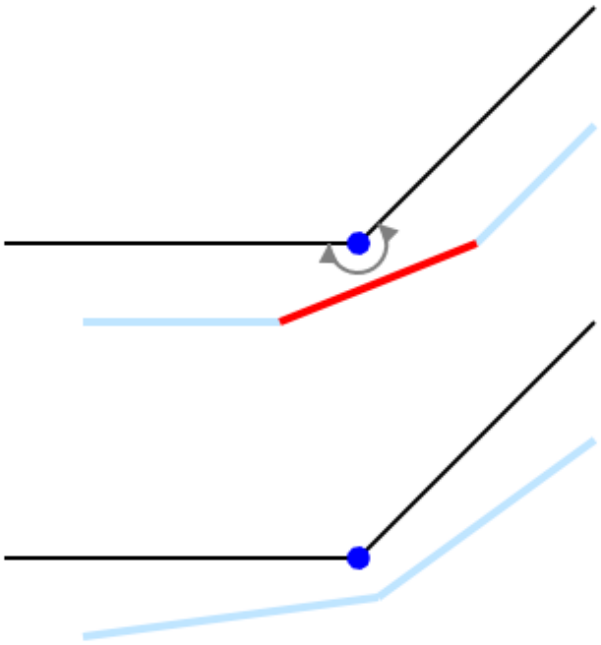


Fig. 54. Conditions for edge collapse, case 5: $180° < \phi < 270°$.

In this situation e_{12} is a non-convex edge. Although f''_1 and f''_2 do not intersect, the distance of the offset surface from the original surface in the region becomes

smaller than the specified thickness as can be seen in Figure 54. Collapsing the edge only exaggerates this non-uniformity. The problem can be fixed by inserting a new vertex in e''_{12} as described later.

6. $\phi \geq 270°$.

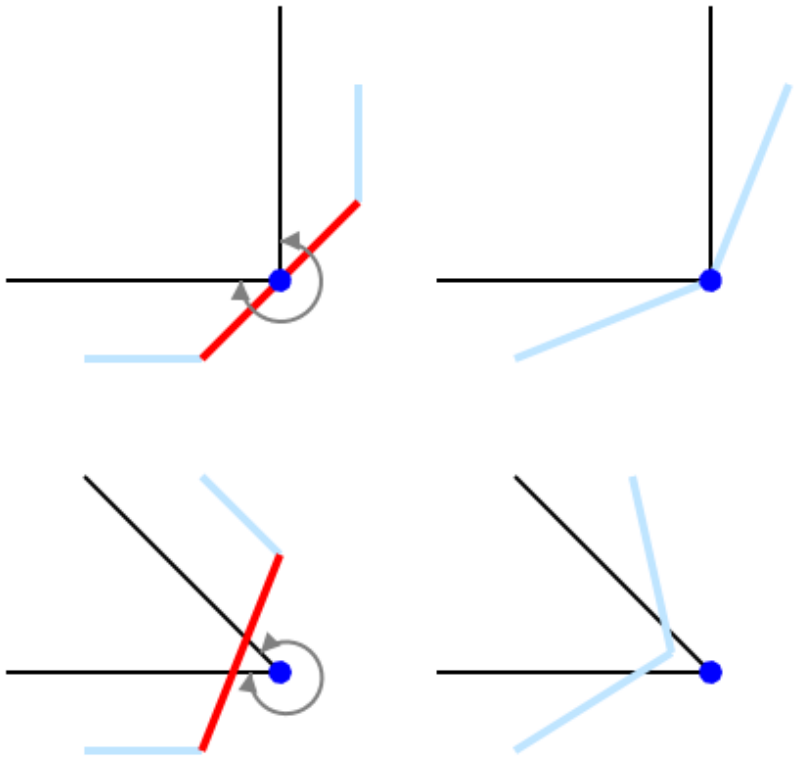


Fig. 55. Conditions for edge collapse, case 6: $\phi \geq 270°$.

This is similar to the previous situation, except that e_{12} now intersects the outer surface, as can be seen in Figure 55. Collapsing the edge makes the intersection more pronounced. This problem can also be fixed by introducing a new point.

VII.5. Special cases

The above algorithm is, in general, applicable to most polyhedral meshes. However, the algorithm, as described, does not work well in some situations. Some modifications in the algorithm are required to produce acceptable results in those situations.

VII.5.1. Non-planar faces

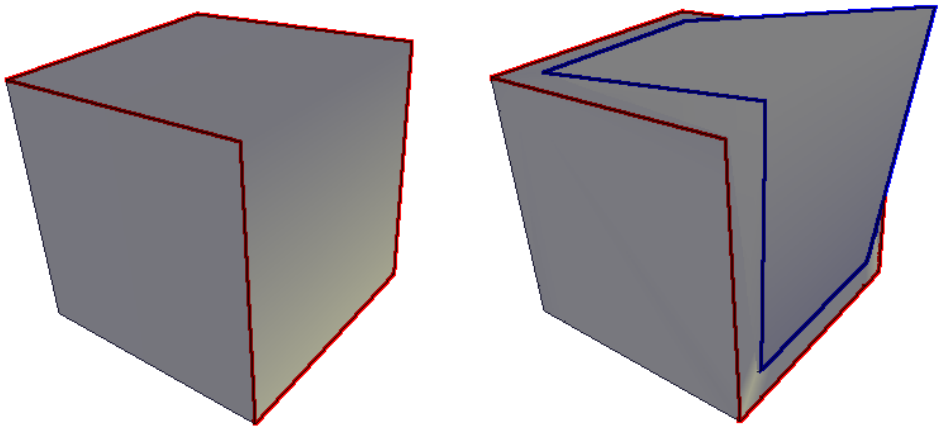


Fig. 56. Problem of creating the remeshing face for non-planar polygons.

Figure 56 illustrates the problem of creating the remeshing face for a non-planar face. The face shown in red color is a non-planar face and the face shown in blue color is the remeshing face created by the algorithm. The problem does not have any simple solution. However this is not really a restriction, since in most applications we rarely come across such faces. Even when we have faces that are non-planar, the skew is usually very minimal and the above algorithm still produces acceptable results. Moreover, non-planar faces can always be triangulated to make them planar.

VII.5.2. Non-convex polygons

In the above algorithm, all faces are assumed to be convex polygons. This greatly simplifies the calculation of the remeshing face from the original face. However, for non-convex polygons, the algorithm produces a remeshing face that intersects the original face as shown in Figure 57.

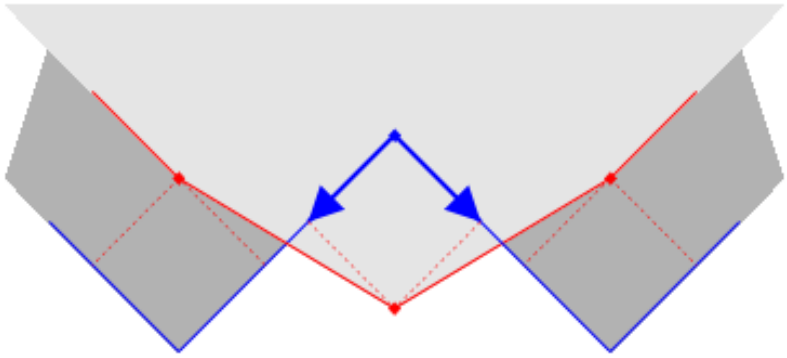


Fig. 57. Problem of creating the remeshing face for non-convex polygons.

If the non-convex corners of the face can be identified, the problem is easily handled. A convex corner is one where the two edges incident at the corner subtend an interior angle less than 180 degrees. All other corners are non-convex corners. For such corners, we merely have to reverse the unit edge vectors used (blue arrows in Figure 57) to compute v'_j from v_j in step 1 of the algorithm.

Thus the step for the computation of v'_j (page 72) is modified as follows:

1. (a) iii. If v_j is a non-convex corner of f_i

$$v'_j = v_j - D\vec{v}_{o,j}$$

else

$$v'_j = v_j + D\vec{v}_{o,j}$$

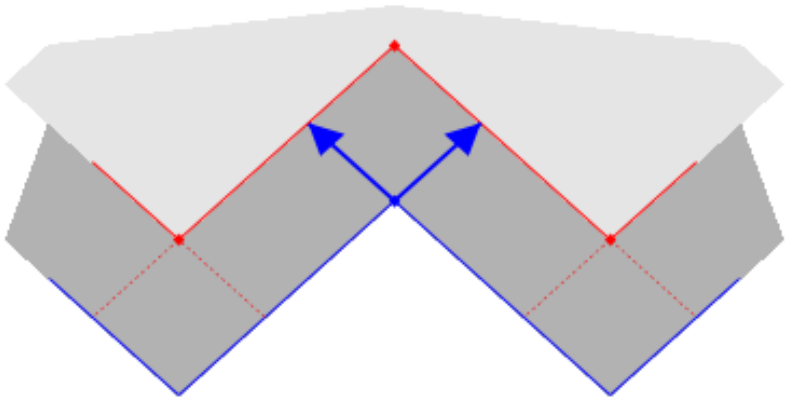


Fig. 58. Corrected remeshing face for non-convex polygons.

With the modification, the remeshing polygon is computed as shown in Figure 58.

VII.5.3. *Winged* corners

There is a special case of a non-convex corner for which the above modification is not sufficient. This is the situation when the angle between the two edges incident at the corner is exactly 180 degrees. That is, the two edges are co-linear. Such corners will be referred to as *winged* corners. For a winged corner, the offset vector $\vec{v}_{o,j}$ computed when creating the remeshing face becomes the zero vector. Figure 59 illustrates the problem of creating the remeshing face for a face with such a corner.

The offset vector has to be computed differently for a winged corner. The offset vector should move the vertex towards the interior of the face. The normal vector to the face can be used to compute the offset vector. The following modification to the algorithm will allow it to handle this special case.

1. (a) ii. Compute the offset vector $\vec{v}_{o,j}$.

   Let $\vec{n}_i$ denote the unit normal to the face f_i

   If v_j is a winged corner

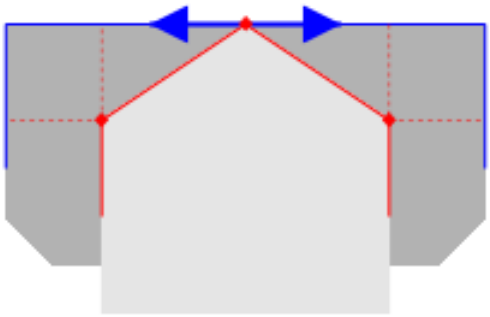


Fig. 59. Problem of creating the remeshing face for a polygon with a winged corner.

$$\vec{v}_{o,j} = \vec{n}_i \times \vec{e}_1$$

else

$$\vec{v}_{o,j} = \vec{e}_1 + \vec{e}_2$$

The $\times$ symbol represents the cross product between two vectors. For a winged corner, the offset vector is essentially the cross product between the unit normal to the face and the unit edge vector originating at the corner and pointing towards the *next* vertex in the face. The existence of a consistent rotation system for each face is implicitly assumed.

With the modification the remeshing face for a face with a winged corner is computed as shown in Figure 60.

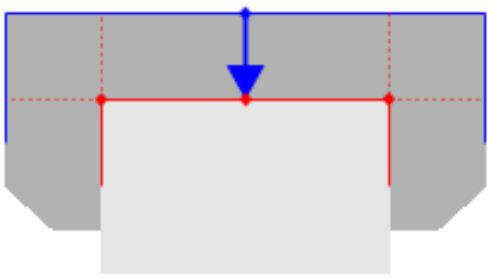


Fig. 60. Computing the remeshing face for a polygon with a winged corner.

VII.5.4. Non-convex edges

As mentioned in Section VII.4 above, non-convex edges can cause self-intersections when creating the offset faces for the inner surface (see Figure 55 on page 81). If the interior angle ϕ is exactly 270° the intersection happens along the edge e_{12} in the original mesh (outer surface) and will not be visually evident, although subsequent operations are likely to introduce visual artifacts. If $\phi > 270°$ the intersection is more pronounced. In both situations, collapsing the edge e''_{12} does not help and in the latter case it actually exacerbates the problem as can be seen from Figure 55.

The intersection can be eliminated by inserting two new vertices (for each end of the edge e''_{12}) as shown in Figure 61. In the figure the black dots represent the new vertices and the green lines are the edges which replace e''_{12} (red lines). A new edge also has to be inserted between the two new vertices (this will be perpendicular to the plane of the paper along the black dot). Implementing this fix is not straightforward and complicates the algorithm, especially when collapsing edges, and has therefore been ignored in the present work.

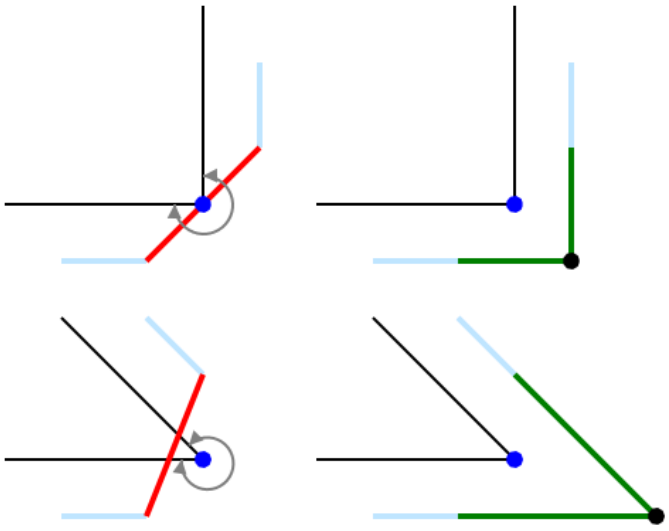


Fig. 61. Avoiding self-intersections for non-convex edges.

VII.6. Examples

Figure 62 shows the result of applying the generalized Menger sponge algorithm twice to a cube, using a thickness parameter that is one-third of the edge length at each stage. The result outwardly resembles the classic Menger sponge after two iterations, although as explained above, the two are not identical in the interior regions.

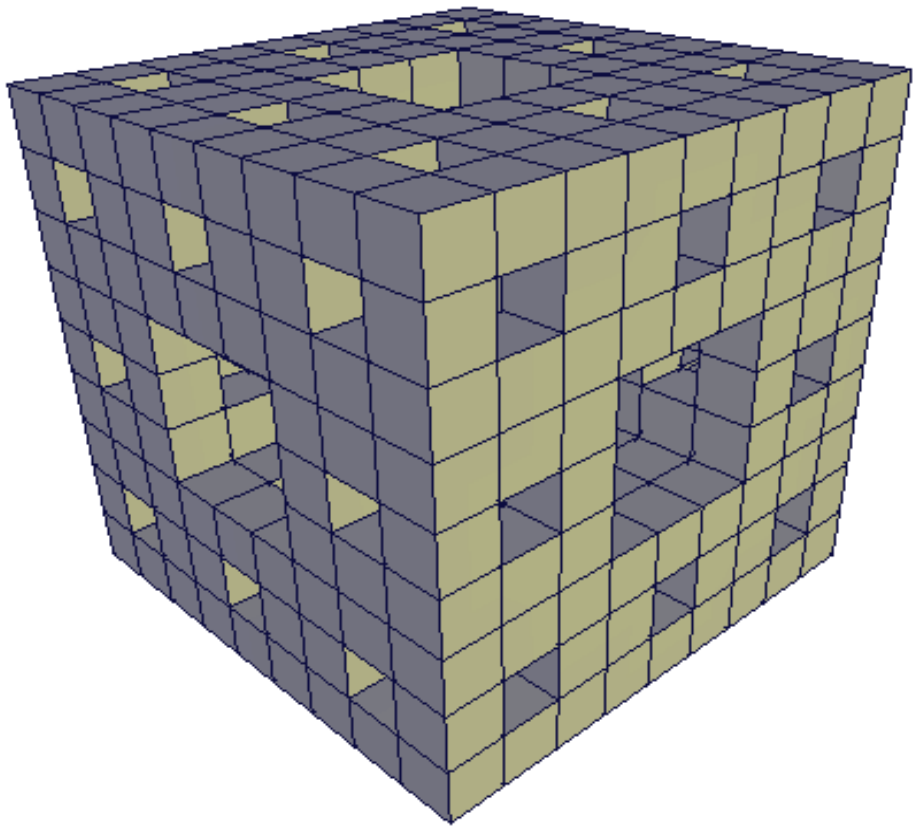


Fig. 62. Generalized Menger sponge algorithm applied to a cube. The thickness parameter is equal to one-third of the edge length.

In Figure 63, the starting shape is still a cube, but a different thickness parameter has been used. Figure 64 shows two examples where the algorithm has been applied to non-cubic shapes. The initial shapes are a dodecahedron and a tetrahedron. The image also shows the shapes after application of a subdivision scheme.

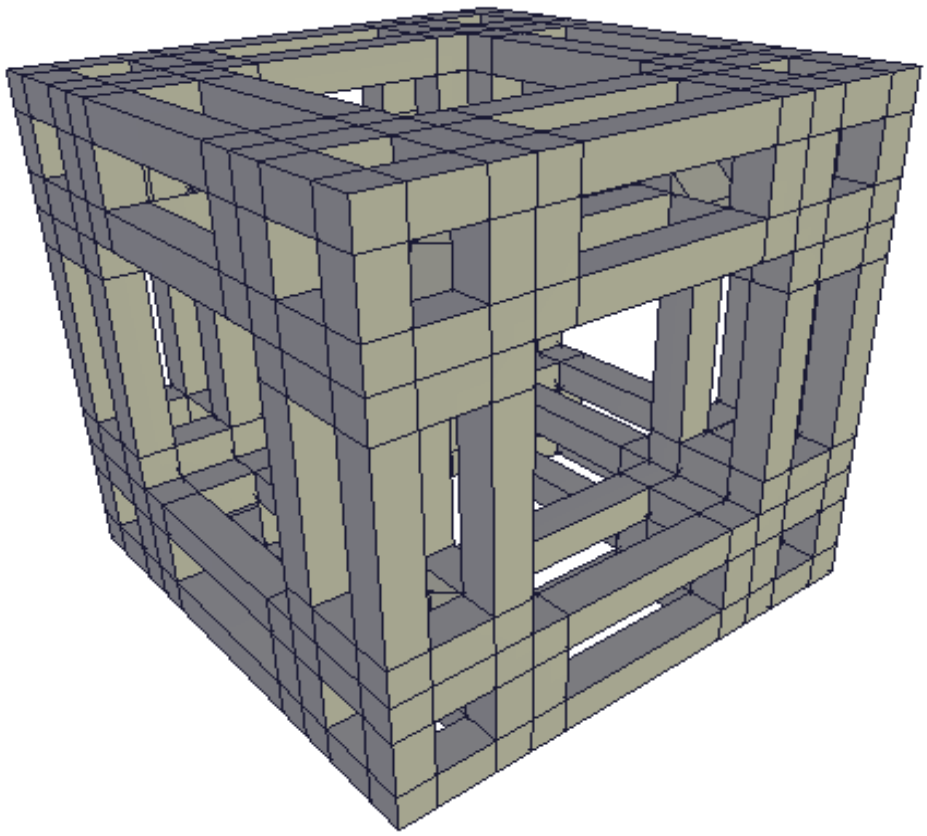


Fig. 63. Menger sponge example with a different thickness parameter. The starting shape is a cube.

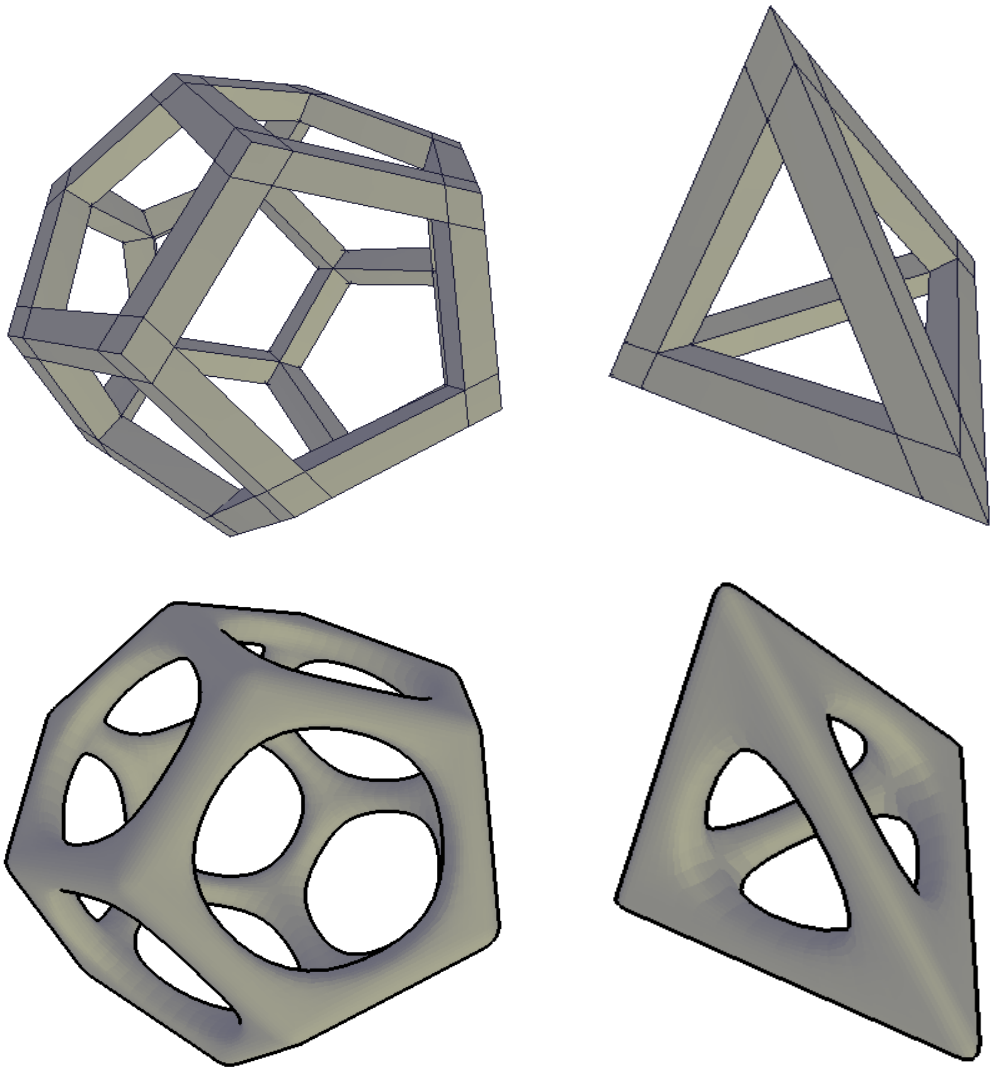


Fig. 64. Generalized Menger sponge algorithm applied to non-cubic shapes. Initial shape on the left is a dodecahedron, on the right a tetrahedron.

CHAPTER VIII

CONCLUSIONS

VIII.1. Summary

The primary focus of this research was on developing tools for high genus mesh modeling. The inspiration for this subject came from art and was pushed further along by mathematics. In Chapter II basic topological concepts, data structures and fundamental mesh modeling operators were explored. Chapter III built on the fundamental operators to present several high-level operators for topological mesh modeling. Maintaining topological consistency of the models has been an important consideration in the development of all the tools.

In Chapter IV we looked at the first of four tools for high genus mesh modeling, namely the creation of multi-segment curved handles. Chapter V presented a tool for creating high genus rind shapes. Both of these interactive tools allow the user to create artistic and functional high genus models.

Chapters VI and VII introduced two automatic approaches to creating high genus models, namely, creation of generalized Sierpinski polyhedra and generalized Menger sponges, respectively. Both tools derive inspiration from fractal geometry and allow the user to easily create very high genus models.

Examples of models created using these tools were also presented in each chapter, along with a discussion of their limitations and possibilities for improvements.

A summary of the modeling tools developed and a few examples of their usage follows.

VIII.2. High-genus mesh modeling tools

The four tools developed in this research can be grouped into two categories – interactive tools and automatic tools. Interestingly, the first set of tools were inspired by art forms, while mathematics provided the inspiration for the tools in the second set.

The tool for creating multi-segment curved handles is the first one in the interactive category. It allows the creation of handles between two faces of a manifold surface. A Hermitian curve is used to define the shape of the handle. The user can control the shape by adjusting the weights of normals used in the Hermitian equation as well as the number of segments in the handle. The tool also allows the user to introduce twists in the handle by appropriate selection of corners in the two faces.

The rind modeling tool allows the creation of high genus rind shapes. Considering the complexity of the models that can be created, the implementation and operation of this tool is surprisingly simple. The user can control the thickness of the rind and punches holes in the rind by merely selecting faces in the mesh. By punching holes in adjacent faces, the user can also created models that look like peeled rinds. Although the tool was developed for high genus modeling, low or zero genus surfaces which resemble rinds can also be created.

Both of the interactive tools can be used in succession to create functional models such as the cup shown in Figure 65. The body of the cup was first created using rind modeling and the handle was added using the curved handle creation tool.

The generalized Sierpinski polyhedra tool allows the creation of very high genus polyhedra that are similar to the Sierpinski tetrahedron. The Menger sponge tool creates generalized Menger sponge-type polyhedra of high genus. In contrast to several existing approaches, both tools produce connected and manifold polyhedra and can be applied to any initial shape. Figure 66 shows an example of an object created

Fig. 65. Model of a cup created using both rind modeling and multi-segment curved handle tools.

using both of these tools in succession.

Figure 67 shows an example of an object created using a combination of the automatic and interactive tools.

VIII.3. Implementation details

The algorithms presented in this research were developed using the Doubly Linked Face List data structure and the minimal set of fundamental operators introduced in Chapter II. The algorithms were implemented in C++, using OpenGL for the graphics and the freely available GUI toolkit, FLTK [16] for the user interface. All code development was done on SGI IRIX and Linux workstations.

Most of the images in this document are screen-shots from the program. Some images were rendered using the 3D modeling, animation and rendering package, Maya. For this the models are exported as Wavefront object files from the program and imported into Maya. Post-processing on the images was done using Adobe Photoshop and GIMP. Photographs of the sculptures and other objects were taken by the author.

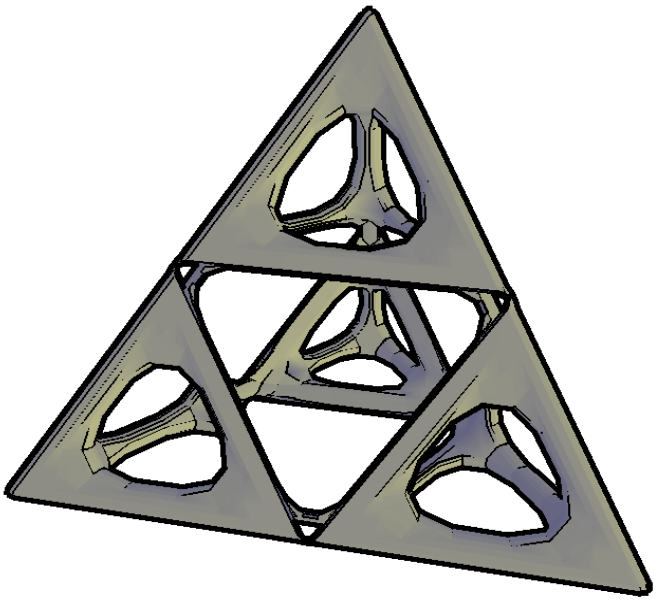


Fig. 66. Model created using a combination of the generalized Sierpinski tool and the generalized Menger sponge tool.

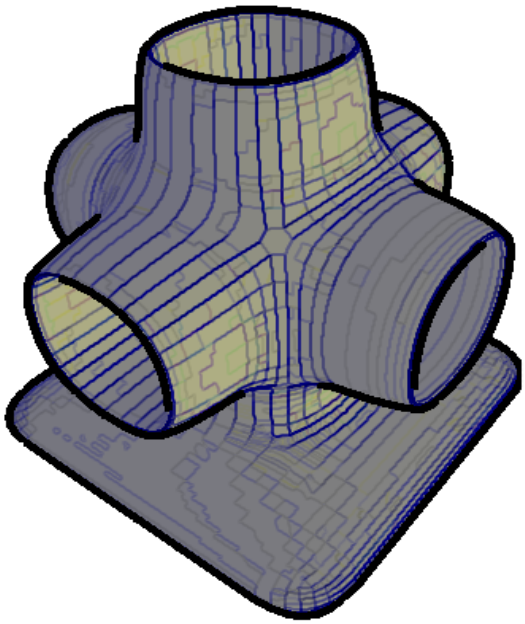


Fig. 67. Model created using a combination of the generalized Menger sponge tool and rind modeling. The starting shape was a cube.

VIII.4. Ideas for future work

The research has provided new insights into topological modeling and its applications in computer graphics. Several interesting ideas for future work have developed through the course of this research. Some of these are improvements to overcome the limitations in the current tools, while others explore new directions in high genus modeling.

1. In the multi-segment curved handle tool, currently, the user has only limited control over the shape of the handle. The range of shapes is also limited by the use of Hermitian curves. Having the ability to create arbitrarily shaped handles would be a useful addition to this tool. The problem is primarily one of developing the appropriate user interface and the core of the current tool can still be used.

2. The user can currently create a handle between two faces. The ability to create handles between multiple faces would provide interesting modeling capabilities.

3. In the rind modeling tool, the algorithm currently creates a rind surface and then allows the user to punch holes in the rind. In some situations, the ability to punch holes on surfaces that already resemble rind shapes would be very useful. An algorithm for such a tool would have to be primarily based on geometrical considerations, which increases the complexity of the problem.

4. As explained in Chapter VII, the generalized Menger sponge tool is currently limited to two iterations because of visual artifacts that are produced with higher iterations. The algorithm also does not create an exact replica of the Menger sponge. Implementing solutions for these problems would be a welcome addition to the tool.

REFERENCES

[1] E. Akleman and J. Chen, "Guaranteeing the 2-manifold property for meshes with doubly linked face list," *International Journal of Shape Modeling*, vol. 5, pp. 149–177, 2000.

[2] E. Akleman, J. Chen, and V. Srinivasan, "A new paradigm for changing topology during subdivision modeling," *Proc. 8th Pacific Conference on Computer Graphics and Applications*, pp. 192–201, 2000.

[3] E. Akleman, J. Chen, and V. Srinivasan, "A prototype system for robust, interactive and user-friendly modeling of orientable 2-manifold meshes," *Proc. International Conference on Shape Modeling and Applications*, pp. 43–50, 2002.

[4] Alias Systems. (2004, Jan.) Maya 5 technical features. [Online]. Available: http://www.alias.com

[5] G. Barequet and S. Kumar, "Repairing CAD models," *Proc. IEEE Visualization '97*, pp. 363–370, 1997.

[6] M. Barnsley, *Fractals Everywhere*. San Diego, CA: Academic Press, Inc., 1988.

[7] B. Baumgart, "Winged-edge polyhedron representation," Stanford University, Tech. Rep., 1972.

[8] F. Bool, J. Kist, J. Locher, and F. Wierda, *M.C.Escher, His Life and Complete Graphic Work*. New York, NY: Harry N. Abrams, Inc., 1992.

[9] I. Braid, R. Hillyard, and I. Stroud, "Stepwise construction of polyhedra in geometric modeling," in *Mathematical Methods in Computer Graphics and Design*, K. Brodlie, Ed., pp. 123–141. Orlando, FL: Academic Press, 1980.

[10] E. Catmull and J. Clark, "Recursively generated B-spline surfaces on arbitrary topological meshes," *Computer Aided Design*, vol. 10, pp. 350–355, 1978.

[11] J. Chen, "Algorithmic graph embeddings," *Theoretical Computer Science*, vol. 181, pp. 247–266, 1997.

[12] D. Doo and M. Sabin, "Behavior of recursive subdivision surfaces near extraordinary points," *Computer Aided Design*, vol. 10, pp. 356–360, 1978.

[13] M. Escher, *M.C.Escher: The Graphic Work.* New York: Barnes and Noble Books, 1994.

[14] H. Ferguson, A. Rockwood, and J. Cox, "Topological design of sculptured surfaces," *Computer Graphics*, vol. 26, pp. 149–156, 1992.

[15] H. Ferguson. (2004, Jan.) helaman ferguson sculpture: Gallery. [Online]. Available: http://www.helasculpt.com/gallery/index.html

[16] (2004, Jan.) The Fast Light Toolkit. [Online]. Available: http://www.fltk.org

[17] A. Fomenko and T. Kunii, *Topological Modeling for Visualization.* New-York: Springer-Verlag, 1998.

[18] M. Forsyth, "Shelling and offsetting bodies," *Proceedings of Solid Modeling'95*, pp. 373–381, May 1995.

[19] J. Gross and T. Tucker, *Topological Graph Theory.* New York: Wiley Interscience, 1987.

[20] L. Guibas and J. Stolfi, "Primitives for the manipulation of general subdivision and computation of Voronoi diagrams," *ACM Transactions on Graphics*, vol. 4, pp. 74–123, 1985.

[21] P. Hanrahan, "Topological shape models," PhD Dissertation, University of Wisconsin, Madison, 1985.

[22] V. Hansen, *Geometry in Nature*. Wellesley, MA: A K Peters, Ltd., 1993.

[23] G. Hart. (2004, Jan.) Geometric Sculpture of George W. Hart. [Online]. Available: http://www.georgehart.com/sculpture/sculpture.html

[24] C. Hoffmann, *Geometric and Solid Modeling, An Introduction*. San Mateo, CA.: Morgan Kaufman Publishers, Inc., 1989.

[25] C. Hoffmann and G. Venecek, "Fundamental techniques for geometric and solid modeling," *Manufacturing and Automation Systems: Techniques and Technologies*, vol. 48, pp. 101–165, 1990.

[26] M. Karasick, "On the representation and manipulation of rigid solids," PhD Dissertation, McGill University, Montreal, Canada, 1988.

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

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

[29] T. Murali and T. Funkhouser, "Consistent solid and boundary representations from arbitrary polygonal data," *Proc. 1997 Symposium on Interactive 3D Graphics*, pp. 155–162, 1997.

[30] J. Rossignac and A. Requicha, "Offsetting operations in solid modeling," *Computer Aided Geometric Design*, vol. 3, no. 2, pp. 129–148, Aug. 1986.

[31] M. Sabin, *Subdivision Surfaces: Tutorial Notes*, 2001 International Conference on Shape Modeling and Applications, Genoa, Italy, May 2001.

[32] C. Schols and B. Mandelbrot, Eds., *Fractals in Geophysics*. Basel, Germany: Birkhäuser Verlag, 1989.

[33] T. W. Sederberg, P. Gao, G. Wang, and H. Mu, "2-d shape blending: an intrinsic solution to the vertex path problem," *Computer Graphics*, vol. 27, pp. 15–18, 1993.

[34] T. W. Sederberg and E. Greenwood, "A physically based approach to 2d shape blending," *Computer Graphics*, vol. 26, pp. 25–34, 1992.

[35] S. Takahashi, Y. Shinagawa, and T. Kunii, "A feature-based approach for smooth surfaces," *Proc. Fourth Symposium on Solid Modeling*, pp. 15–18, 1997.

[36] D. Turcotte, *Fractals and Chaos in Geology and Geophysics*, 2nd ed. Cambridge, United Kingdom: Cambridge University Press, 1997.

[37] G. Vanecek, "Set operations on polyhedra using decomposition methods," PhD Dissertation, University of Maryland, College Park, MD, 1989.

[38] K. Weiler, "Edge-based data structures for solid modeling in curved-surface environments," *IEEE Computer Graphics and Applications*, vol. 5, pp. 21–40, 1985.

[39] K. Weiler, "Topological structures for geometric modeling," PhD Dissertation, Rensselaer Polytechnic Institute, Troy, NY, 1986.

[40] W. Welch and A. Witkin, "Free-form shape design using triangulated surfaces," *Computer Graphics*, vol. 28, pp. 247–256, 1994.

[41] R. Williams, *The Geometrical Foundation of Natural Structure: A Source Book of Design*. New York, NY: Dover Publications, Inc., 1979.

[42] F. Yamaguchi and T. Tokieda, "Bridge edge and triangulation approach in solid modeling," in *Frontiers in Computer Graphics: Proceedings of Computer Graphics Tokyo'84*, T. Kunii, Ed., pp. 44–65. Berlin, Germany: Springer-Verlag, 1985.

[43] D. Zorin and P. Schröder, *Subdivision for Modeling and Animation: Course Notes*, 27th International Conference on Computer Graphics and Interactive Techniques, New Orleans, LA, July 2000.

VITA

Vinod Srinivasan, of Chennai, India, earned the degree of Bachelor of Technology in aerospace engineering at the Indian Institute of Technology, Madras in August 1996. Following that, he received a scholarship to continue his studies in aerospace engineering at Texas A&M University, College Station and earned the degree of Master of Science in December 1999. He started his Ph.D. in architecture in January 2000 under the guidance of Dr. Ergun Akleman in the Visualization Laboratory at Texas A&M University. He can be reached care of:

K. Srinivasan
MIG 276A, NH 1, Velliveedhyar Street
MaraimalaiNagar, Tamil Nadu 603 209
INDIA