WIRE AND COLUMN MODELING

A Thesis

by

ESAN MANDAL

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

MASTER OF SCIENCE

May 2004

Major Subject: Visualization Sciences

WIRE AND COLUMN MODELING

A Thesis

by

ESAN MANDAL

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

MASTER OF SCIENCE

Approved as to style and content by:

Ergun Akleman (Chair of Committee)	Donald House (Member)
John Keyser (Member)	Phillip Tabb (Head of Department)

May 2004

Major Subject: Visualization Sciences

ABSTRACT

Wire and Column Modeling. (May 2004)

Esan Mandal, B.Arch., Birla Institute of Technology

Chair of Advisory Committee: Dr. Ergun Akleman

The goal of this thesis is to introduce new methods to create intricate perforated shapes in a computing environment. Modeling shapes with a large number of holes and handles, while requiring minimal human interaction, is an unsolved research problem in computer graphics. In this thesis, we have developed two methods for interactively modeling such shapes. Both methods developed create perforated shapes by building a framework of tube like elements, such that each edge of a given mesh is replaced by a *pipe*. The first method called *Wire modeling* replaces each edge with a *pipe* that has a square cross-section. The result looks like a shape that is created by a framework of matchsticks. The second method, called *Column modeling* allows more rounded cross-sections for the *pipes*. The cross-sections can be any uniform polygon, and the users are able to control the number of the segments in the cross-section. These methods are implemented as an extension to an existing modeling system guaranteeing that the *pipes* are connected and the resulting shape can be physically constructed. Our methods require an initial input mesh that can either be imported from a commercially available software package, or created directly in this modeling system. The system also allows the users to export the models in *obj* file format, so that the models can be animated and rendered in other software packages.

To my parents

ACKNOWLEDGMENTS

I would like to take a moment to thank my committee chair, Dr. Ergun Akleman, for his knowledge and guidance which helped me to accomplish this thesis and led me through the difficulties in this research. Special thanks to Dr. Donald House who not only served on my committee, but also helped me to grow and to learn in the lab. I would also like to thank another committee member, Dr. John Keyser, for his enthusiasm and encouragement. A very special thanks to Karen Hillier, who helped me grow and see artistically. My sincere gratitude to Dr. Frederick Parke and Carol La Fayette for their advice and tremendous help throughout the past two years.

I would also like to thank Vinod Srinivasan for his help and advice in programming in my thesis work. Also, I would not have been able to finish this thesis without a very special person, Avneet, who gave me hope, support, strength, friendship and love.

Above all I would like to thank my parents and my brother for their encouragement, understanding and unconditional love which has helped me all through my life.

TABLE OF CONTENTS

CHAPTER Page

LIST OF FIGURES

FIGURE Page

FIGURE Page

FIGURE Page

CHAPTER I

INTRODUCTION

I.1. Motivation

I.1.1. Architecture

Classical Architecture

Classical architecture in many parts of the world is ornate and tends to use a lot of perforated stone, wood and metal elements in the interior as well as the exterior of a built structure. Such elements are typically used in building fenestration, furniture, balustrades etc. Usually they are built either by carving a single large piece of building material like stone, or they are assembled from smaller pieces of building material such as wood or metal. Such architectural elements have been prevalent in different styles, throughout architectural history across the world.

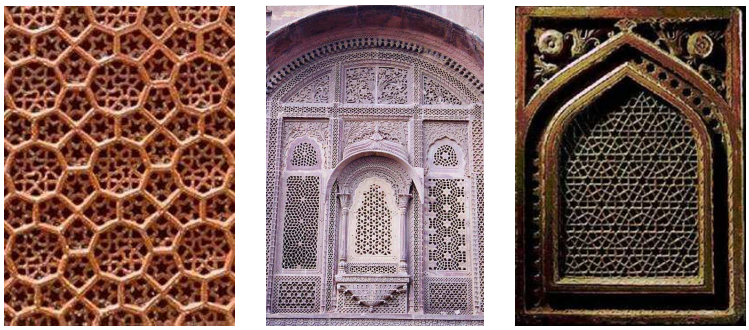


Fig. 1. Examples of a typical stone screen fenestration in early classical Indian architecture [25, 3].

Figure 1 shows examples of architectural motifs such as stone screens and decorative

---

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

cladding that were extensively used in Indian Islamic as well as Hindu classical architecture [25, 3].

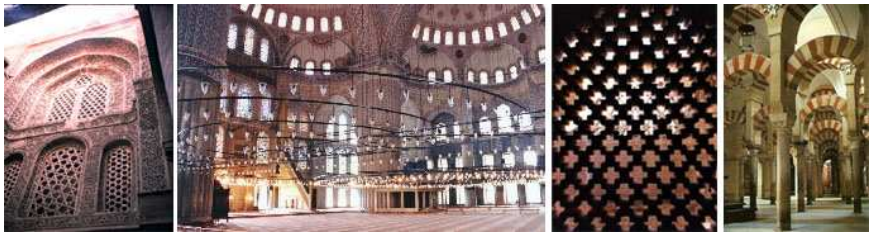


Fig. 2. Classical Islamic architecture [18, 20, 28, 22].

Islamic civilizations in the middle-east have used such elements in their architecture since ancient times [18, 20, 28, 22]. Figure 2 shows examples of the perforated form, which is used at different scales in the built form. The perforated elements range in size from small stone screens to larger forms which influence the whole built space.

Fig. 3. Classical Roman architecture [18].

This kind of porous architectural form was also used in European architectural styles. In ancient Roman architecture, as shown in Figure 3, this kind of perforated feature is used very effectively at a large scale, and dominates the whole built form [18].

In Christian architecture, religious buildings as well as many castles in ancient England used these kind of perforated intricate form in their design, as shown in Figure 4, [10]. The work of Antoni Gaudi, in the late 1800s and early 1900s [21], demonstrates an

Fig. 4. Classical Christian architecture [10].

extensive use of such perforated and ornate architectural elements, as shown in Figure 5.

These kinds of complex, perforated shapes and forms are used even today in architectural interiors as well as exteriors. Figure 6 shows examples of some such modern interior elements [7, 4].

Fig. 5. Antoni Gaudi's architecture [21].

Modern Architecture

Modern Hi-Tech architecture also incorporates the use of many such built elements [16]. In this style of architecture the whole built form is typically made of only steel column and beam elements. As shown in Figure 7, this approach creates a highly perforated built envelope as opposed to a solid envelope common in other architectural styles. This style can be seen as a modern interpretation of the classical perforated stone and wood elements, and makes the classical concept prevalent even today.

The perforated feature of building elements, is incorporated in architectural styles at

different levels of detail: some have it on the surface of the built envelope, some have it at a finer level in windows, openings and interiors, and some have it incorporated in the built volume itself. Thus it can be seen that this characteristic is an integral design feature of many architectural philosophies.

I.1.2. Sculptures

(a) Wooden screen (b) Balustrade

Fig. 6. Interior architectural elements [7, 4].

Stylized sculptures, with a large number of perforations and holes has been a part of modern and ancient art and architectural history. For instance, Indian and Chinese stone sculptures as shown in the Figures 8a,b, display such characteristics. Even contemporary sculptors like G.W.Hart make such highly perforated geometric sculptures [14], as shown in Figure 8c. The beauty of these objects lies in the intricacy of the surface, which is due to the large number of perforations. These perforations make even an extremely tough material like stone look delicate.

In the present day, computer technology has touched almost every aspect of our lives. Computer graphics has enthralled more people than ever due its realistic visual representation of imaginary as well as real worlds. Thus, at some point such beautiful objects need to be represented in the computing environment; sometimes to depict reality as a part of an

architectural visualization; sometimes to realize imaginary worlds in computer art, or as a visualizing tool to make sculptures, as done by George Hart. These objects are as hard to construct in real life as they are to model in a computing environment.

Fig. 7. Modern Hi-Tech architecture [16].

As a part of computer modeling research, it becomes important to devise a method for the easy and fast visualization of such objects in the computing environment. There has been very limited amount of related research in computer graphics that addresses the modeling of such objects, making this an intriguing problem.

(a) Indian Sculpture (b) Indian Sculpture (c) Hart Sculpture

Fig. 8. Sculptures.

Until now these objects were most successfully represented in the computing environment by using indirect methods like texture mapping and rendering-shading techniques.

These methods do not model the objects with three-dimensional data, but just create the look of such objects. Such representations consistently lack the depth and richness of an actual three-dimensional model, and are typically useful only when viewed from a distance. To represent such objects three-dimensionally one would have to use boolean operations repeatedly or tile the smallest tileable feature by copying several times and placing them next to each other. Such available modeling methods tend to be very time consuming and inefficient. There are some techniques available which facilitate efficient modeling of highly perforated objects in computer graphics, but these are very specific to the fields of biology and chemistry where they are used to model molecular structures for scientific study purposes. These methods cannot be used to model more common objects, like perforated architectural elements, sculptures and other interesting forms and shapes in nature.

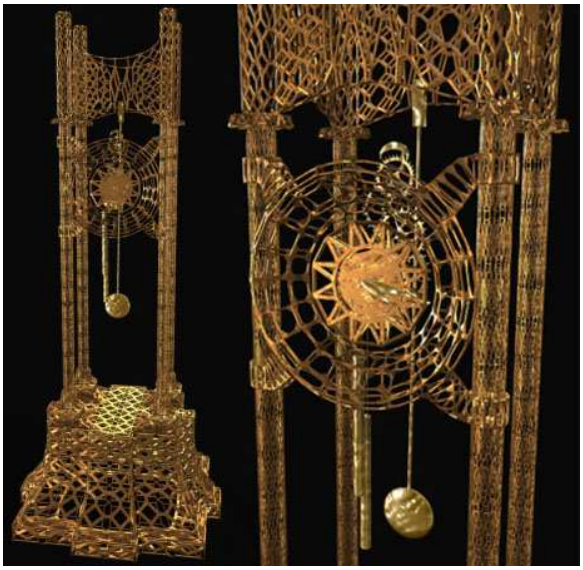


Fig. 9. A clock visualization with perforated elements.

The goal of this thesis is to develop automated computational methods to create highly perforated and intricately designed three-dimensional meshes, as seen in Figure 9, as quickly and efficiently as possible.

CHAPTER II

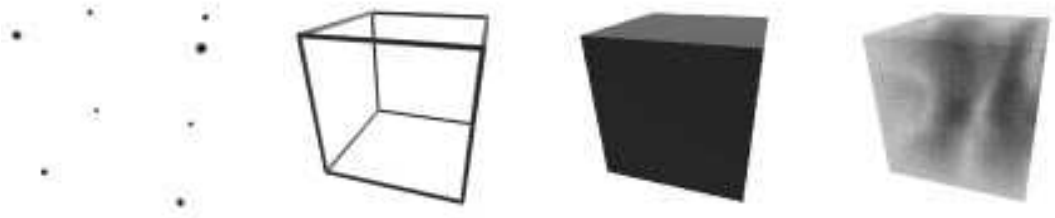
BACKGROUND AND PREVIOUS WORK

II.1. Modeling

Modeling can be considered as a representation of different types of: architectural, engineering, economic, financial, organizational, scientific, mathematical, social and environmental systems to simulate their behavior under various conditions. A *model* is any single representation of the system. Graphical models are often referred to as geometric models, because the component parts of the system are represented with geometric entities such as lines, polygons, volumes etc [15]. Within the scope of this thesis, we are concerned with only geometric models.

Geometric modeling deals with the representation and manipulation of geometric objects in a computer. Geometric Models are represented with geometric entities such as lines, polygons, volumes etc [15]. The field comprises the core of the discipline of Geometric Computation, which encompasses the theoretical and application areas of computer science that deal with geometry and visualization. Among these areas are computer graphics, computer animation, mechanical computer-aided design (MCAD), computer-aided manufacture (CAM), robotics, computer vision, discrete computational geometry and computer-aided geometric design (CAGD) [23]. One of the aims of modeling is the representation of objects by transferring information about reality into the computer. Depending on the type of graphical information to be depicted there are a number of methods, as shown in Figure 10, for modeling a representation in the computer. These include Point Cloud, Wireframe, Boundary and Volumetric Representations.

1. A Point cloud has low computational complexity and is useful to describe complex

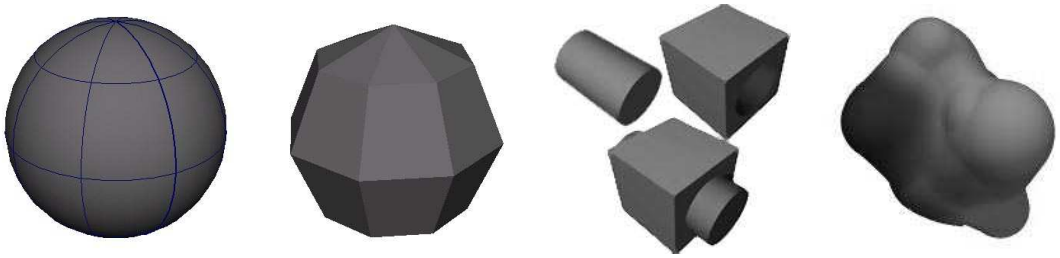


(a) point cloud (b) wireframe (c) boundary (d) volumetric

Fig. 10. Types of geometric modeling representations - point cloud, wireframe, boundary and volumetric respectively.

objects, where their shape is not very important. An object can be represented by one point or by a set of closely spaced points. Particle systems are also used to model natural phenomena.

2. Wireframe models describe objects by vertices and edges [6]. Edges are usually straight lines, but can also be curves, such as NURBS.



(a) Nurbs (b) Polygonal (c) CSG (d) Implicit

Fig. 11. Types of boundary representations.

3. Because a surface is usually the only visible part of an object, this representation is one of the most common representations used to visualize objects. As shown in Figure 11b, complex objects are represented using a polygonal mesh [6]. For better approximations of a curved surface, bezier or NURBS patches are used instead of flat polygons, as shown in Figure 11a. The boundary of some objects can also be

represented using CSG modeling and Implicit surfaces, as seen in Figure 11c,d.

4. In contrast to previous representations, a volumetric representation also regards the inner structure of an object. Simple objects can be represented analytically. Complex ones are described using CSG or Voxel representations. A Voxel representation is especially useful where an object has a heterogeneous inner structure. A big disadvantage of this method however, is that it is needs a huge amount of volume data to represent very simple objects.

For the purpose of this thesis we are concerned with only the Boundary Representation technique. We use the polygonal modeling method of boundary representation because it is one of the most widely used techniques to represent complex objects and is compatible with many commercially available software packages.

For a better understanding of this thesis work we would like to define a few terms which are related to this work.

1. **Topology**

   Topology is the study of how geometric objects are intrinsically connected to themselves. Since topologists are not concerned with the geometric measurements of objects, people often say that they study objects up to continuous deformation. But usually topologists consider spaces which have a topology (a qualitative shape or connectivity) but no predefined (quantitative) geometry. Knots and manifolds are typical examples of topological objects.

2. **Manifold**

   In mathematics, a manifold is a topological space that looks locally like the “ordinary” Euclidean space and is a Hausdorff space. One such example is the surface of a

sphere such as Earth, which is not a plane, but small patches of it are homeomorphic to (i.e., topologically equivalent to) patches of the Euclidean plane.

3. **2-Manifold**

   A 2-manifold is a manifold in two dimensions, usually embedded in higher dimension space. In mathematics, any surface is a 2-manifold. A 2-manifold is physically realizable, and hence can be constructed in real life.

4. **Genus**

   In geometric topology, the number of holes of an object/shape is defined as its genus. So a high genus manifold is a surface that has many holes in its shape.

In this thesis we be modifying the topology of a low genus 2-manifold to create a high genus 2-manifold.

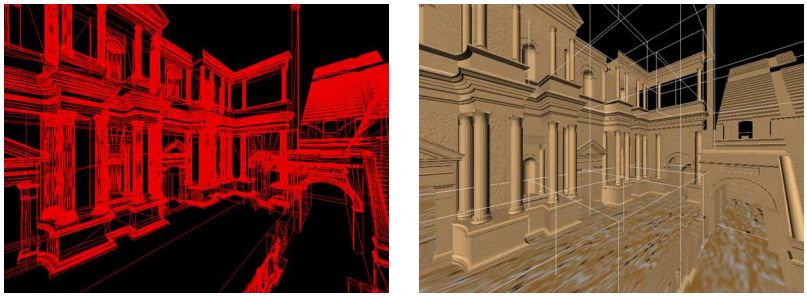
Geometric modeling has been the topic of a lot of research in Computer Graphics. Polygonal modeling has been popular both in the industry as well as in the academic circles. In spite of this interest in modeling methods, the modeling of high genus shapes has received little attention from graphics researchers. On the other hand, industry professionals have come up with interesting indirect methods to represent such shapes. This section briefly discusses some such approaches and research methods.

II.2. Previous work to represent very high genus models

II.2.1. Wireframe modeling

The wireframe model is perhaps the oldest way of representing solids. A wireframe model consists of two tables, the vertex table and the edge table. Each entry of the vertex table records a vertex and its coordinate values, while each entry of the edge table has two

components giving the two incident vertices of that edge. A wireframe model does not have face information. This technique is very efficient as it can convey a lot information with very little resources. Many commercial software packages typically CAD packages support wireframe modeling.



(a) Autocad wireframe rendering (b) Autocad shaded rendering

Fig. 12. Autocad rendering in wireframe as well as shaded mode [5].

All models created using the wireframe modeling technique appear to be made of wires and look high genus. This is so because the model is represented only with edges and vertices, but no faces. The model is not really high genus, and the wires in the model are not actual 3D geometry, but just geometric lines. Moreover, as there is no face information the model cannot be shaded, creating a very flat look. As shown in Figure 12a, wireframe models are very ambiguous as just by looking at one it is difficult to tell which parts are actually holes and which are solid [5]. In Figure 12b it can be seen where the actual holes are.

II.2.2. Wireframe rendering techniques

There are several rendering tricks that have been used to make such objects look as though they are wireframe models. Several commercial software packages support wireframe rendering. In this kind of rendering technique, the renderer does not render the

polygonal faces but just renders the polygonal edges. Since it renders only the edges, the object looks as though it is made of wires.

Wireframe rendering techniques can be categorized into two categories.

1. Hardware wireframe rendering is supported by software packages like Maya and Cinema4d by using their Hardware Render buffer. Using this technique, one can quickly create wireframe images of scene models. The main drawback in using this technique besides getting a flat looking image, is that the whole scene needs to be rendered in wire frame. Moreover, the thickness of the 'wires' in the wireframe are fixed. Jared [17] developed a methodology to do hidden wireframe renders in Maya. His methodology uses the Maya hardware buffer hence has the same drawbacks, the only advantage being that his method occludes back and hidden faces, as shown in Figure 13.

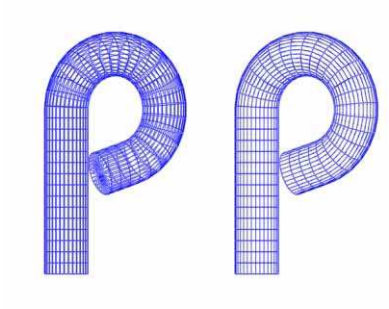
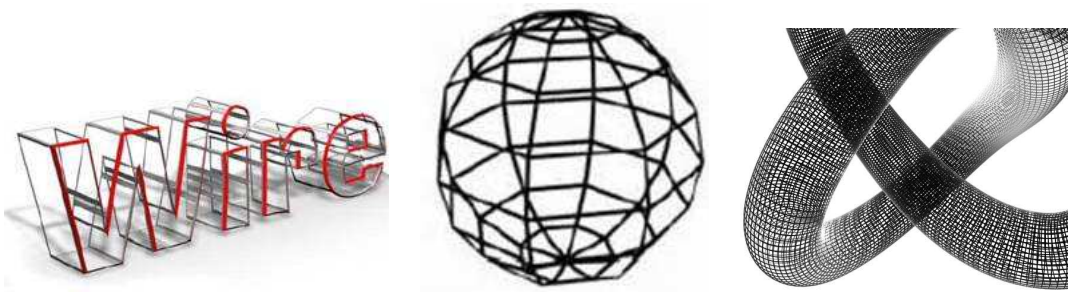


Fig. 13. Maya hardware rendering for wireframe render with and without backface showing.

2. Another popular technique is to use shaders to shade the polygon edges but not the faces, with a thickness. For instance, Everett [9] has developed a Cinema 4d shader plug-in that lets the users render only the edges as, shown in Figure 14a. This shader lets the user choose the thickness and color of the wireframe. Harris has developed a shader for Softimage XSI that shades only the polygon edges and lets everything else be fully transparent, as shown in Figure 14b, [13]. 3ds Max has built-in material

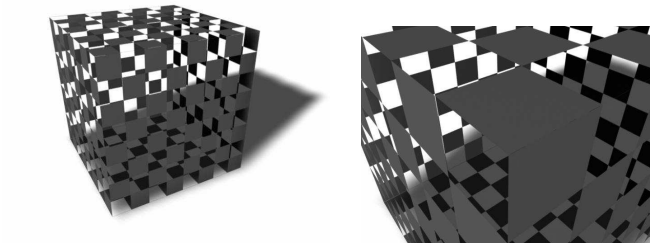


(a) Cinema4D plugin (b) XSI shader (c) 3DSmax material

Fig. 14. Software wireframe renderings.

to wirefame render objects, as shown in Figure 14c, [27]. These approaches are not truly three-dimensional and hence lack the depth and richness of a real world object.

II.2.3. Texture mapping techniques



(a) Transparency mapping with shadows (b) Close up detail

Fig. 15. A simple cube rendered with a checkered transparency map .

Transparency is the most widely used approach to represent objects that look very high genus. The technique makes use of the alpha channel in an image for a texture map to make parts of it transparent and parts of it opaque. When applied to an object it makes the object look as though it has holes in the transparent areas, as shown in Figure 15.

It can be very effective in creating the illusion of an object that is high genus and is extremely complex. Note that one can generate correct shadows with transparency mapped

textures, it is just more complicated to implement. Objects represented using this technique lose depth and richness when seen up close in detail, as shown in Figure 15c.

Transparency mapping can also be combined with displacement mapping to produce such high genus models with more depth and richness. Transparency mapping and displacement mapping can be combined it two ways. The first method is illustrated in Figure 16.

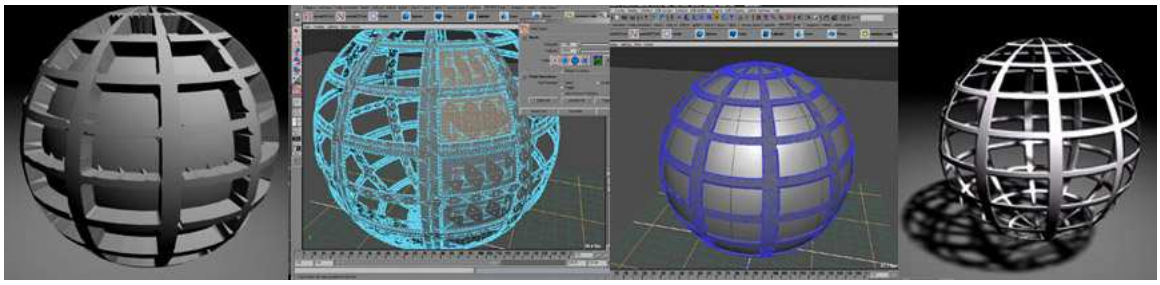


Fig. 16. This example illustrates the use of transparency maps with displacement maps [26]. The image shows the polygon version of the displacement map, the manual deletion of faces that are transparent, the displacement polygon on top of the original polygon and the final result.

In this example [26] normal transparency mapping is used for the model color where as richer shadows are obtained by the use of displacement mapping. A polygonal model is transparency mapped in the standard way and is used only for the primary color render with the shadow casting turned off. A displacement map with positive displacement for opaque areas is applied on a copy of the model. A Maya feature, which converts a displacement map to its equivalent polygonal mesh, is used to get a 3D mesh of the displacement map. The parts of the displacement mesh which correspond to the transparent areas in the transparency map is manually deleted using Maya's artisan tool. This version of the model is then layered on top of the original model and is used only to render the shadow; color rendering is turned off for this model.

The final result is obtained by compositing the two different renderings in maya. This technique is quite effect which is quite effective when viewed from a distance but loses quality and richness on very close inspection.

The second method to combine Transparency with displacement is illustrated in Figure 17. Peter implemented this method [24] in XSI Softimage using a transparency map to make parts of the model look like they have holes. He uses a displacement map, which is identical to the transparency map to give depth to the opaque areas in the model. Both the transparency and displacement maps are derived from the color map so that all three of them match up and create a rich looking high genus object.

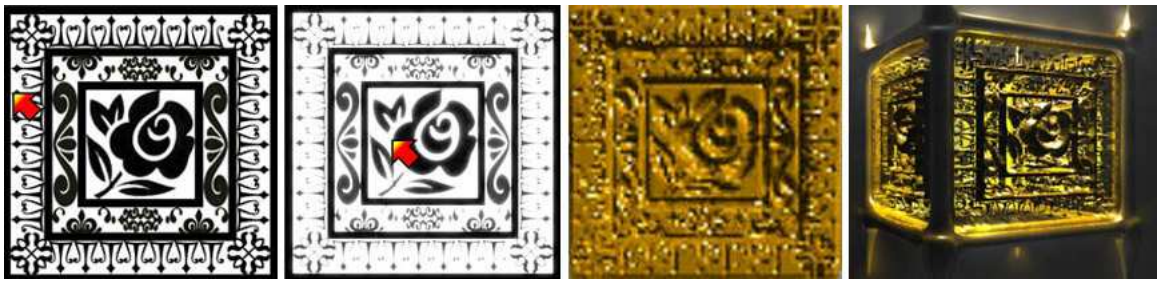
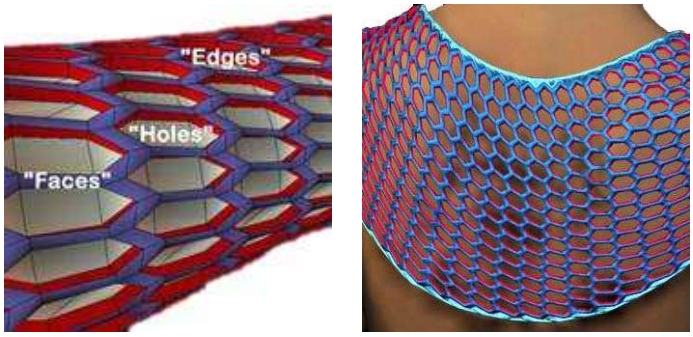


Fig. 17. This example illustrates the use of transparency maps with displacement maps [24]. The image shows the transparency map, the displacement map the color map and the final image with all the three maps applied together.

Transparency mapping can also be used in conjunction with a very detailed model, as shown in Figure 18b. This example [19] creates a very detailed model with a feature called "*Parallax Highlights*". This feature causes polygonal faces to be shaded differently depending on their orientation to the camera. The mesh created is detailed, such that it has separate faces for the holes and opaque areas. As can be seen in Figure 18a, the model has separate materials for the "*Face*", "*Edge*" and "*Hole*" in the mesh. The "*Face*" and "*Edge*" are usually textured opaque and the "*Hole*" is textured transparent. The parallax feature is then used to give more depth to the model. This method creates a very high polygonal

count. Since each face group is assigned a different texture, texturing also becomes a tedious process. This sophisticated method creates a more believable three dimensional look.



(a) Mesh Detail (b) Final Model

Fig. 18. This is another example of a texture map with transparency. The method used here is more sophisticated and use “parallax highlights” to create a more believable three dimensional look [19].

Transparency mapping is the most popular technique to represent high genus models. It is very easy to use and is usually supported by most commercial software packages. This technique can create very complex looking shapes as the final shape is based on an image. Although this technique is very versatile and efficient it does not create an actual three-dimensional shape. The model loses richness and three-dimensional depth when viewed from a close distance. When used with displacement maps the model looks more three dimensional, but even this method loses detail on close inspection, and has displacement related artifacts. Moreover, such models look three-dimensional from only one side as the displacement is positive only along one direction.

Thus to summarize transparency mapping techniques are very efficient and easy to use but the results are not truly three-dimensional and hence lack depth and richness.

II.2.4. Modeling techniques

Boolean Operations

The most commonly used modeling technique to create high genus shapes is Boolean Operations, which is usually supported by most commercial modeling packages. It can be used to quickly and easily create holes in a model, as shown in Figure 19.

(a) Simple Boolean operations (b) Boolean operation to model a building (c) Undesirable Mesh structure

Fig. 19. Some examples with boolean operations.

This is an approach that increases the genus of actual three dimensional geometry. However, it is very unsuitable for creating objects with very high genus [29]. One would have to use the Boolean operation many times. Boolean operations are not very robust and result in mesh data that is not clean, thus resulting in an unstable model state. One can manually clean up a mesh that has only a few Boolean operations, but it is unreasonable to do so for a large number of such operations.

Molecular Modeling

Many scientific visualization software packages, used in biological and chemical molecular visualization [12, 11], are used to represent molecular structures, as shown in Figure 20. These methods create objects that are truly high genus and three dimensional; thus

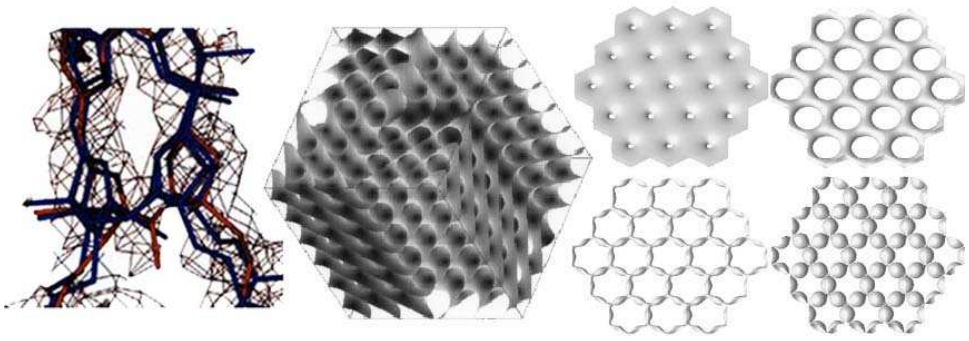


Fig. 20. Molecular visualization.

have depth, and are rich in appearance. Unfortunately, in spite of producing good results, the modeling is based on protein data, and is specific to the purpose. It cannot be used to model a wide variety of real world objects and has limited flexibility.

Special Purpose Software

There are also people who develop their own special purpose software.

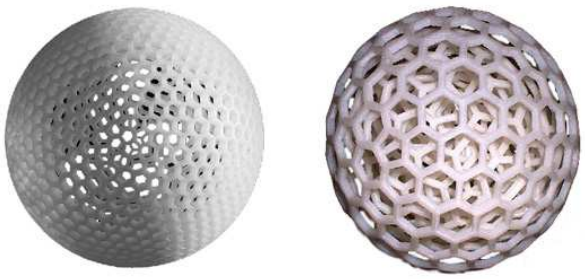


Fig. 21. George W. Hart's sculptures.

For instance, mathematical sculptor George Hart creates highly geometric sculptures, including very high genus models, using his own software. His sculptures are very intricate and truly three-dimensional, but he creates only geometric shapes which are mathematically visualized, as seen in Figure 21. His process is not geared towards ease of use as he codes specifically for each of his creations, making them very personal. Moreover it cannot

be used to create organic natural shapes [14].

Other mathematicians create mathematical visualizations in which very beautiful high genus shapes are created. These shapes are usually the result of mathematical equations plotted in three-dimensional space, as shown in Figure 22. As these shapes are generated from mathematical equations, it becomes very difficult to generalize the methodology so as to generate any kind of shape. Moreover, to create a new shape one would have to come up with new equations or modify existing ones; which is not very intuitive for artists and users who do not have a mathematical background.

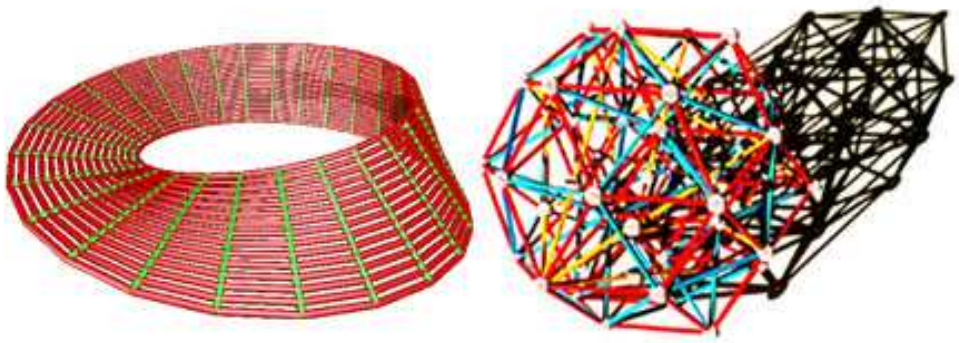


Fig. 22. Mathematical visualization.

Rind Modeling

Rind modeling is a method that can be used to easily create holes on the surface of an object [2]. This method creates a surface thickness which can be sculpted with holes, as shown in Figure 23.

This surface thickness is created by constructing another surface that is offset from the original by a user specified amount. To create this offset each vertex in the surface mesh is moved along the average of the face normals for that particular vertex. This causes the new surface to be nested inside the original surface. The normals of the second nested mesh are then reversed. This operation changes the inside and outside of the 2-manifold

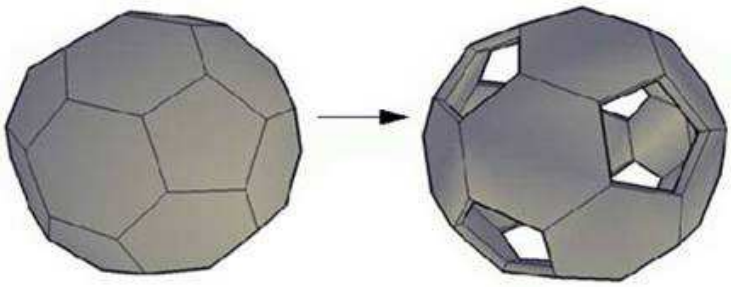


Fig. 23. Rind modeling.

mesh by changing the rotation orders of the faces. Both of these surfaces together create the desired surface thickness that can be sculpted. The user can then select the faces that need to be punctured. The method automatically identifies the corresponding face in the nested surface and does a hole-handle operation to open a connecting hole between these two faces to make them one surface. The two faces are deleted and the sides of the hole are closed with new faces to complete the manifold. Some shapes created using this method are shown in Figure 24.

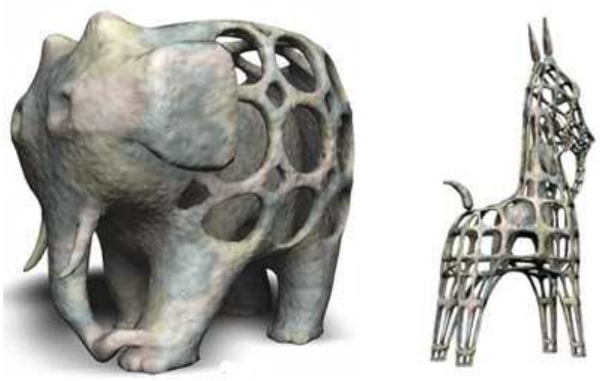


Fig. 24. Rind modeling example.

The Rind modeling method creates shapes that are three-dimensional and high genus. Since the shapes are three-dimensional they have depth and create rich images when rendered. The only disadvantage in this method is that to create a very high genus shape the

user has to repeatedly apply the hole-handle operation, which is very tedious. One of our proposed methods is based on the Rind modeling hence this method is very important for this thesis.

This brief study shows that there are a lot of indirect yet innovative methods to represent such complex objects. Unfortunately almost all these methods consistently lack the detail and richness of a true three-dimensional model. On the other hand they have one important advantage, that most of these methods do not generate large amounts of mesh data. In contrast to these indirect methods there has been some work done by computer graphics researchers which addresses this issue more directly but unfortunately they have limited capabilities. This study thus implies that there is a lot of room for improvement in this field.

CHAPTER III

METHODOLOGY

In this chapter we will discuss the methodology to achieve our goal to create very high genus models with the characteristics discussed before. To approach the problem it will be necessary to study the actual objects that we are going to model. These objects have the following notable characteristics

1. They have a very large number of perforations, and may be of genus thousand or more.
2. There are no large continuous faces in the object shape. The solid parts in the objects are typically cylinder-like, making the object shape look like an intricate framework of ‘wires’.
3. These objects have an overall recognizable shape, i.e. the object may look like an elephant, sphere etc.

We will also formulate certain criteria from a usability point of view.

1. There should be minimum user intervention. The process should be nearly automatic.
2. The finished model should be in an easily transferrable format for animation and rendering purposes.
3. The user should be able to control the cross-section of the *3D pipes*.

Based on the above observations we will model shapes that have a large number of perforations whose solid parts would be pipe-like, and have a recognizable shape. These objects can be thought of as being constructed in two different ways.

1. The first point of view would be to see these objects as having a large number of perforations. The perforations are only on the surface of the object. They are a surface feature and do not change the overall shape of the object. For example a sphere will still look like a sphere but will have a large number of holes on the surface. This approach would prompt a methodology that will sculpt an input mesh with a large number of holes on the surface, as shown in Figure 25.

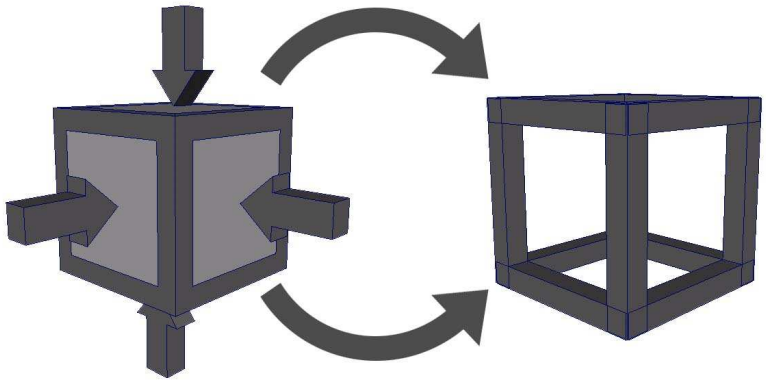


Fig. 25. First approach: Sculpt the input mesh by punching holes on the surface.

2. The other point of view would be to think of these objects as made up of a large number of interconnected *3D pipes*. This approach is like building the object with building blocks, in which the building blocks consist of pipes and joints, as shown in Figure 26. The pipes can be of different cross sections, and the joints will hold the pipes in place to create the final shape.

Both of the above mentioned approaches are opposite in principle but similar in result. One approach creates the final shape essentially by subtracting 3D geometry from a larger geometry, while the other approach creates the final shape by putting together many smaller pieces of 3D geometry to create a large geometry. The end result of both the approaches is essentially the same, i.e. each create intricate wire-frame like shapes that are made of *3D*

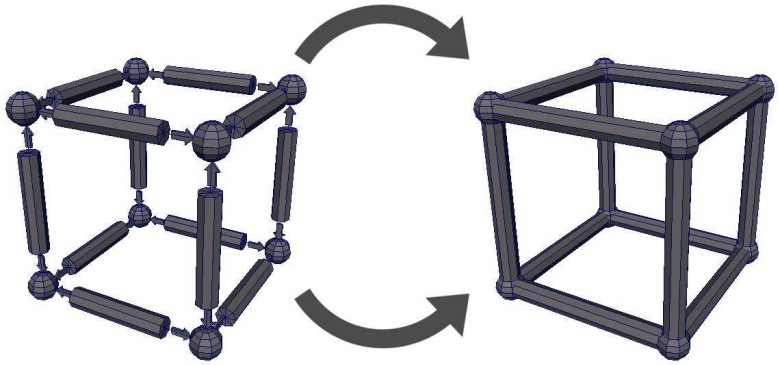


Fig. 26. Second approach: Assemble joints and *3D pipes*.

pipes systematically joined together. It should be noted that since we are aiming to create a recognizable shape in the end it will be necessary to start with such a shape initially. The first approach will sculpt this initial shape with holes, while the second approach will use it as a reference to place and assemble the *3D pipes* in space.

Although conceptually both the methods are different but both of them create a model that looks like a framework of *3D pipes*, hence in both the cases we will refer to these model elements as *3D pipes*.

Each approach has advantages and disadvantages. We argue that the first approach is easier to implement and less expensive. In the first approach, we are sculpting the object surface with holes, thus we have more control on the shape and size of the holes and almost no control on the solid parts, i.e. the *3D pipes*. The second approach allows the user to control the cross-section of the *3D pipes* because conceptually the *3D pipes* are building blocks which we assemble and hence, can have any cross-section. This approach is more expensive because the joints that connect the *3D pipes* are complex and difficult to compute.

Since both of the above approaches have advantages as well as disadvantages, we implement both. As part of the discussion of results in this thesis we compare and contrast the two approaches from the point of view of usability. In doing so we see that certain

configurations of the input mesh makes it advantageous to use one approach over the other.

III.1. Wire modeling approach

The Wire modeling scheme is an extension of the Rind modeling method [2]. In our approach we have automated the Rind modeling procedure for all the faces in the input mesh.

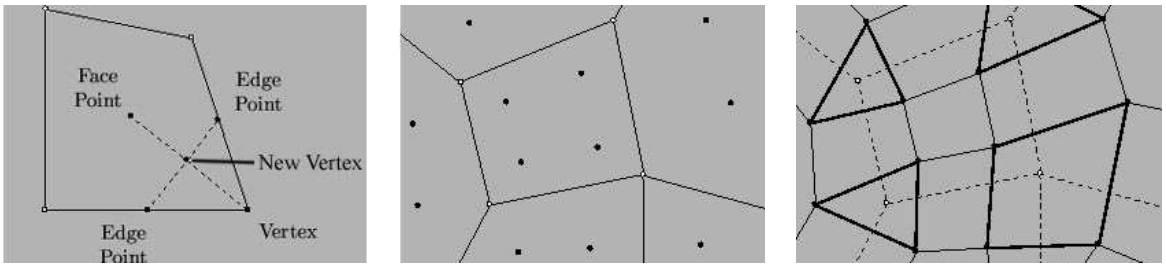
In this scheme we sculpt an input mesh with holes. We have modified the input mesh structure such that we have control on the shape and size of the holes. Since the *3D pipes* lie along the edges of the input mesh and are joined together at the vertices so we replace the edges and vertices in the input mesh with faces. These new faces in the input mesh form the *3D pipe* structure in the final shape.

This modification in the mesh structure can be easily achieved by applying a Doo-Sabin [8] subdivision scheme to the input mesh. The Doo-Sabin [8] scheme creates a new face for every *edge*, *vertex* and *face* in the input mesh, which we call *edge-faces*, *vertex-faces* and *face-faces* respectively, as shown in Figure 29a. We modify the original Doo-Sabin scheme so that we have more control on the shape and size of the new faces created.

III.1.1. Doo-Sabin modification in Wire modeling

Any polygonal 3D mesh has three main components - *faces*, *edges* and *vertices*. A vertex in the mesh is typically shared by a number of edges and faces, which defines the valence of the vertex.

Consider a vertex, and a face that shares that vertex, in the original mesh. The Doo-Sabin [8] scheme creates a new vertex for every such *face-vertex* combination. This new vertex is created such that it is the average of the vertex point, the two edge points (the



(a) Step 1 (b) Step 2 (c) Step 3

Fig. 27. The original Doo-Sabin scheme.

midpoints of the edges that are adjacent to this vertex in the face), and the face centroid Figure 27. This is done for every such face-vertex combination in the mesh, and then edges are inserted between the newly created vertices to get a smooth shape.

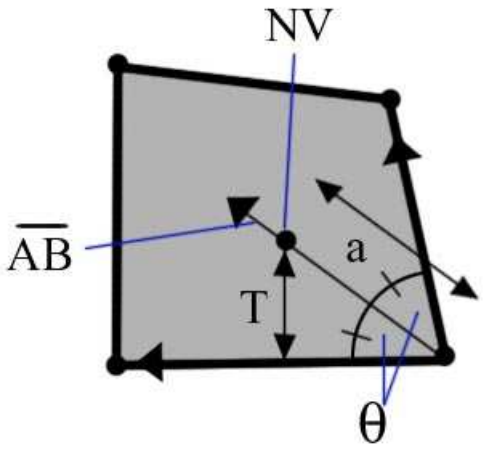
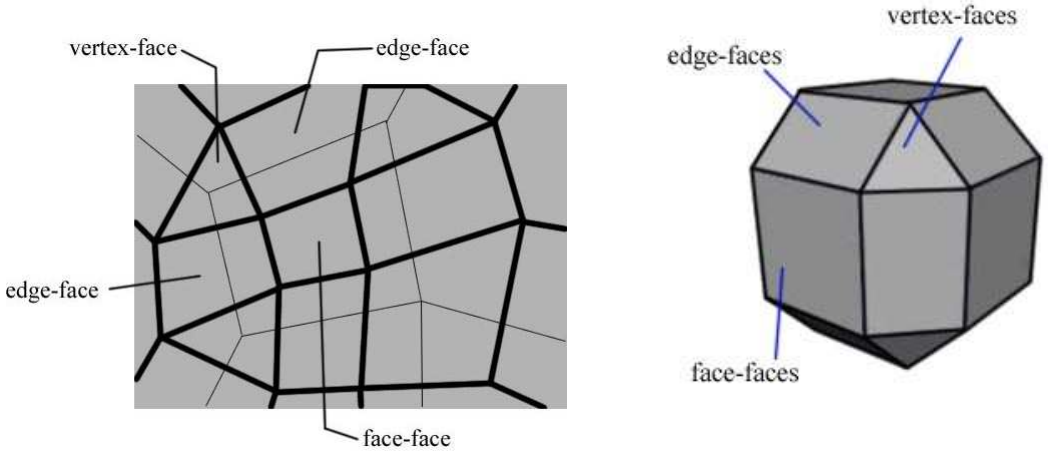


Fig. 28. The modification in the Doo-Sabin scheme.

In our scheme we modify the position of the new vertices such that they lie on the angle bisector of the corresponding two edges and also fall on the corresponding face plane, as shown in Figure 28. Since the new vertex *NV* lies on the angle bisector *AB* we can specify how far out it lies along the angle bisector. This distance *a* controls the size of the new

faces created, and with some simple math it can be easily used to control the width *T* of the faces that corresponds to the edges and vertices.



(a) Modified Doo-Sabin (b) Modified Doo-Sabin applied on a cube

Fig. 29. The modified Doo-Sabin scheme.

So our first step is to apply this modified Doo-Sabin scheme to the input mesh which creates a mesh structure suitable for our purpose, as shown in Figure 29a.

III.1.2. Rind modeling integration

The modified Doo-Sabin scheme creates a mesh structure, as shown in Figure 29b. The faces labeled as *edge-face* is developed into the *3D pipe* geometry.

As mentioned before the Rind modeling method creates a nested mesh which has corresponding faces for the *vertex-face, edge-face and face-face*. We keep track of the *face-face* faces in the mesh and automate Rind modeling to punch holes in all such faces. This leaves behind the *edge-faces* and *vertex-faces* in the geometry, and creates the framework of *3D pipes* to form the final shape.

The *edge-faces* are connected to their respective faces in the nested geometry via

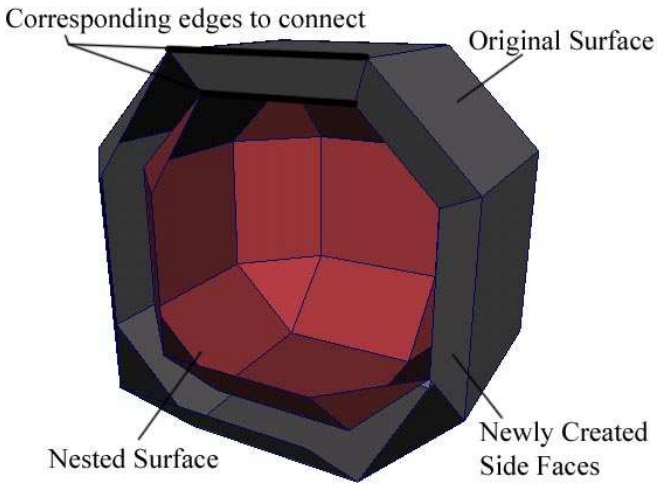


Fig. 30. The nested surface and connecting side faces in Rind modeling.

newly created *side-faces* in Rind modeling, as shown in Figure 30. These pair of *edge-faces* and pair of *side-faces* form the *3D pipe*. As the *3D pipes* need to have a square cross-section, all four faces must be perpendicular to each other. Since the structure of the *3D pipes* are formed by the vertices of the nested surface,hence we are very particular as to how we create them.

The mesh configuration at a vertex is shown in Figure 31. As we are dealing with quadrilateral faces, we cannot guarantee that all such faces will be planar. So we localize our perpendicularity condition to the vertex in question. So restating, we want the *3D pipe* faces to be perpendicular locally at the vertices. In the original Rind modeling scheme we create the nested vertices such that they lie on the average of the face normals that share the vertex. This does not guarantee the required condition, hence we approach the problem, as shown in Figure 31 and Figure 32.

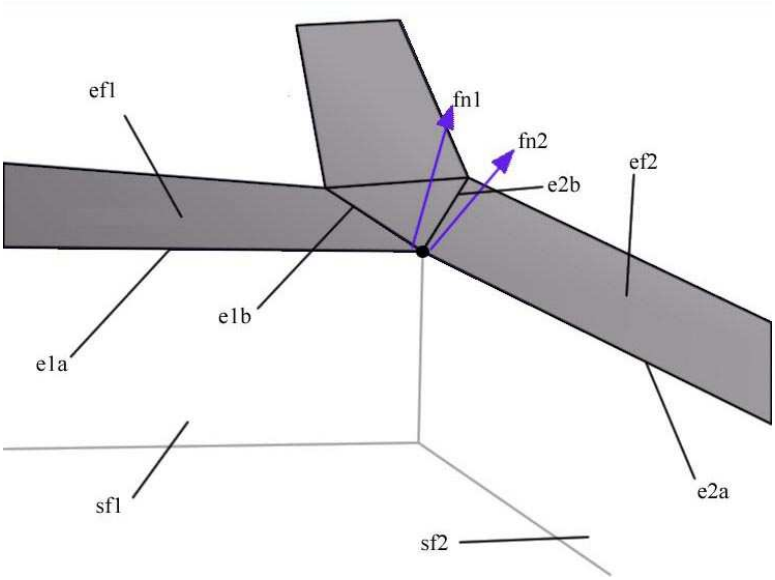


Fig. 31. The mesh configuration at a vertex.

fn1 is the face normal of face ef1 at vertex V in Figure 31.

$$fn1 = e1b \times e1a$$

fn2 is the face normal of face ef2 at vertex V in Figure 31.

$$fn2 = e2a \times e2b$$

sfn1 is the face normal of the side-face sf1 (not created) at vertex V in Figure 32a.

$$sfn1 = fn1 \times e1a$$

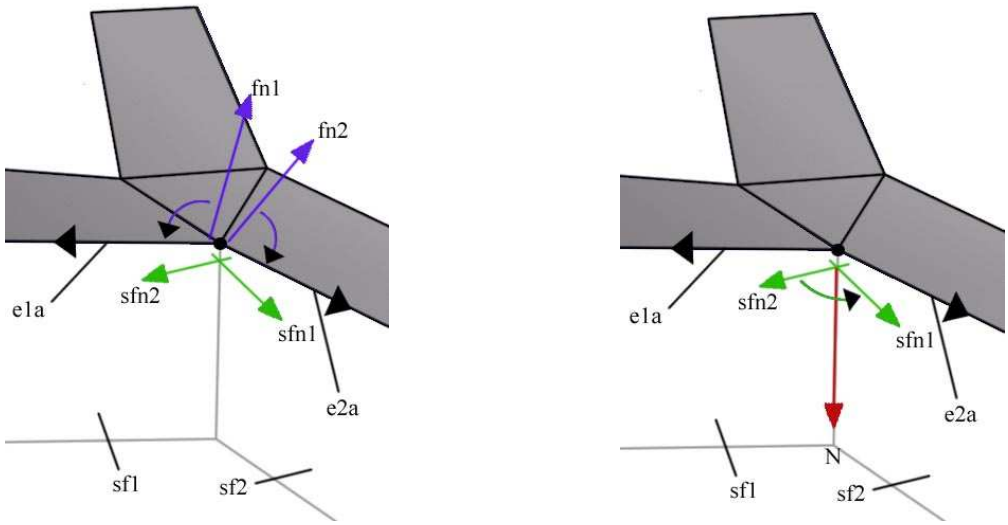
sfn2 is the face normal of the side-face sf2 (not created) at vertex V in Figure 32a.

$$sfn2 = e2a \times fn2$$

N is the direction along which the nested vertex should lie, as shown in Figure 32b.

$$N = sfn1 \times sfn2$$

It is easy to see that N is perpendicular to both ef1 at V as well as ef2 at V. This modification guarantees that the faces will be locally perpendicular at the vertices and the nested surface will be calculated using this methodology.



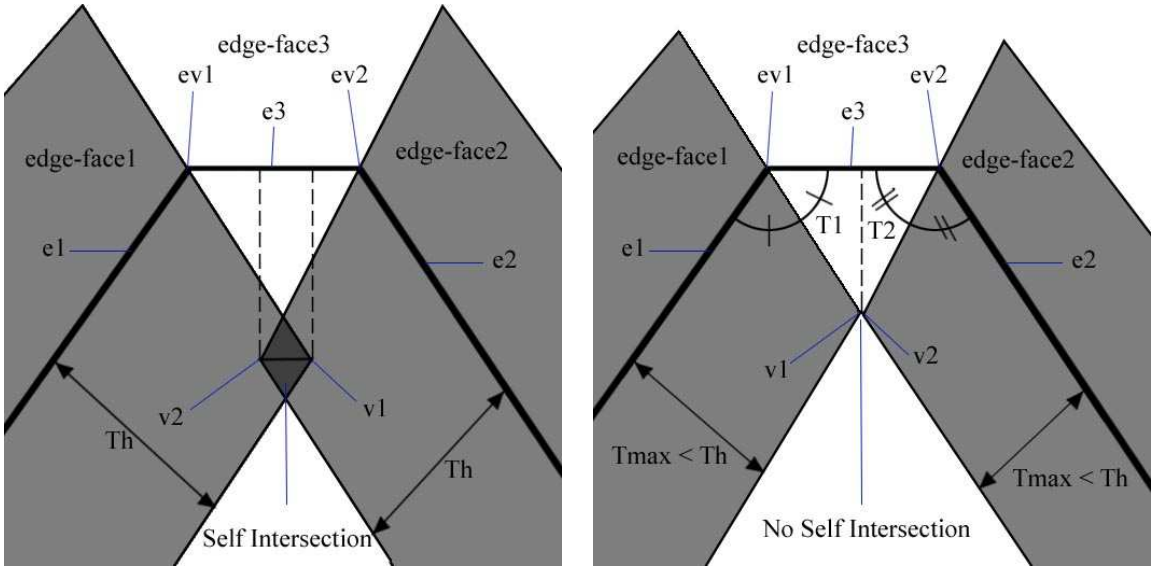
(a) Calculate *sfn1* and *sfn2*

(b) Calculate *N* - the direction at which the nested vertex is offset

Fig. 32. The faces of the *3D pipe* are locally perpendicular.

III.1.3. Dimension control of the 3D pipes

We need to define the parameters in the model that let us control the dimensions of an individual *3D pipes*. The depth of the 3D pipes is the same as the surface thickness set during the Rind modeling step. The length is the length of the edge that the particular *3D pipe* corresponds to, which depends on the input mesh. The thickness of the *3D pipe* is the width of the edge-face that we created in the Doo-Sabin step. The thickness and depth of the *3D pipe* is the same as it has a square cross-section. We provide a user input field for specifying this parameter. It should be noted that if the thickness of the *3D pipes* exceeds a certain value it causes self-intersections in the model,which is not desirable. We thus limit the thickness of the *3D pipe* so that we do not have such intersections.



(a) Self-intersection with user defined thickness

(b) No self-intersection with corrected thickness

Fig. 33. Calculating *Tmax* to prevent self-intersection.

III.1.4. Self-intersection

Self intersection occurs when there is an edge configuration, as shown in Figure 33. We know that vertex *v1* lies on the angle bisector of edges *e1* and *e3*, and vertex *v2* lies on the angle bisector of edges *e3* and *e2*. To avoid intersection *v1* and *v2* need to be moved along their respective angle bisectors towards *ev1* and *ev2* respectively, till there is no intersection. To simplify the problem we assume that the three edges lie on a plane. Although this does not yield an accurate minimal result, it however guarantees no intersection. We then move *v1* and *v2* such that the vector *v1v2* has zero magnitude, i.e. they coincide. Thus the maximum permissible thickness *Tmax* is

$$Tmax = (edgeLength \times tanT1 \times tanT2) \div (tanT1 + tanT2\ )$$

where, edgeLength is the length of edge e3, i.e. magnitude of the vector ev1ev2

T1 is the angle bisector of angle between edges e1 and e3

T2 is the angle bisector of angle between edges e2 and e3

In Wire modeling all the *3D pipes* have the same depth, hence we find the maximum permissible thickness for every set of three continuous edges and use the minimum of all the maximum thicknesses, to make the model. If the user input thickness is less than the calculated one it is used, otherwise the calculated maximum thickness is used. It should be noted that this step needs to be done before the Doo-Sabin scheme is applied to the initial input mesh, in fact this is the first step in the methodology.

After we have made this modification we automate Rind modeling to punch holes in all the *face-face* faces in the mesh. This last step completes the wire-frame model.

To summarize the Wire modeling methodology:

1. Calculate the maximum permissible thickness of *3D pipes* . Use this calculated thickness or the user input thickness whichever is less.

2. Apply the modified Doo-Sabin scheme to the input base mesh to create a mesh structure which creates faces that can be developed into *3D pipes* .

3. Keep track of the faces that correspond to original faces in the Doo-Sabin scheme, call them *face-face*.

4. Modify the creation of the nested surface in Rind modeling to guarantee that all the faces in the *3D pipes* are perpendicular to each other.

5. Automate Rind modeling on the modified mesh to punch holes in all the *face-face* faces to complete the model.

III.2. Column modeling

In Column modeling we use an input base mesh as a reference to place the *3D pipes* as opposed to sculpting it directly, as done in Wire modeling. As mentioned before, Column

modeling creates shapes by making a framework of *3D pipes* and *joint shapes* that keep them together. There are four main steps to the Column modeling methodology.

1. Compute the joint shapes using a convex hull algorithm that joins the *3D pipes* to make the model.

2. Clean up the convex hull to get rid of triangulation and get faces in the shape of the desired *3D pipe* cross-section for each edge in the model.

3. Each edge in the model has a pair of corresponding faces in the convex hulls created. Keep track of the edges and their corresponding faces.

4. Use the built-in handle operation to create a handle between the pair of faces for each edge as stored in the previous step. This completes the model.

III.2.1. Joint shapes computation

The computation of the joint shapes is the most expensive and complicated part of the methodology. The joint shape is typically very complex and organic, and its shape depends on the complexity of the input mesh structure. In this section we outline the methodology to compute the joint shape.

In Column modeling the *3D pipes* replace the edges in the input mesh. The edges are joined together at the vertices in the input mesh hence the joint shapes that we create are centered at the vertices too.

The *3D pipes* can be thought of as frusta with an arbitrary user defined cross-section. The joint shapes create a minimal shape by connecting the end-faces of the frusta which belong to edges that originate from one vertex, as shown in Figure 34. Hence, to compute the joint shape we need this set of end-faces for the vertex. The end-face is a planar polygon

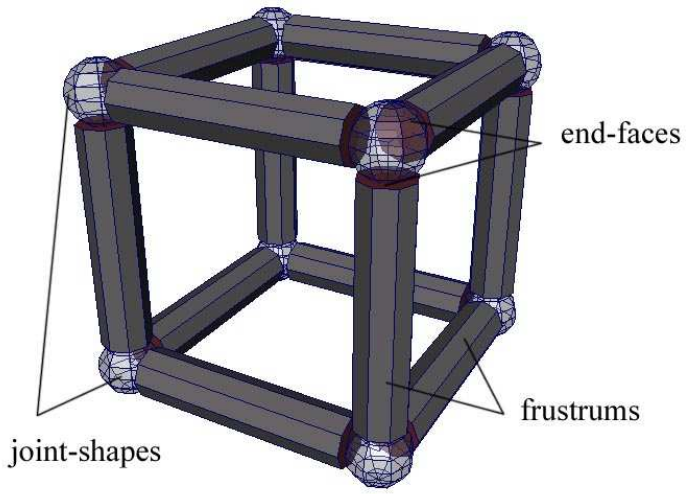


Fig. 34. Frustums or *3D pipes* and the joint shapes.

of the shape of a user defined cross-section for the *3D pipes*, whose normal vector is the same as the edge vector and whose centroid lies on the edge vector, as shown in Figure 35.

Such a polygonal face for an edge is created by computing the position of each vertex in the polygon incrementally, as shown in the Figure 36. The first vertex is along a vector that is the average of the face-vertex normals of the two faces that share the edge in question. It is at a distance from the edge defined by the user as the thickness of the *3D pipe*, which is the radius of the circumscribed circle of the polygon. We then create the next vertex which is the first vertex rotated at an *incremental angle* along a plane that has the edgevector as its normal. The plane is positioned on the edge depending on a *radius* parameter whose computation is explained shortly. The *incremental angle* depends on the number of segments in the cross-section such that :

$$angle = 360 \div Numberofsegments$$

While we create the vertices we will index them in the order of creation to facilitate the cleanup of the convex hulls created at a later stage.

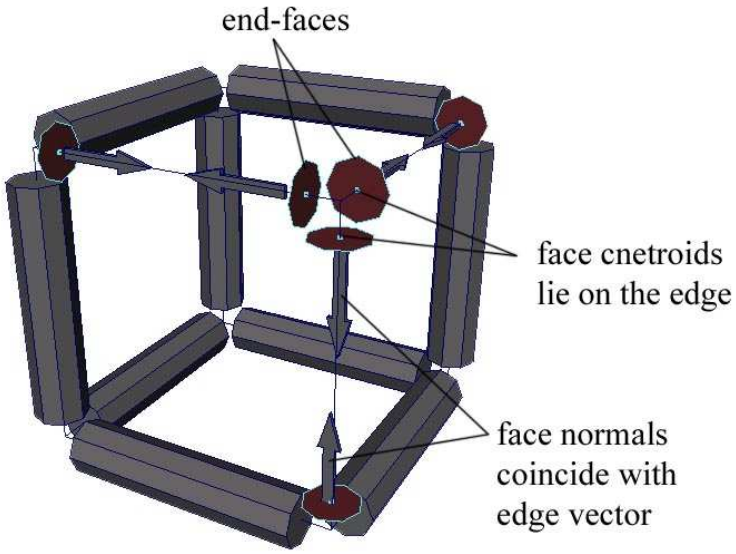


Fig. 35. The end-faces.

In the methodology we consider one vertex in the input mesh at a time and create such polygons for all the edges that originate from that vertex. The set of vertices generated by these polygons defines the joint shape for that vertex. To create the joint shape geometry we use a convex hull algorithm which creates a minimal 3D geometry shape for the set of vertices.

In the joint shape we have to make sure that the end-faces are fully intact in the convex hull geometry. As the convex hull creates a minimal geometry from the polygon vertices hence it is easy to see that the polygons should lie on the surface of an imaginary sphere centered on the vertex, as shown in Figure 37. The polygonal faces are positioned such that they are tangent to the sphere and at least two faces just touch each other.

The vertices of the polygons thus generated will be used to create the convex hull geometry. Any face that lies inside the sphere is either represented partially or not represented at all. Any face that lies outside such a sphere causes partial representation of the other faces, besides increasing the size of the joint shape, thus making it no longer a mini-

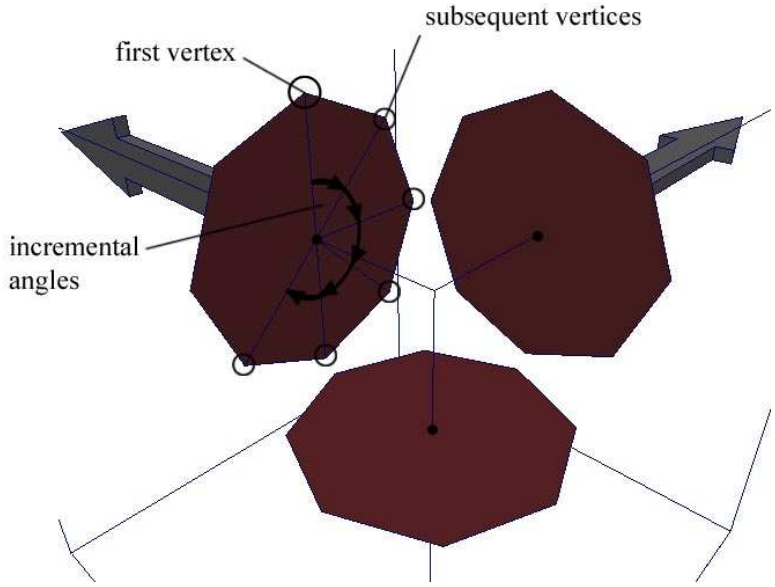


Fig. 36. The creation of individual vertices of the end-faces.

mal shape as desired. Now the question arises as to the computation of the *radius* of such a minimal sphere.

Consider an arbitrary *vertex-edge* configuration of the input mesh in two dimension, as shown in the Figure 38. We want a minimal sphere, as shown in the dotted line, such that we can place the polygonal faces on its surface, centered at each *edge-sphere* intersection, and also ensure no self-intersection between the faces. A self-intersection is most likely to occur between end-faces that correspond to adjacent edges which have the least angle between them. In the example in Figure 38b, we can visually tell that edges *e1* and *e2* are the most acute. The minimum radius will occur when faces *f1* and *f2* meet at the angle bisector of *e1* and *e2*. As *t* is a user defined thickness parameter, the minimum radius is :

$$minimalsphereradius = t \div tan(T/2)$$

We consider every pair of edges in the vertex to get the smallest possible radius corresponding to that vertex.

It should also be noted that since the thickness *t* is the radius of the circumscribed

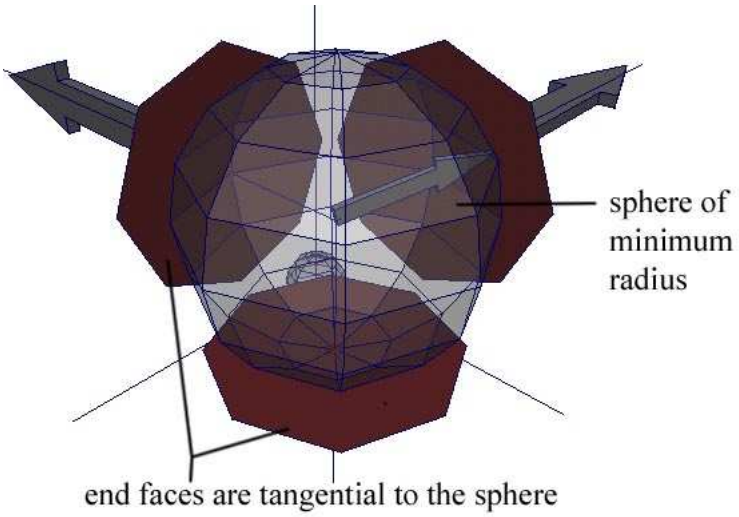


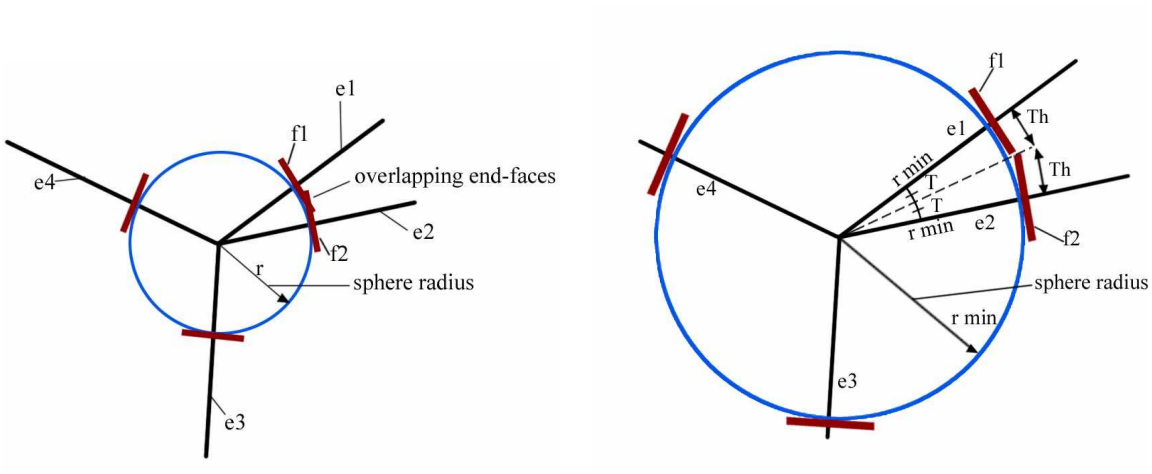
Fig. 37. The end-faces should be tangent to a minimal sphere centered at the vertex.

circle of the cross-sectional polygon, due to numerical discretization and approximation it is impossible to find the exact point of contact of two such circles. Therefore, to be on the safe side we will increase the radius by a small error factor.

So using this calculated radius for the minimal shape we place our cross-sectional polygonal end-faces on the edges. The vertices of the set of such end-faces generated for a vertex in the input mesh, are used to create a joint shape for that vertex using the convex hull algorithm.

III.2.2. Convex hull cleanup

The convex hull geometry created for the joint shape results in a triangulated mesh structure, as shown in Figure 39a. We cleanup the convexhull geometry so that we have one complete face representing the end-face of the user defined cross-section, for every edge originating from a vertex. We do this cleanup in two stages. In the first stage we use an algorithm that deletes the shared edge between two adjacent faces if the face normals



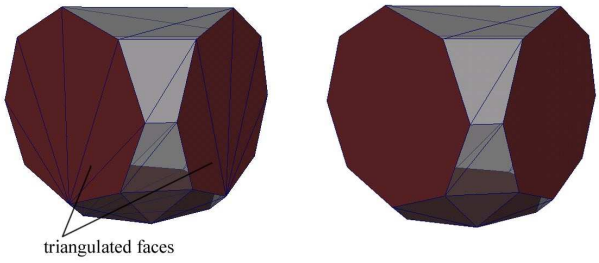
(a) Self-intersection

(b) Correct calculation of sphere radius to prevent overlapping end-faces

Fig. 38. Calculating *Tmax* to prevent self-intersection.

are same. Doing this cleans up only the end-faces in the convex hull geometry because we have created them to be planar. In the second stage we cleanup the rest of the geometry to get rid of the triangulation so that we have as many quadrilaterals as possible. It should be noted that this just creates a cleaner mesh structure that is good for subdivision, and is desirable; but it is not a necessary step to create the final mesh.

Consider a simple case of such a convex hull situation when the *3D pipe* has even number of segments in the cross-section. As mentioned before when we created the polyg-



(a) Triangulated convex hull

(b) End-faces cleaned

Fig. 39. Convex hull with the first step of cleaning.

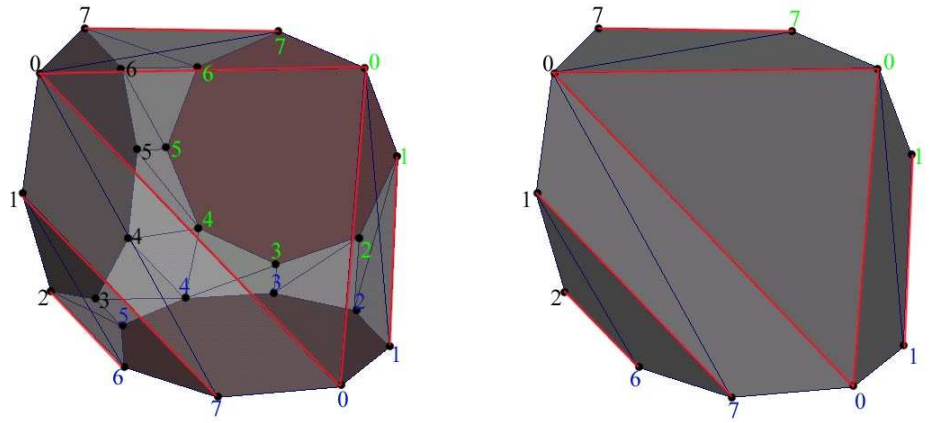


Fig. 40. Convex hull with the second step of cleaning.

onal *end-faces* we indexed the vertices in a circular order, as shown in Figure 40. When the convex hull is created using these vertices it is triangulated. Careful observation shows that, to obtain a clean mesh structure we want only the edges between all pair of vertices that have both even indices or both odd indices. Hence, we go through all the edges in the shape and delete the edges that connect even indexed vertices with odd indexed vertices. While doing so we will take care not to delete edges that belong to an *end-face*. This approach works very well when the number of segments in the *3D pipe* cross-section is even, as shown in Figure 41.

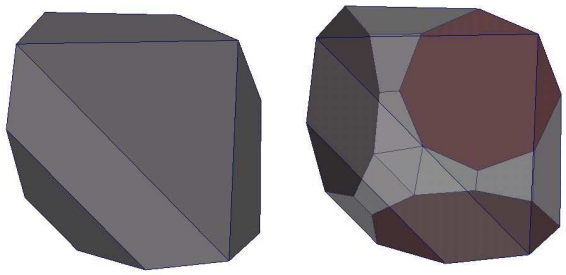


Fig. 41. Convex hull cleaned.

Consider a case where the number of segments in the cross-section is odd, as shown

in Figure 42. It is evident that there is no set pattern of connections. In this case we want some of all the edges, that connect pair of vertices with both even indices, odd indices and even-odd indices, to remain. Hence, we leave this situation out of the scope of this thesis.

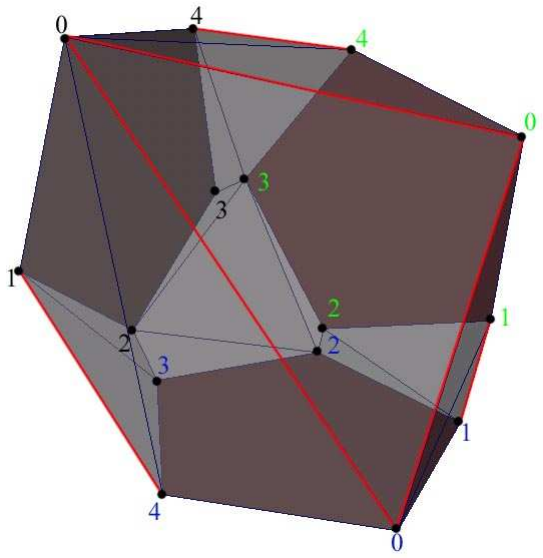
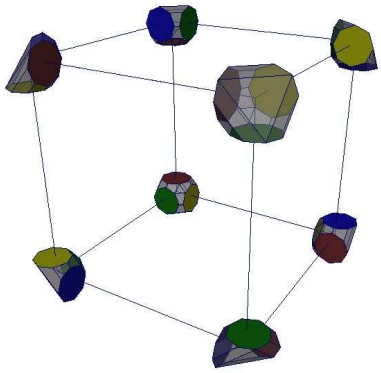


Fig. 42. Convex hull for odd segmented *3D pipe*.

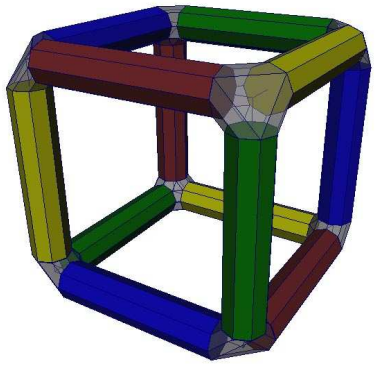
With even cross-sections if the mesh structure around the vertex is not symmetrical then this method does not guarantee a regular mesh structure for the joint shape every time. However, in most cases the results are acceptable.

III.2.3. Keeping track of connecting faces

We have to keep track of the pair of *end-faces* that correspond to the same edge in the convex hulls, so that we can create a handle between them to make the *3D pipe*. It should be noted that such a pair of faces belongs to two different joint shapes, as shown in Figure 43a. To do this we create an array of face-pointers with the array size twice the size of the total number of edges in the input mesh. This provides us with two array elements for every



(a) Matching faces corresponding to an edge in the convex hulls



(b) Connecting the matching faces

Fig. 43. Matching and connecting faces to complete the model.

edge, as we need two elements for the two *end-faces* that correspond to every edge. The DLFL implementation assigns a unique ID number to every edge in the input mesh. We use this unique ID to reserve two elements in the array for every edge. During the creation of the convex hull we store the face pointers for the pair of end-faces corresponding to an edge, based on its edgeID, in the array. The face-pointers are stored next to each other in the array and are indexed based on the edgeID. This system eliminates the need to search the pair of end-faces that need to be connected.

III.2.4. Connecting the faces

To connect the faces, we use the handle operation which is already implemented in the DLFL program. The handle operation connects two faces with a handle. It does so by systematically inserting edges between the vertices of the two faces. Thus, it can properly connect only faces with the same number of vertices. Since, this operation is already implemented, it is done trivially as we just need to go through the array and keep connecting the face-pointer entries next to each other, as shown in Figure 43b. Doing this for every element in the array completes the model.

CHAPTER IV

IMPLEMENTATION

The Wire and Column modeling system that we have developed can be used to convert any polygonal model to a complex high genus model where in the polygonal faces become holes and the edges become three dimensional like matchsticks. This modeling system is useful in modeling highly perforated ornate architectural elements and other similar objects.

The modeling system is implemented in C++ and included as an option in an existing 2-manifold modeling system called the *"DLFL mesh modeling system"* (DLFL is a type of data structure used by this mesh modeling system and stands for Doubly Linked Face List). We extend the capabilities of the DLFL modeling system [1], to incorporate the algorithm we have developed. Both systems currently run on *SGI-Iris, Linux* and *Windows* platforms. All of the interactive examples we have produced were run on an SGI-Linux Box. The modeling system allows the creation of models with texture coordinates and the resulting mesh can be exported to any commercial software package using the classical *"obj"* file format.

The implementation of both the Wire and Column modeling methods need an initial polygonal input model that either can be imported in the *obj* file format, which is compatible with several commercially available software packages; or the model can be constructed directly in the DLFL program. The modeling methods modify the input model as described in this thesis work, to output very high genus intricate models. The genus of the final model can be in the order of G 100 or more depending on the initial input mesh.

IV.1. Wire modeling

The *"Wire modeling"* scheme is incorporated in the *DLFL* modeling program as follows:

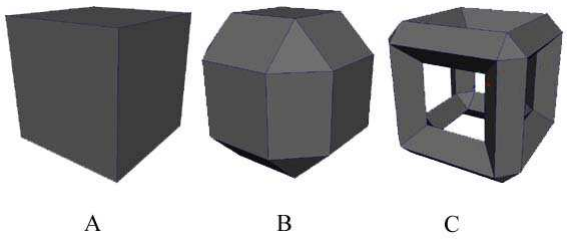


Fig. 44. Wire modeling implementation.

1. A modified Doo-Sabin subdivision [8] scheme is applied to the input mesh. The Doo-Sabin subdivision [8] scheme is modified such that the faces that are created for every edge and vertex are smaller and tighter and the face that corresponds to original faces is larger than in the original scheme [8]. This creates boundary faces that replace all original edges and vertices and also gives control of the width of such faces, as seen in Figure 44B.

2. Once the surface is subdivided, a nested surface is created that is offset by a user specified distance from the original surface. A similar surface is created in the Rind modeling system [2], but this implementation is slightly altered in Wire modeling. The vertices are offset in such a way that the faces that fill the sides of the holes, that are created using the handle-hole operation, are perpendicular to the faces of the original as well as the nested surfaces.

3. After achieving this mesh configuration, holes are punched through the faces that correspond to an original face in the Doo-Sabin [8] subdivision scheme. This is done

using the hole-handle operation, as seen in Figure 44C. When this is done for all such faces the final output model is completed.

The Wire modeling implementation lets the user control the thickness of the *wires* created in the model. The program calculates a maximum thickness that prevents any self-intersections in the resulting mesh and limits the user defined thickness to this calculated value if required.

IV.2. Column modeling

The *" Column modeling "* method is implemented as follows:

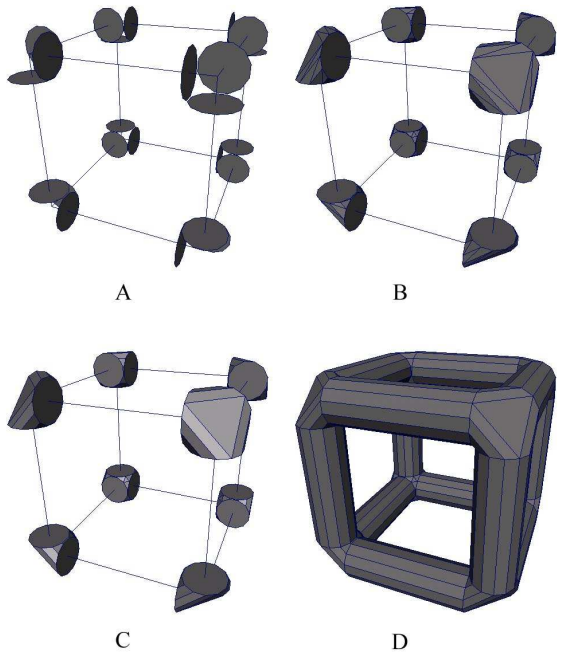


Fig. 45. Column modeling implementation.

1. *Column modeling* uses the input mesh as a reference to place the joints and the *3D pipes* in the model.

2. A joint shape is created corresponding to every vertex in the reference mesh. The algorithm loops through each vertex in the input mesh one at a time and considers every edge that originates from that vertex. For every such edge it computes a set of points that correspond to the vertices of a regular polygonal face that has its centroid on the edge, and normal coinciding with the edge vector, Figure 45A. The number of sides in the polygonal face is user defined. Such a set of points for every edge of the vertex in question is created and used to generate a convex hull, Figure 45B. The shape generated by the convex hull is the joint shape. Since the convex hull is triangulated it is cleaned up to make sure that there is only one face corresponding to every edge in the vertex, Figure 45C. A convex hull joint shape is created corresponding to all the vertices in the input mesh

3. The joint shapes thus created have a pair of faces for every edge in the model. These pairs of faces are identified and then *"create handle"* operation is used to make the *3D pipes* to join them, Figure 45D. Once this is done for every edge we have our completed output model.

In Column modeling the user can control the thickness, i.e. the radius of the circumscribed circle of the cross-sectional polygon of the *3D pipes*. The user can also control the number of segments in the cross-section of the *3D pipes*. Presently this method can generate models with only even number of segments in the cross-section for the *3D pipes*. The Column modeling implementation does not check the thickness of the *3D pipes* for self intersection. This can be implemented in the future.

IV.2.1. User interface

The User-Interface for the *"Wire and Column modeling"* mapping program is developed as an extension to the existing User-Interface of the DLFL mesh modeling system [1],

and is seamlessly integrated in to it. *Wire modeling* and *Column modeling* can be found under the *Crust modeling* menu in the *DLFL Mesh modeling* program. The User interface is very simple and easy to use. There is one button each for *"Wire modeling"* and *"Column modeling"*. There is a "Thickness" input option that is accessed by both the methods. There is a *"Cross-Section segments"* input option which is used only by Column modeling. Screen captures of the user interface of the implementation of Wire and Column modeling methods are shown in Figurs 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56.

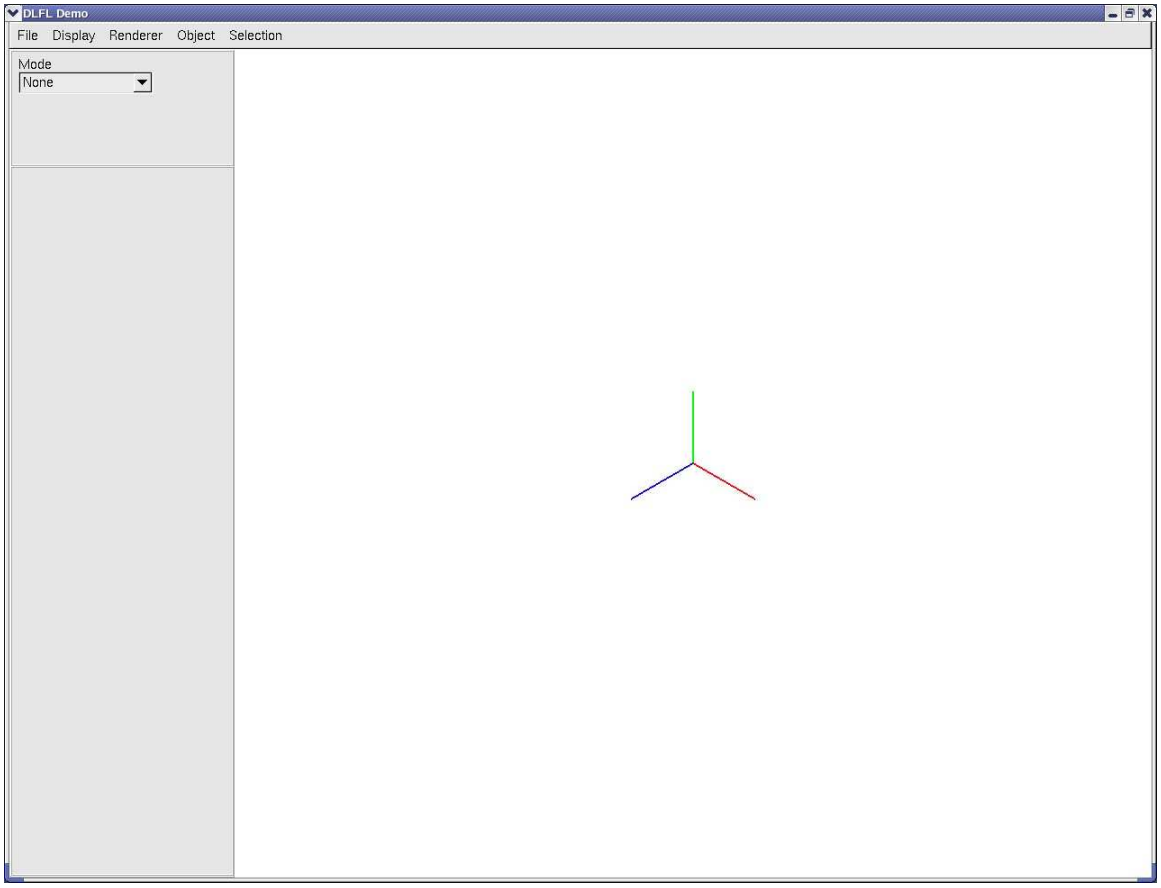


Fig. 46. The DLFL mesh modeling user interface.

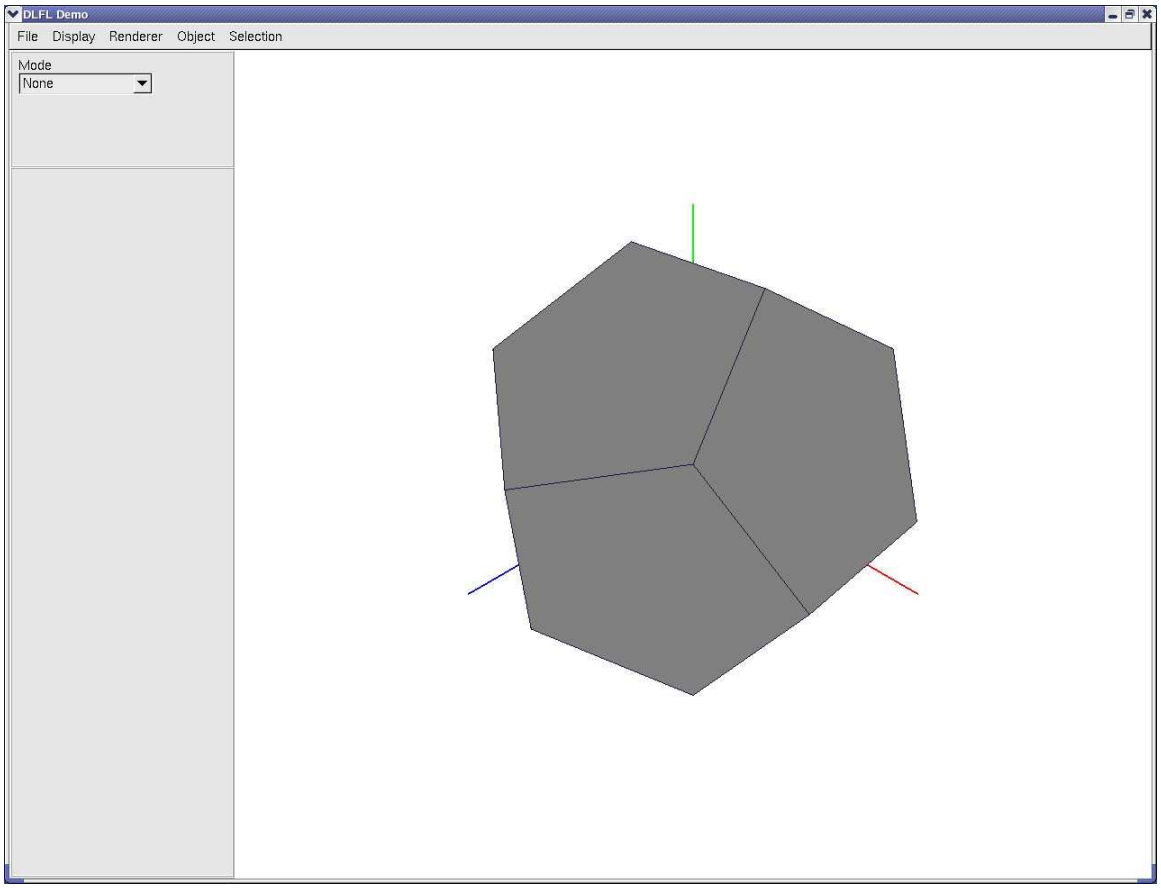


Fig. 47. A dodecahedron model imported in the *.obj* file format.

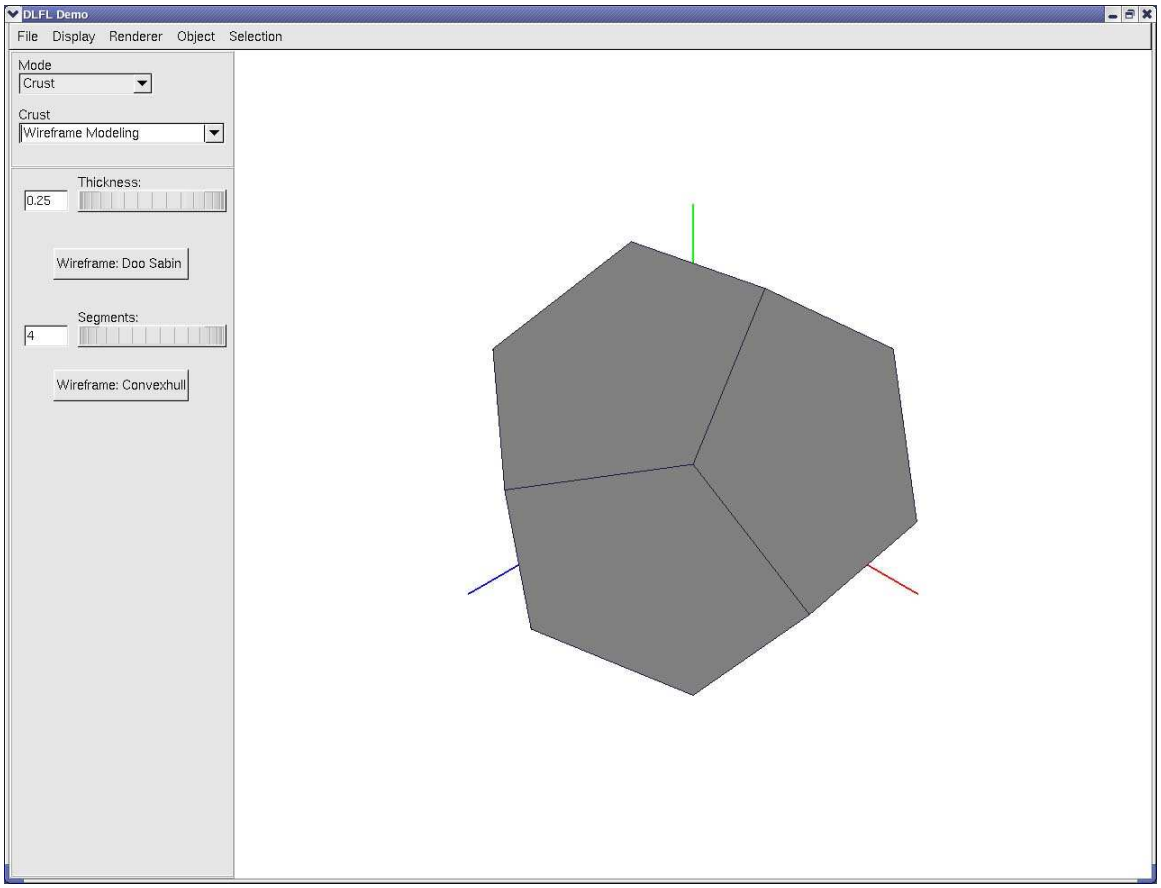


Fig. 48. *Wire and Column modeling* is under the *Crust modeling* menu.

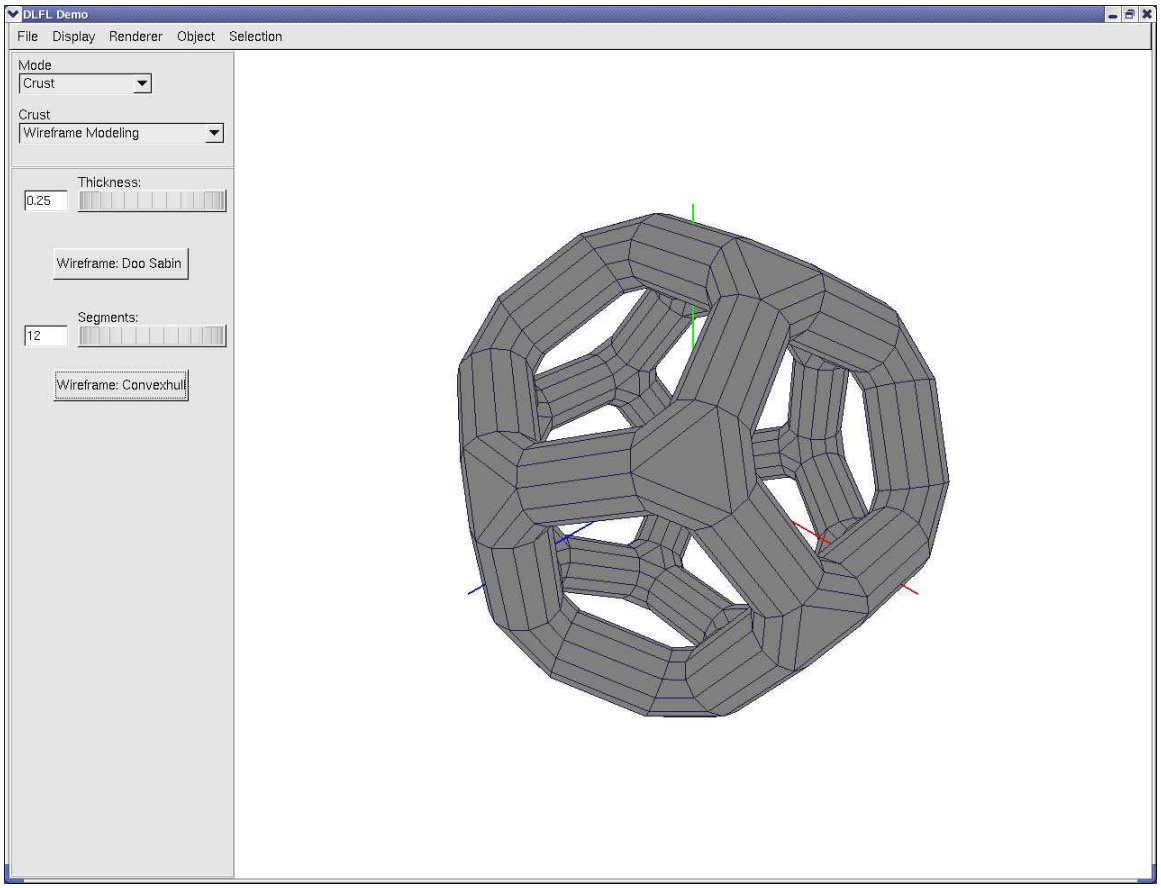


Fig. 49. Column modeling applied to the model with a thickness of 0.25 and number of segments as 12.

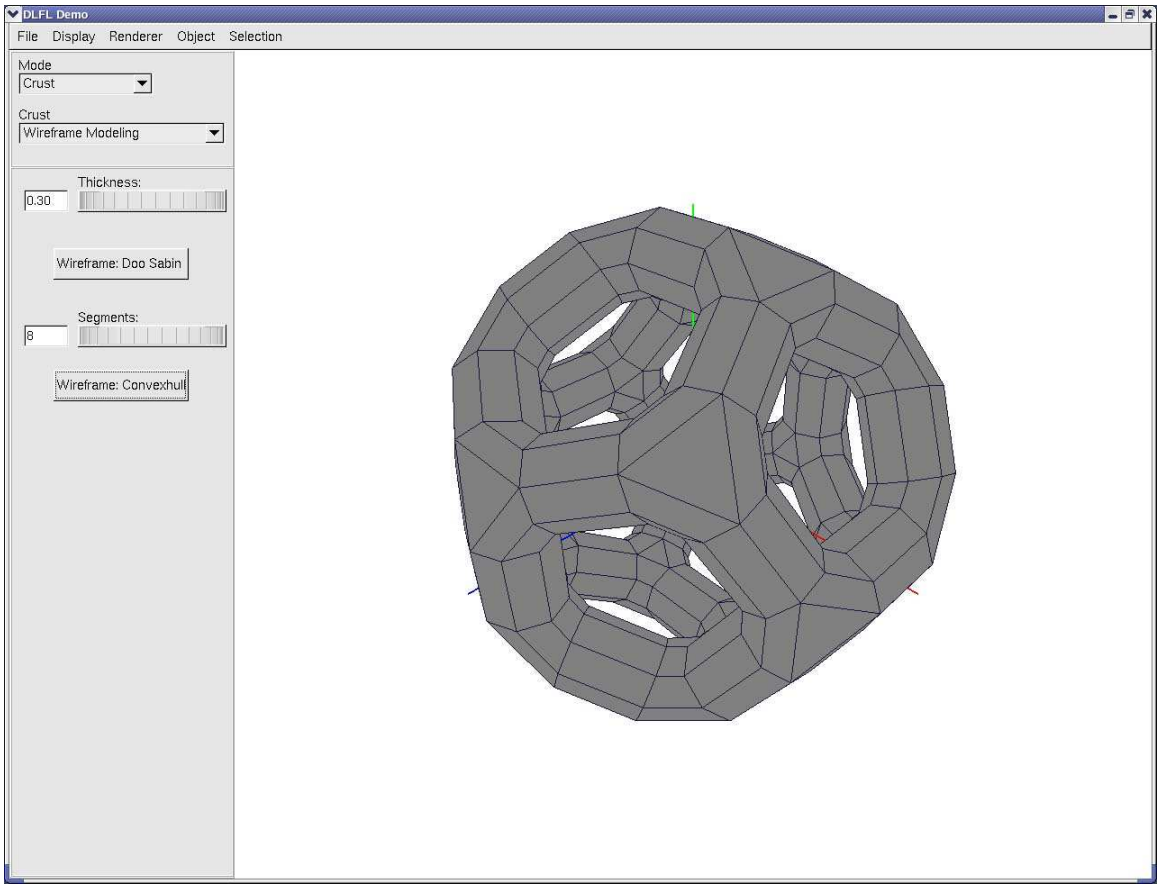


Fig. 50. Column modeling with thickness 0.30 and number of segments 8.

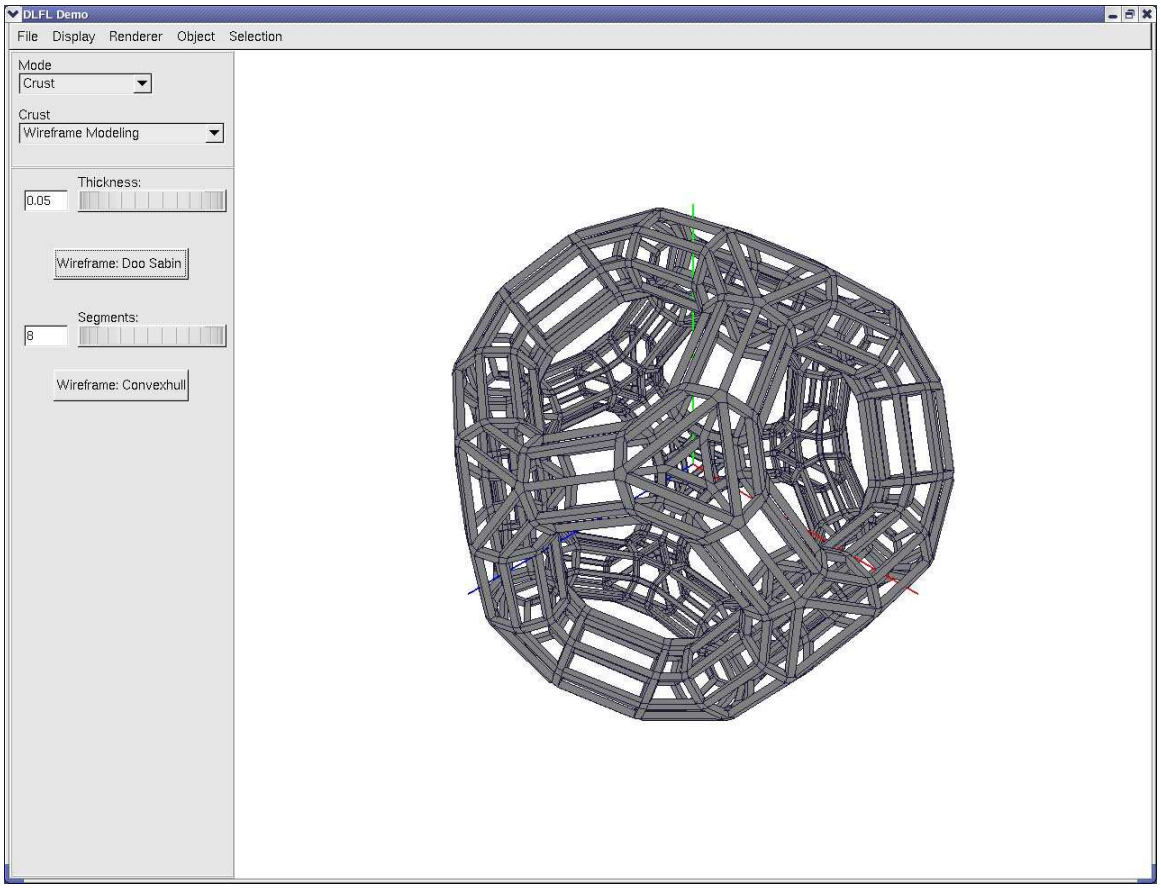


Fig. 51. Wire modeling with thickness 0.05 applied on top of the previous column model.

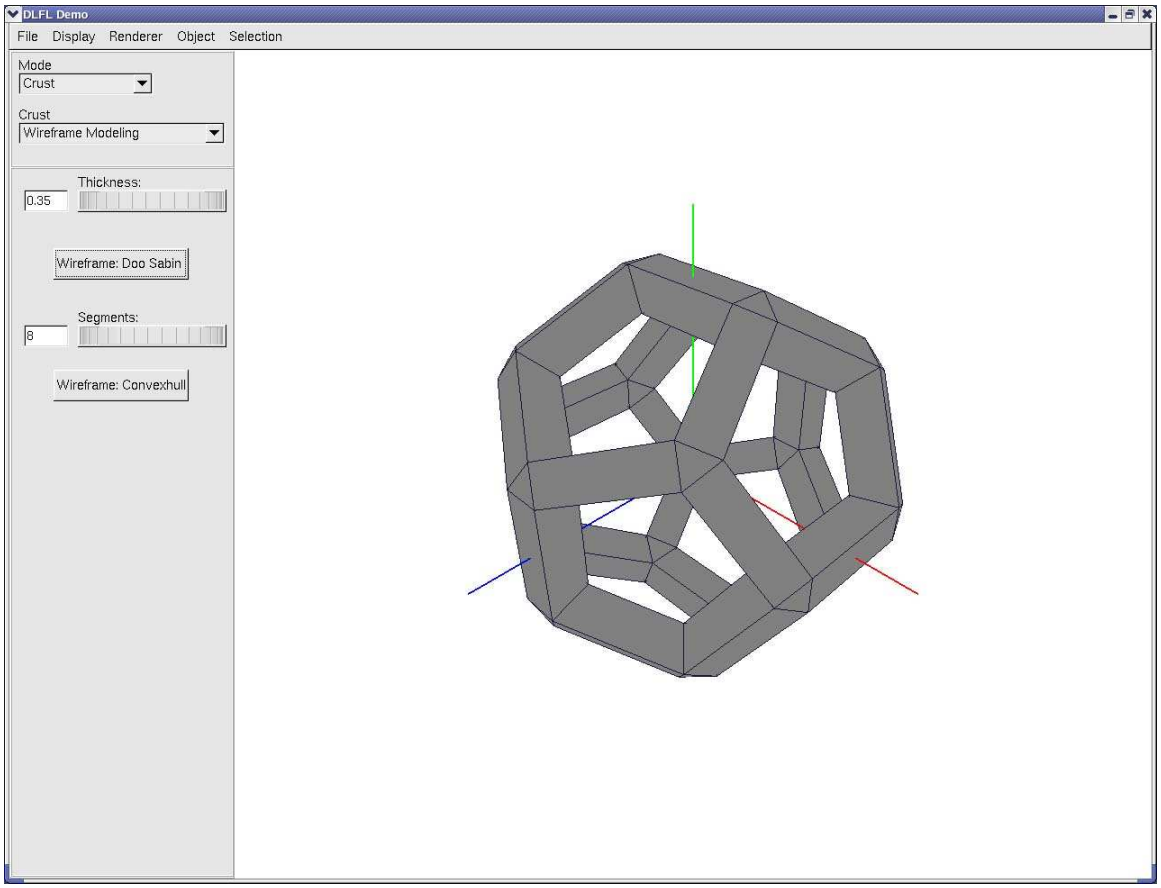


Fig. 52. Wire modeling with thickness 0.35.

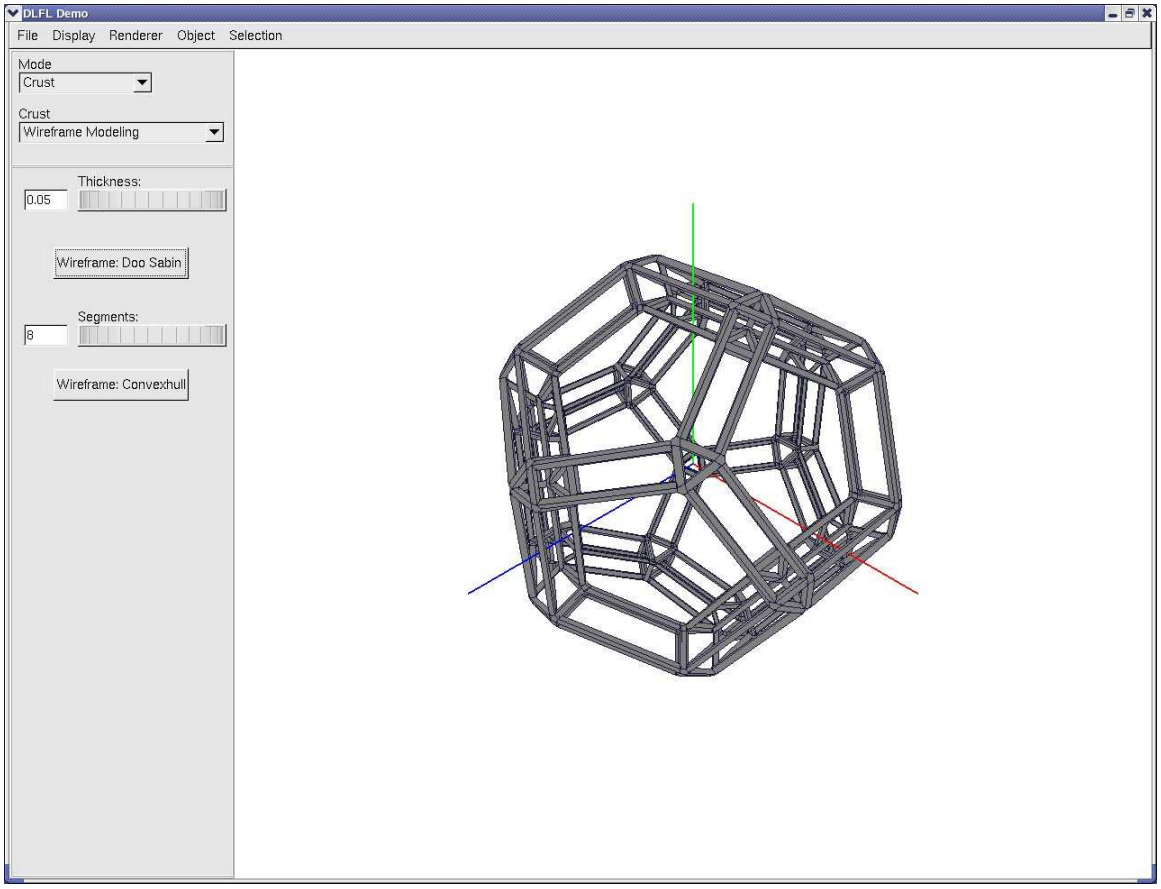


Fig. 53. Wire modeling of thickness 0.05 applied on top of Wire modeling with thickness 0.35.

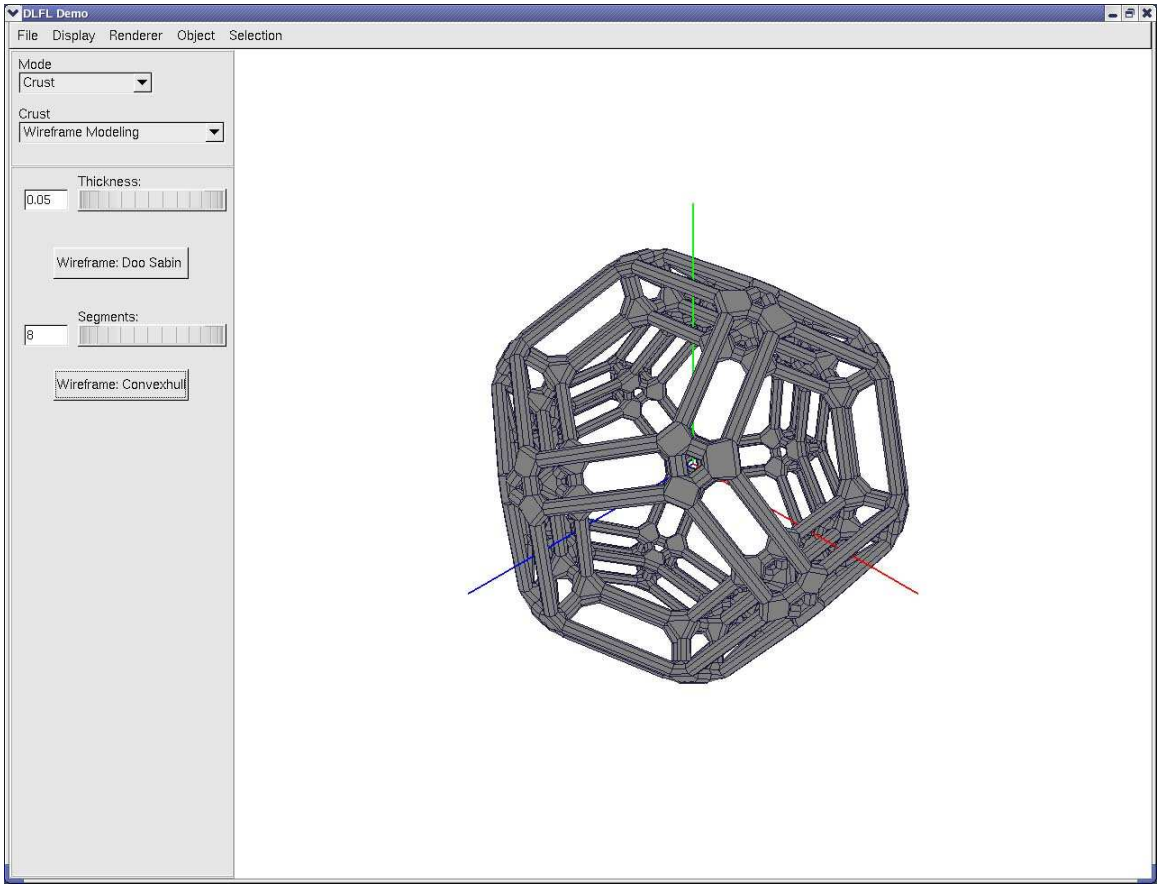


Fig. 54. Column modeling of thickness 0.05 and segments 8 is applied on Wire modeling of thickness 0.35.

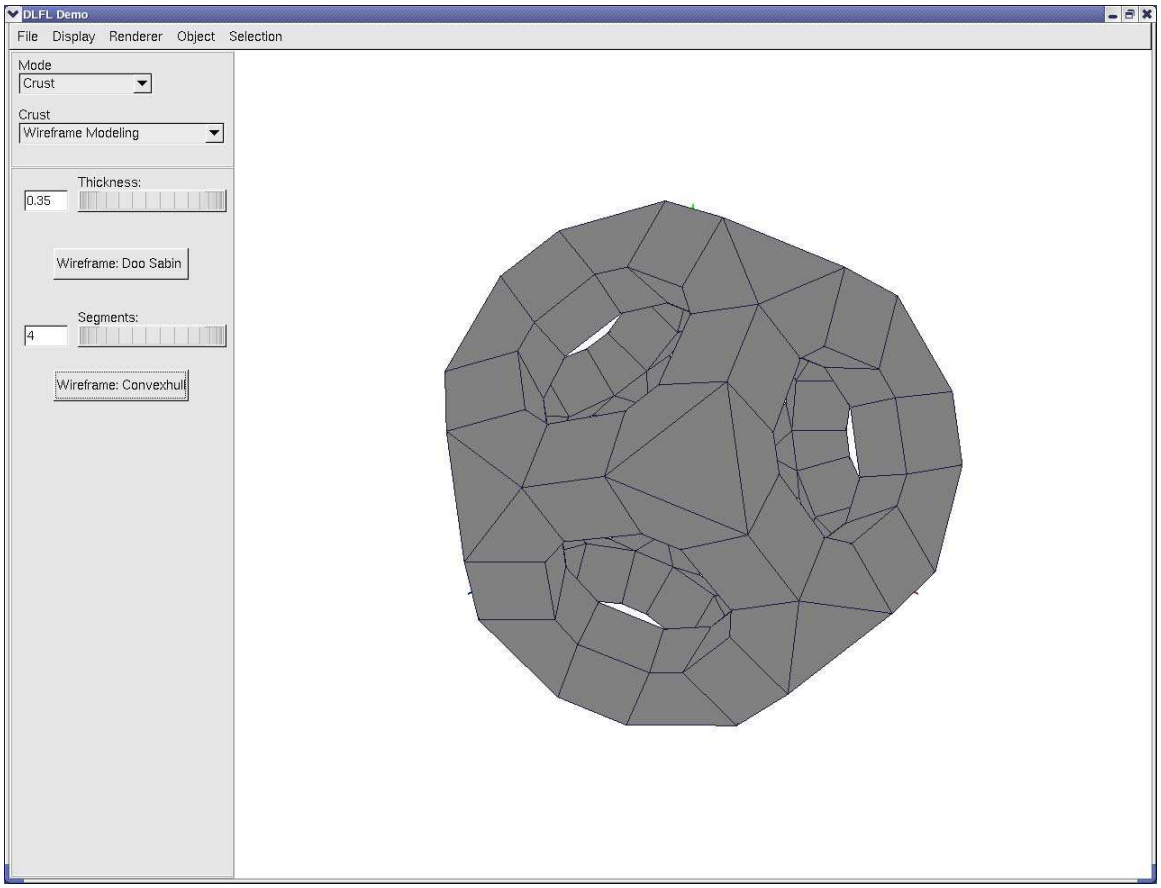


Fig. 55. Column modeling of 4 segments and thickness 0.35. Compare with Wire modeling of similar parameters.

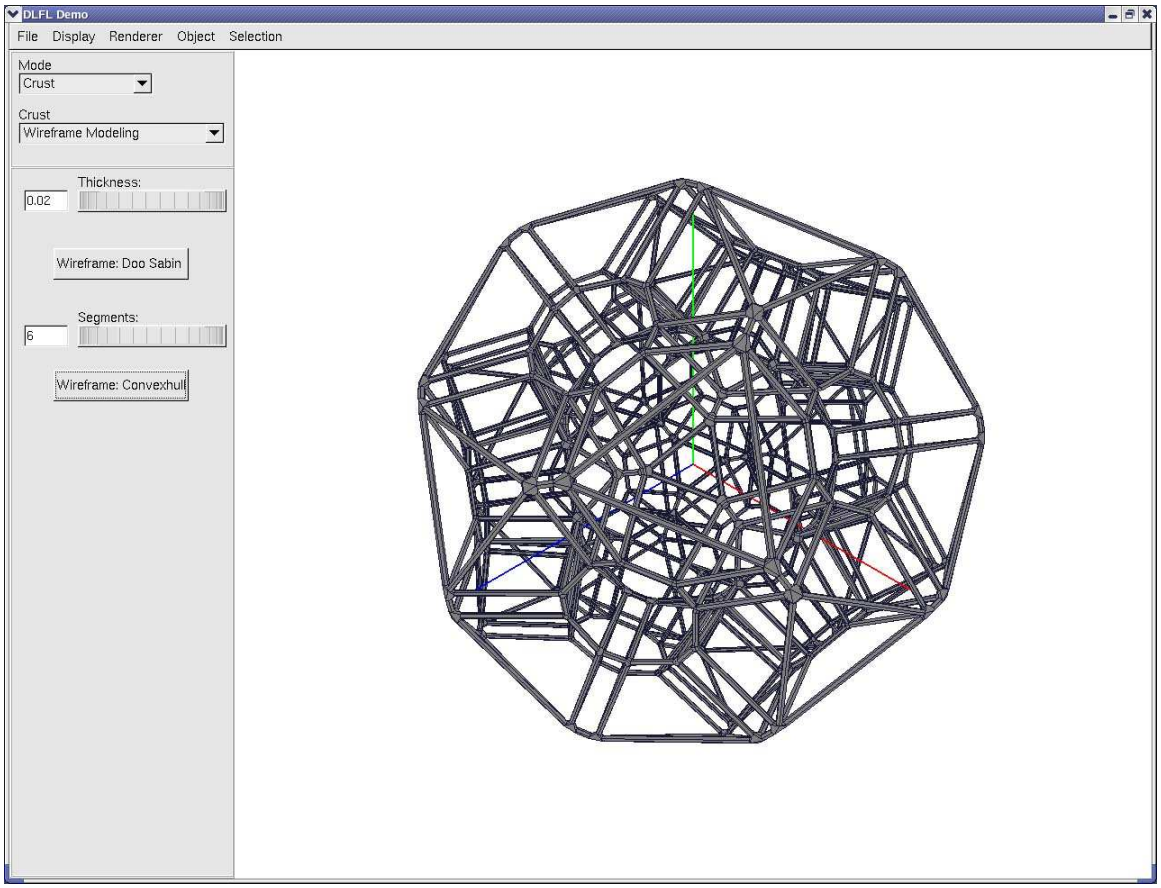


Fig. 56. Column modeling applied twice.

CHAPTER V

USABILITY ANALYSIS OF THE TWO METHODS

In this chapter we will discuss the features and drawbacks of the methods presented in this thesis work. We will also discuss some suggested guidelines to achieve the best results form the implementation.

V.1. Wire modeling usability

V.1.1. Features

- This method is used to create very high genus shapes by sculpting an input mesh.
- The output mesh looks like a shape that is a framework of wires or *3D pipes* joined together.
- The *3D pipes* have a rectangular cross-section.
- The user can specify the thickness of the *3D pipes*. All the *3D pipes* in the model have a uniform thickness.
- The method makes sure that there is no self-intersection in the output mesh provided there is none in the input mesh to begin with, and may change the user input thickness to do so.
- It can take any shape as an input mesh which can be imported as an *.obj* file, which is a very popular model file format.
- It is very fast when compared to other techniques that may be used to create identical high quality models.

- The mesh geometry generated in the output mesh is very clean and the mesh is predominantly made up of quadrilaterals, which is good for subdivision schemes.
- This method is faster than the Column modeling method for similar input meshes and modeling parameters.

V.1.2. Limitations

- This method creates models wherein the *3D pipes* can have only rectangular cross-sections.
- The implementation of this method cannot handle vertices with valence less than two.
- The user has no control on the orientation of the *3D pipes* around the edge axis.
- The user cannot have different thicknesses of *3D pipes* in the same model.

V.2. Column modeling usability

V.2.1. Features

- This method is used to create very high genus shapes by modifying an input mesh.
- The output mesh looks like a shape that is a framework of assembled tube-like building blocks or *3D pipes* that are joined together with joint shapes.
- The *3D pipes* can have a user defined even number of segments
- The user can specify the thickness of the *3D pipes*. All the *3D pipes* in the model have a uniform thickness.
- The method has no restrictions on the vertex valence in the input mesh.

- It can take any shape as an input mesh that can be imported in the program as an *.obj* file, which is a very popular model file format.
- It is very fast when compared to other techniques that may be used to create identical high quality models.
- The mesh geometry generated in the output mesh is predominantly quadrilateral, which is good for subdivision schemes.

V.2.2. Limitations

- This method is slower than the Wire modeling method for similar input meshes.
- The method does not check for self intersection hence the output mesh can have self intersections. The onus is on the user to find a suitable thickness for the *3D pipes* which does not result in self intersection.
- The joint shapes are complex and may not always have a clean mesh structure as desired. The method works best for a symmetrical or uniform input mesh structure.
- The user has no control on the orientation of the *3D pipes* around the edge axis
- The user cannot have different thicknesses of *3D pipes* in the same model.

V.3. Usability tips

V.3.1. Use of subdivision algorithms

In both methods the output mesh is heavily dependant on the input mesh structure, as the *3D pipe* framework strictly follows the edge configuration in the input mesh. It is thus implied that the more articulated the input mesh is, the more beautiful the final results will

be. Moreover, many real life objects like stone screens, furniture etc. have very geometric designs. We therefore suggest that the several in-built subdivision algorithms be used to articulate an input mesh. Subdivision algorithms are typically used for mesh refinement to add detail and smoothness to a model. It is here that we suggest taking advantage of these algorithms from a purely visual and artistic point of view. Subdivision algorithm are based on highly geometric principles. Thus it turns out that it is also very useful to make the mesh structure very geometric and beautiful. The following subdivision algorithms are provided in the DLFL mesh modeling system

1. Catmull Clark
2. Doo Sabin
3. Honey Comb
4. Corner Cutting
5. Root
6. Simplest
7. Vertex Cutting
8. Pentagonal
9. Dual

These subdivision algorithms can be applied in any permutation and combination to achieve an interesting and articulated mesh structure. The following suggestions should be noted, although they are not necessary for obtaining the best results.

The input mesh should be as sparse as possible, i.e the polygon count should be low, and the vertex valence should be typically between three and five. Moreover, quadrilateral faces are preferred as they respond to subdivision better than other polygons.

Although there is no limit on the number of subdivisions that can be applied to a mesh, it is suggested to keep the number of subdivision between three and five. Very few iterations of subdivisions does not articulate the mesh well enough to be beautiful, however too many iterations creates a very dense mesh, which is not suitable for either of the methods presented in this thesis.

Based on the above observations the recommended work flow for column and Wire modeling would be as follows:

- Import an input mesh of the desired shape, preferably with a low polygon count.
- Apply several different combinations of subdivision algorithms to articulate the mesh structure.
- Apply Wire modeling or Column modeling as desired to obtain the final output mesh.

The above observations are purely based on repeated usage of the program and are stated without proof, experimental or otherwise. To study these observations and come up with a concrete methodology for the discussed work flow, is suggested as a possibility for future work.

V.3.2. Column modeling

Column modeling works best for a uniform input mesh structure. If the mesh structure has vertices with a wide range of valences then the joint shapes generated are of widely different sizes. For high valence vertices, or in case of vertices where the edges are too close to each other, the joints become very big compared to other low valence vertices, and

look ugly. If the mesh is uniform then all the joints formed are of similar size and the model looks much better.

Column modeling is more advantageous due to the fact that one can increase the number of segments in the cross-section to give a more rounded look. To take advantage of this feature it is better to use larger thickness for the *3D pipes* in the model so that one can see the roundness of the *3D pipes*. Thus, it is better to use input meshes that are not dense. With repeated use and experimentation it is found that Column modeling is more suitable for shapes that are not very organic, like architectural models and geometric sculptures.

V.3.3. Wire modeling

Wire modeling produces beautiful results with almost any kind of input mesh. If the input mesh is very dense then it is better to use Wire modeling over Column modeling because it is faster. Moreover, the *3D pipes* formed are small and their cross-section do not add any considerable visually quality to the model, thus using Column modeling is unjustified.

One can also use both the methods in succession. The suggested order would be to first do the Column modeling and then the Wire modeling. Column modeling should be preferably done with large thickness and then Wire modeling should be done with a small thickness. The Column modeling in this case dictates the over all shape of the model as against the initial input mesh doing so. One can even do more than one level of Column or Wire modeling, but with reducing thicknesses.

In this chapter both the methods presented in this thesis were discussed from a usability point of view. The suggestions for better results presented are not rules, but guides which may help produce better looking models. One should not be restricted to these suggestions, and many interesting models can be made without following these guidelines.

CHAPTER VI

RESULTS

As a proof of concept, we have created all final images and animations in Maya, as shown in Figures 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67. The usability of the system was tested in a graduate level computer graphics course. Students with diverse backgrounds including art, architecture and computer science took the course. All the students, regardless of their background, were able to successfully create very high genus models using a variety of input meshes. Following are some significant results that have been achieved, by using the modeling methods presented in this thesis work:

1. Very high genus models are created.
2. The modeling methods are automated and take significantly less time than traditional methods.
3. The models created are always 2-manifold and hence physically realizable.

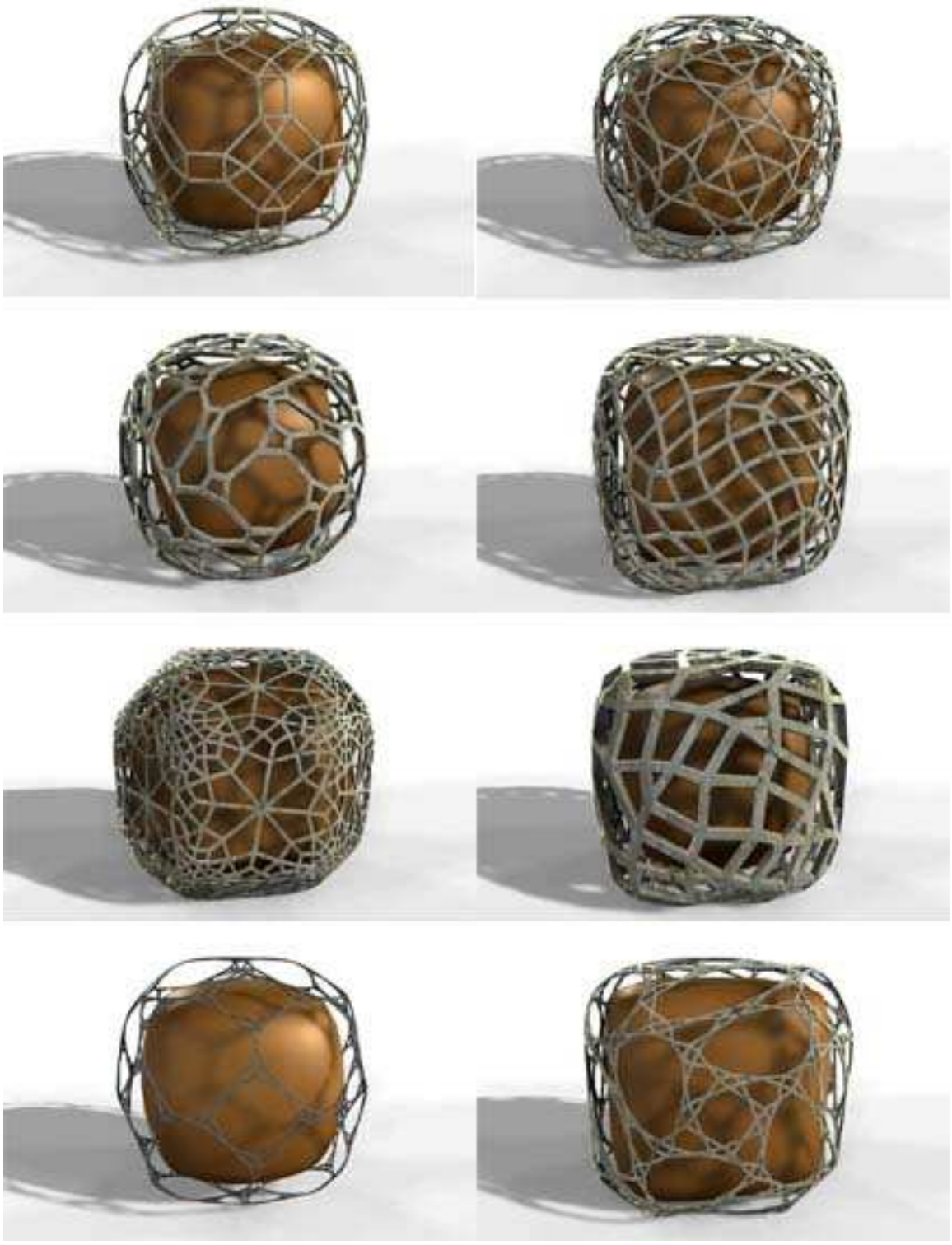


Fig. 57. Wire modeling results : Cubes with different meshes created using permutations of subdivision schemes.

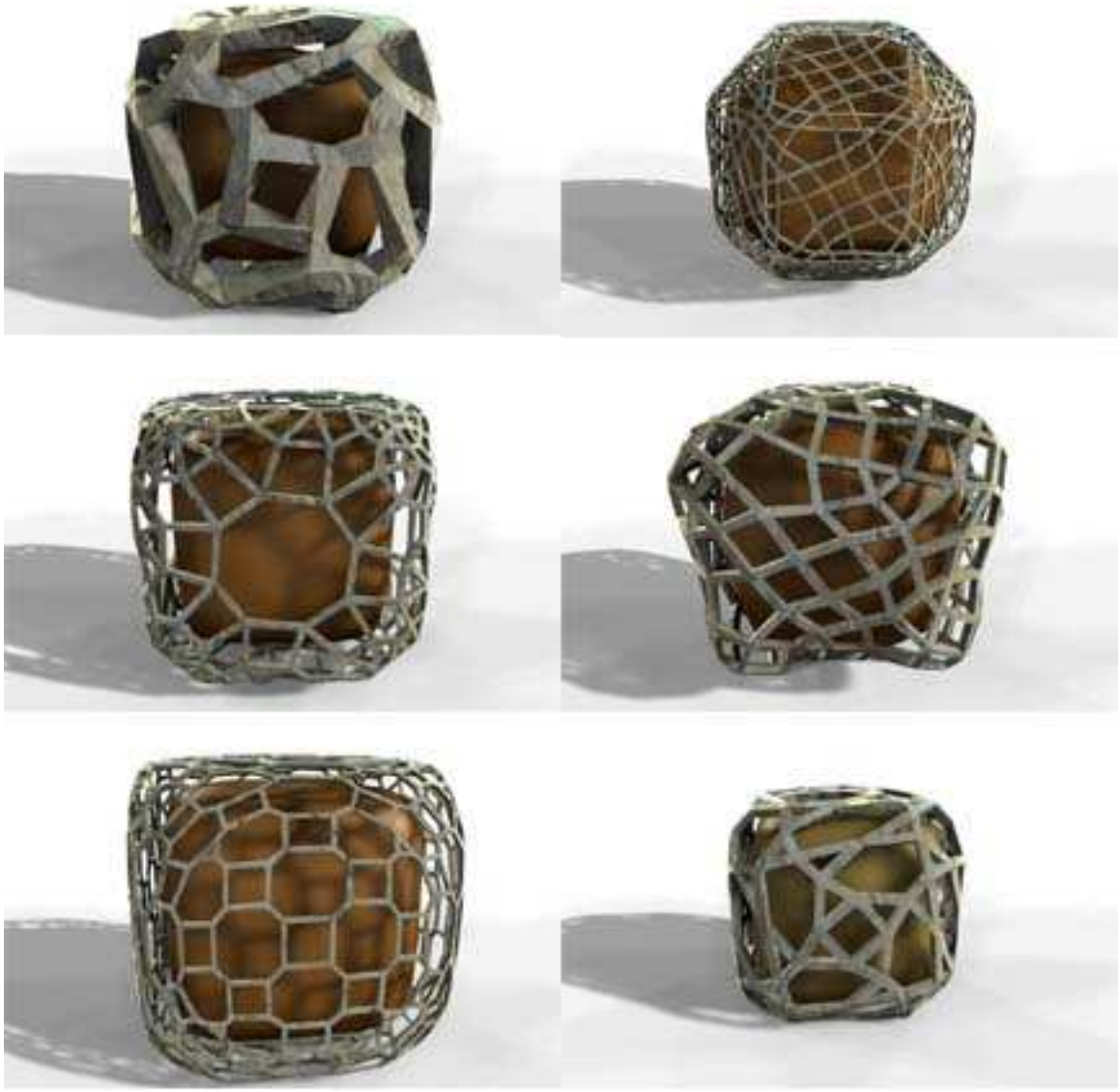


Fig. 58. Wire modeling results : Cubes with different meshes created using permutations of subdivision schemes.

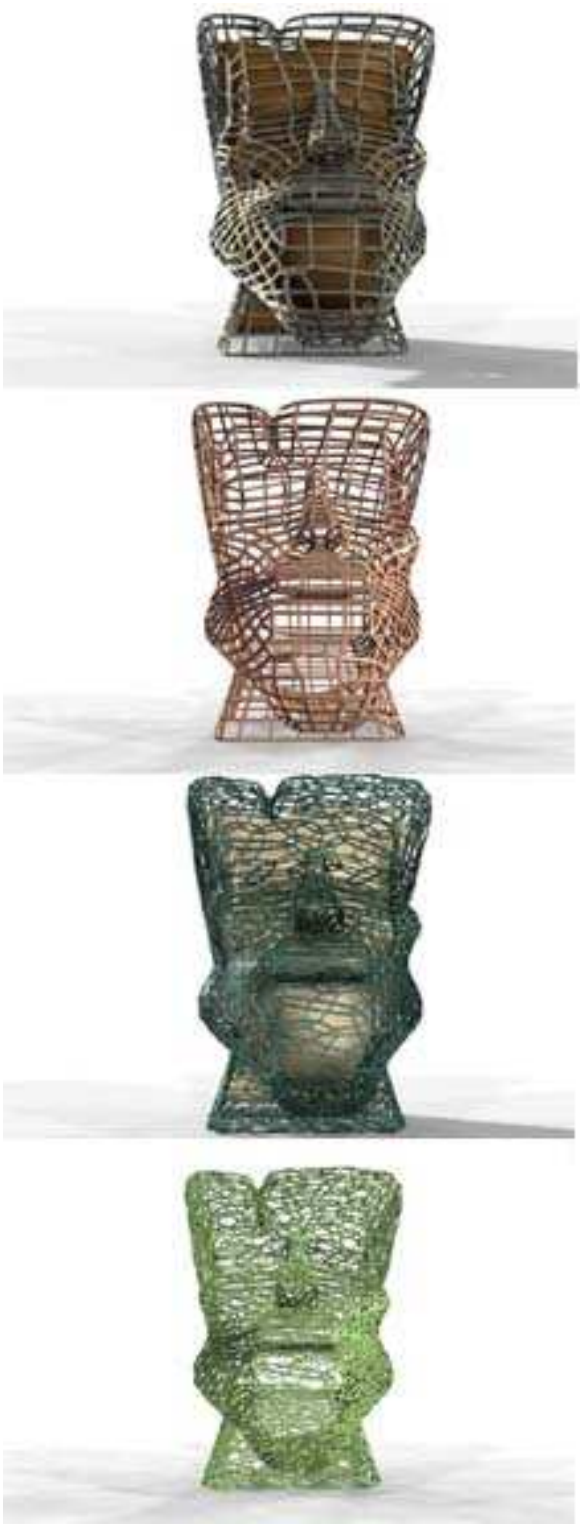


Fig. 59. Wire modeling results : Caricature of Arnold with different meshes created using permutations of subdivision schemes.

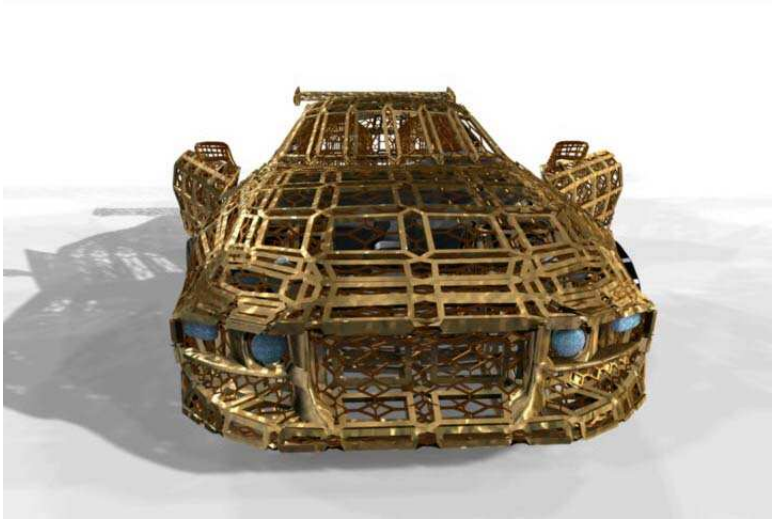


Fig. 60. Wire modeling results : Car.

Fig. 61. Wire modeling results : Caricature of Humphrey Bogart with different meshes created using permutations of subdivision schemes.

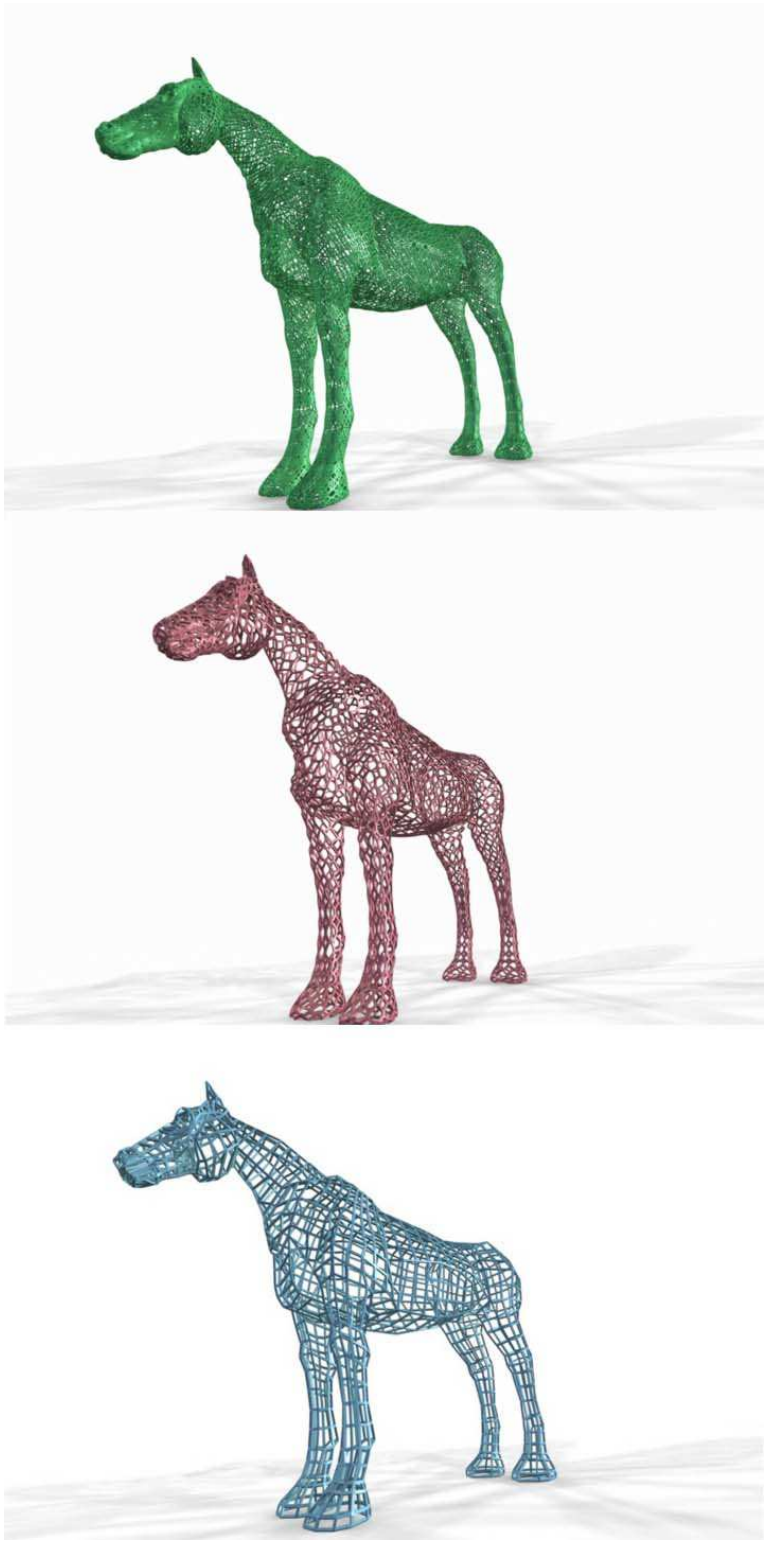


Fig. 62. Wire modeling results : Horse with different meshes created using permutations of subdivision schemes.

Fig. 63. Wire modeling results : Rabbit with different meshes created using permutations of subdivision schemes.

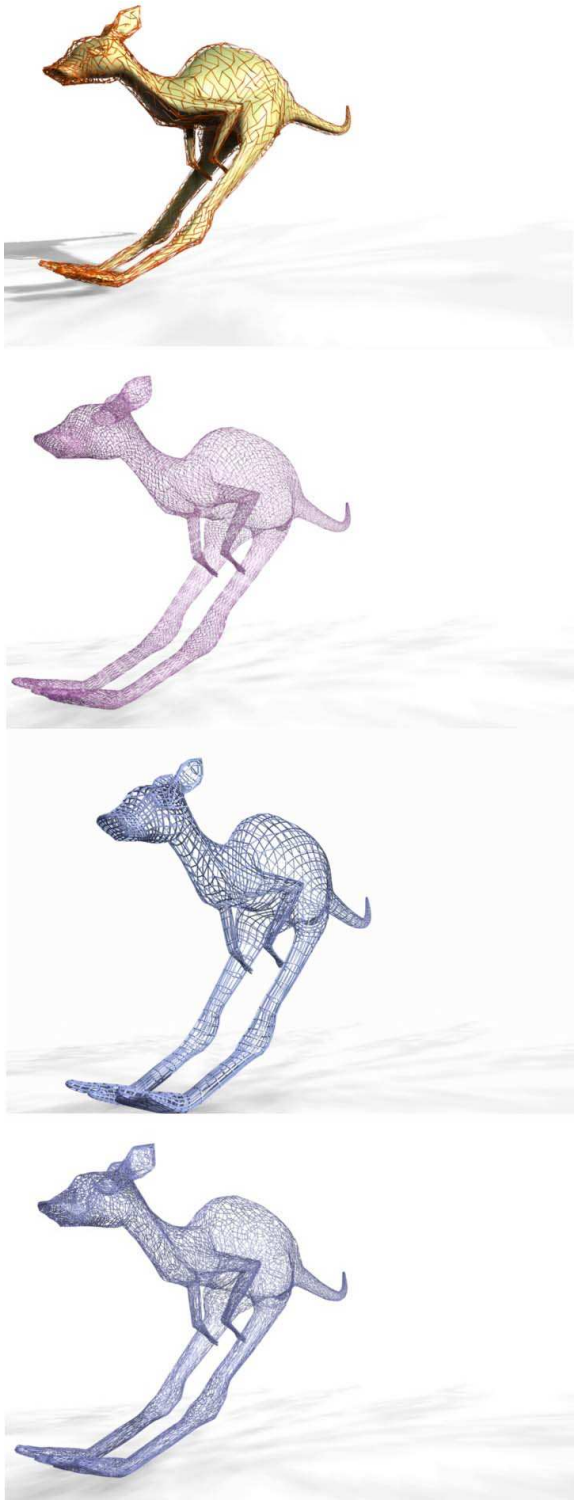


Fig. 64. Wire modeling results : Kangaroo with different meshes created using permutations of subdivision schemes.

Fig. 65. Column modeling results : Eiffel Tower.

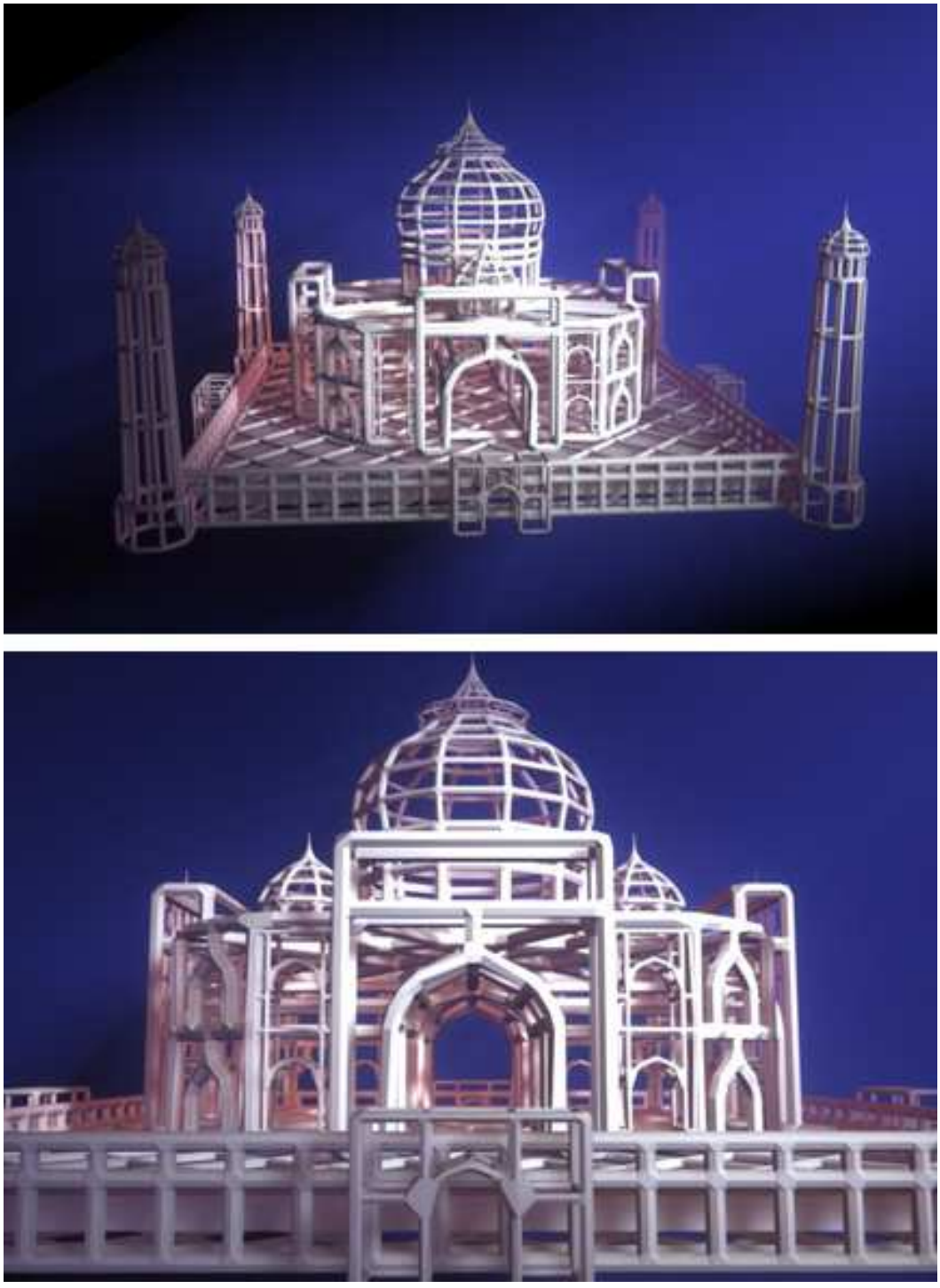


Fig. 66. Column modeling results : Taj Mahal.

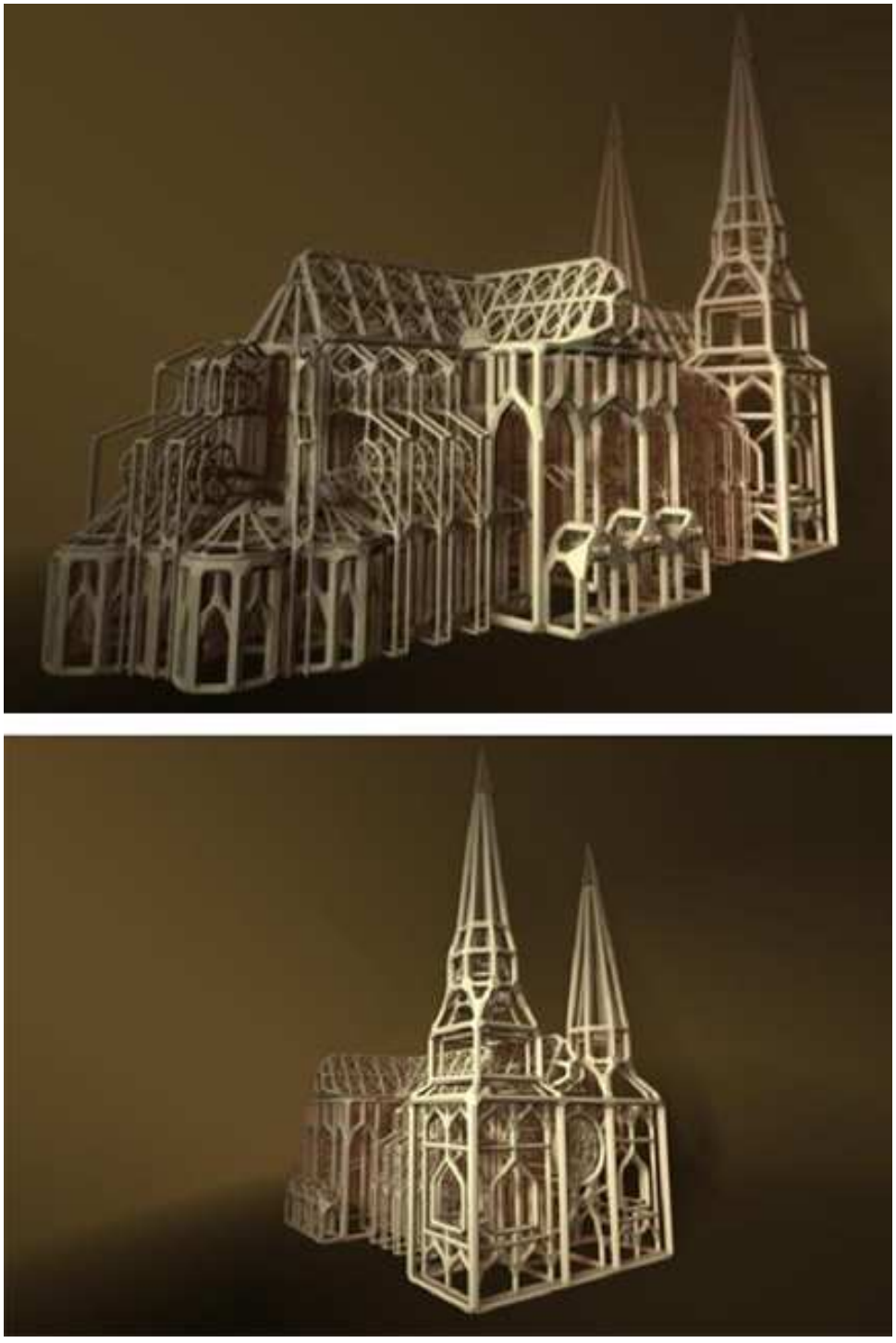


Fig. 67. Column modeling results : Cathedral.

CHAPTER VII

CONCLUSION AND FUTURE WORK

VII.1. Conclusion

The Wire modeling system proposed in this thesis work creates extremely high genus models from any polygonal mesh. We have presented two methods to achieve this goal, one which creates only square cross-sectional models (*Wire modeling*), while the other creates variable cross-sectional models (*Column modeling*). It is thus implied that it is also possible to create square cross-sectional models using Column modeling, but the difference is in the orientation and joints of the *3D pipes*; besides the fact that it is computationally expensive. The input polygonal mesh required for these methods need not be high resolution. In fact a sparse model with low polygon count is preferred. Subdivision schemes can be used to articulate the surface mesh to make it beautiful. Hence, with very little effort on the part of the modeler one can create very complicated and beautiful models. Since the program output is in the *obj* file format, these models can be easily incorporated into commercially available software for rendering and animation purposes.

VII.2. Future work

There are many directions in which possible future work can be carried on with respect to the work presented in this thesis.

One set of work can be directed towards overcoming the limitations of the two methods, to make them more robust and versatile. Some examples would be:

- Extend Column modeling to be able to implement odd cross-sectional models instead of just even cross-sections, and also have non-convex polygon cross-sections, like

star-shapes, for the *3D pipes*.

- The Column modeling method may include corrective measures to prevent self-intersection in the model.

- The joint shapes created in Column modeling depend on the edge configuration of the vertex; this may result in a joint shape mesh that is not very clean or symmetrical. One possible solution would be to have spheres as joint shapes.

- Wire modeling can be made more robust so that it handles all vertex-valence configurations.

- Both methods presently allow only one thickness of *3D pipes* throughout the model. It would be very helpful to have user control and provision for different thicknesses.

The methods presented in this thesis are dependent on the mesh structure of the input mesh. Subdivision algorithms are suggested to articulate the input mesh for more beautiful results. There is a large scope of work in investigating available subdivision algorithms from an artistic point of view. New subdivision schemes can also be created, which generate artistic mesh structures.

Interesting mesh structures can also be generated using three-dimensional and two-dimensional geometry. Instead of using subdivision schemes or manually building a mesh, one can copy the structure of a particular part of the mesh and spread it to the whole mesh based on spatial symmetry. This could be an interesting avenue for future work.

Another way of creating interesting mesh structures may be to generate them based on a 2D image. One could read in an image, digitize it, and then repeat it on the whole model based on symmetry, or some other logic.

The two methods presented in this thesis create a *3D pipe* for every edge in the model. It would also be interesting if one can create *3D pipes* for selective edges based on some

symmetry logic, such that there is more solid surface in the model.

It would also be interesting to find alternative modeling methodologies to generate such shapes that does not have the limitations of the methods presented here.

REFERENCES

[1] E. Akleman, and J. Chen, “Guaranteeing 2-Manifold Property for Meshes by Using Doubly Linked Face List,” *International Journal of Shape Modeling*, vol. 5, no. 2, pp. 149-177, 2000.

[2] E. Akleman, V. Srinivasan, and J. Chen, “Interactive Rind Modeling,” *Proceedings of the Shape Modeling International 2003*, pp. 23-23, Seoul, Korea, May 12-15, 2003.

[3] H. Aslaksen, “Mathematics in Art and Architecture,” Retrieved January 23, 2004, from http://www.math.nus.edu.sg/aslaksen/teaching/math-art-arch.shtml.

[4] T. Bleackley, “Art in Iron,” Retrieved Febuary 25, 2004, from http://www.artiniron.-co.+nz/page_balustrades.htm.

[5] P. Bourke, “Autocad to Interactive OpenGL,” Melbourne University School of Architecture, Retrieved Febuary 12, 2004, from http://astronomy.swin.edu.au/p̃bourke/-õpengl/autocad/.

[6] P. Cech, “Generalized Solids of Revolution,” Retrieved January 15, 2004, from http://ptr.host.sk/dw/modeling_intro_en.php#chapter_2.1.2.

[7] S. Craft, “Wooden Screens Manufacturer,” Retrieved Febuary 22, 2004, from http://-www.sajjacraft.net/wooden-screens.html.

[8] D. Doo, and M. Sabin, “Behavior of Recursive Division Surfaces Near Extraordinary Points,” *Computer-Aided Design*, Vol. 10, No. 6, pp. 356-360, September 1978.

[9] P. Everett , “WireShader,” Retrieved September 12, 2003, from http://www.tools4d.-com/Wireshader/wireweb/wireshade_manual.pdf.

[10] “FreeFoto Free Pictures,” Retrieved Febuary 22,2004, from http://www.freefoto.-com/index.jsp.

[11] W. Gozdz, and R. Hoyst, “High Genus Periodic Gyroid Surfaces of Nonpositive Gaussian Curvature,” *Physical Review Letters*, Vol. 76, no. 15, pp. 2726-2729, April 1996.

[12] C. K. Haluska, W. T. Gozdz, H.G. Dobereiner, S. Forster, and G. Gompper, “Giant Hexagonal Superstructures in Diblock Copolymer Membranes,” *Physical Review Letters* , vol. 89, no. 23, pp. 2383021 - 2383024, December 2002.

[13] E. Harris, “Toon Wireframe”, Retrieved September 17, 2003, from http://-www.edharriss.com/tutorials/tutorial_toon_wireframe/toon_wireframe.pdf.

[14] G. W. Hart, “Geometric Sculpture,” Retrieved January 20, 2004, from http://-www.georgehart.com/sculpture/sculpture.html.

[15] D. Hearn and M. P. Baker, *Computer Graphics*, Prentice Hall, Englewood Cliffs, NJ, 2002.

[16] J. Howe, “A Digital Archive of Architecture,” Retrieved Febuary 10, 2004, from http://www.bc.edu/bc_org/avp/cas/fnart/arch.

[17] M. Jared, “Single Sided Wireframe Rendering in Maya (all geo types!),” Retrieved September 12, 2003, from http://www.geocities.com/jozvex/tutorials/-wireframe.html.

[18] R. M. Larmen, “Introduction to Fine Art,” Retrieved January 28, 2004, from http://-faculty.evansville.edu/rl29/art105/sp04/index.html.

[19] C. Lynx , “Chayne Lynx Collection,” Retrieved on September 17, 2003, from http://-www.nerd3d.com/index.html.

[20] A. Masood, “Fletcher Perspectives/Detail of Mosque Window,” Retrieved January 25, 2004, from http://fletcher.tufts.edu/perspectives/2003/masood/pages/DetailOf-MosqueWindow.shtml.

[21] J. D. Meltzer, “Gaudi Central,” Retrieved Febuary 22, 2004, from http://www.op.-net/j̃meltzer/gaudi.html.

[22] T. Photos, “Persian Calm, Natanz Mosque, Iran,” Retrieved Febuary 19, 2004, from http://community.webshots.com/photo/2572766/5003646qAANVMQEdp.

[23] A. Rappoport, “Geometric Modeling: A New Fundamental Framework and Its Practical Implications,” *Proceedings, Third ACM/Siggraph Symposium on Solid Modeling and Applications (Solid Modeling ’95)*, May 1995, Salt Lake City, Utah, ACM Press, pp. 31-42.

[24] P. Shin, “Transparency Mapping,” Retrieved January 28, 2004, from http://-dishinpeter.hihome.com/tutorial/tut3d/tutorial8/ark%20of%20rose.htm

[25] M. Spink, “Designs by Michael and Henrietta Spink,” Retrieved January 22, 2004, from http://www.michaelspink.com/ufdesigns.htm.

[26] The Gnomon Workshop, Inc., “The Gnomon Workshop,” Retrieved September 18, 2003, from http://www.thegnomonworkshop.com/tutorials/transparency_shadows/-trans_shadow.

[27] Voidix, “Voidix Wireframe Tutorial,” Voidix, Retrieved Febuary 12, 2004, from http://www.voidix.com/wireframe.html.

[28] W. Wong, “Travel Pictures,” Retrieved Febuary 19, 2004, from http://members.-tripod.com/wendywon/med1998/istanbul.html.

[29] O. Zeller, “Booleans: Problems and Solutions,” Retrieved September 15, 2003, from http://www.3dgraphicsoutpost.com/booleansarticle01.htm.

VITA

Esan Mandal

Visualization Laboratory
A216 Langford Center
3137 TAMU
Texas A&M University
College Station, TX 77843-3137

esan@viz.tamu.edu

Education

M.S. in visualization sciences	Texas A&M University, May 2004
B.Arch	Birla Institute of Technology, India, May 1999

Research Interests

Geometric Modeling
Physically Based Simulation

Employment

Research Assistant	Texas Center for Applied Technology, March 2002 - May 2004
Technical Director Intern	PIXAR Animation Studios, June 2003 - August 2003
Graduate Assistant	Texas A&M University, September 2000 - March 2002
Architect	CSDirekt, India November 1999 - July 2000