A Hybrid Hole-filling Algorithm

by

Junhui Long

A thesis submitted to the

School of Computing

in conformity with the requirements for

the degree of Master of Science

Queen's University

Kingston, Ontario, Canada

September 2013

Abstract

A polygon mesh, or a 3D mesh, consisting of a collection of vertices, edges, and polygons in three-dimensional space, is the standard way of representing 3D objects. In practice, polygon meshes acquired from the 3D scanning process fail to meet the quality requirements for most practical applications. Mesh defects like holes, duplicate elements, non-manifold elements are introduced during the scanning process, which lowers the quality of the output meshes. In this thesis, we describe a complete mesh-repairing process that fixes all defects within a polygon mesh. This process is divided into two parts: the mesh-cleaning part and the hole-filling part. In the mesh-cleaning part, we describe the ways of repairing different types of mesh defects. In the hole-filling part, we discuss two main hole-filling approaches: the surface-based method and the volumetric. In addition, we present a hybrid algorithm by combining the surface-based approach and the volumetric approach. We compare the meshes created by different hole-filing algorithms and show that the new algorithm is a good alternative to the existing ones.

Acknowledgments

First thing first, I would like to thank my supervisor, Dr. David Rappaport, who has consistently provided support and guidance throughout my education at Queen's. I would also like to thank my lab mates, Junjie Zhu and John Howat who have helped me with my research.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the development of three-dimensional printing techniques, polygon meshes, or 3D meshes, have become prevalent in numerous application domains including manufacturing, architecture, automotive, gaming design, medical research etc. Polygon meshes are the digital representation of three-dimensional objects. Because they are stored as computer-readable files, polygon meshes provide an effective means for software designers to view, design and modify 3D objects.

3D scanning is a way of digitally acquiring shapes for three-dimensional objects. Currently, a number of 3D scanning technologies are being used to get digital counterparts of real world objects. While being effective for simple objects, they sometimes fail to produce satisfactory digital outputs for large and complex objects. We call the models acquired directly from the three-dimensional scanning process raw polygon models.

In practice, a polygon mesh needs to meet some certain quality requirements and geometric criteria to be used by practical applications, for example, 3D printing. Raw polygon models created directly by 3D scanning process may fail to meet these requirements. In this thesis, the criteria that a model must satisfy to be geometrically correct will be discussed in detail. We call the errors that violate those criteria ‘geometric defects’. We will also describe a complete mesh-repairing process for removing these defects.

1.1 Applications of 3D Meshes

1.1.1 Modelling and Manufacturing

In recent years, 3D printing techniques have made, and continue to make a direct impact on the manufacturing world. For manufacturing companies, the 3D printing technology provides a more accurate and more efficient alternative to traditional manufacturing. This new technology is especially useful in fields like medical materials production, construction and the aerospace industry where strict accuracy is highly required. Also, 3D printing, also called rapid manufacturing, is faster and more reliable than traditional manufacturing methods. Moreover, digital models are easy to access and store. In fact, traditional prototyping requires considerable time and space for storing and applying changes. However, for 3D digital models, people can easily copy, modify, or delete models as they want; and this provides great flexibility and efficiency for manufacturers to track, produce and manage finished goods.

1.1.2 Game and Movie Development

In the area of computer games, 3D meshes are used to create game characters, visual effects and background scenes. Easy accessibility to 3D meshes enables game developers to view, add, subtract, and stretch objects from different angles. Developers can easily modify and animate characters according to their requirements. For movie shooting, 3D meshes are widely used in various graphic softwares to create three-dimensional scenes and perform three-dimensional animation. Compared with 2D planar animation, 3D meshes are easy to move, flip and rotate. 2D animation techniques tend to focus on image manipulation while 3D techniques usually build virtual worlds in which characters and scenes move and interact.

1.1.3 Industrial Design

3D meshes also play an important role in the industrialized world to help people design. Traditional designing methods describe objects in two dimensions, but objects with 3D properties are hard to represent. Three-dimensional Computer Aided Design(CAD) helps people save time on designing and modifying objects using 3D techniques. With the development of CAD software, people can manipulate an object model no more than clicking a mouse.

1.1.4 3D Facial Recognition

3D facial recognition is a new technology where 3D scanning techniques are applied to distinguish different faces. By capturing geometric features of human faces, computers are able to perform identity checks. Traditional recognition methods like fingerprint and eye iris systems analyze 2D geometric features such as distance, size and relative positions. 3D facial recognition techniques, on the other hand, utilize 3D sensors to build a temporary 3D facial mesh and compare it with the ones in a database. 3D recognition systems have proved to have better accuracy and capability than traditional systems in detecting tiny facial differences.

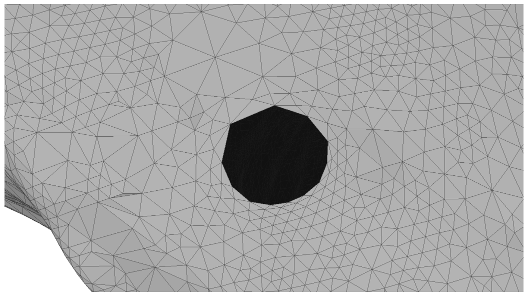
1.2 Problem Description

Range scanning is a type of 3D scanning techniques. It is an information collection process; by using emission devices like light, X-ray or ultrasound, a range scanner is able to collect geometric information and construct a raw mesh for a target object. In practice, range scanning methods work best when a target object has a flat profile shape or a simple convex curved surface. It is capable of producing high resolution, visually accurate models. However it has a few limitations.

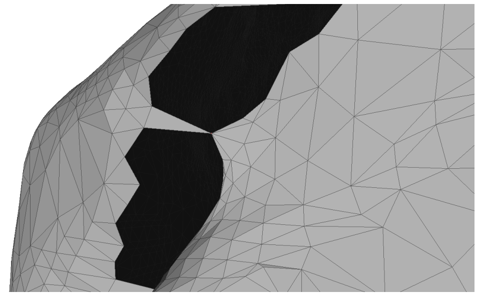
Since the technique uses a linear sensor, the sensor can only detect the surface regions where light rays can reach. For concave regions where rays cannot reach (that means the sensor's sight is obstructed by the object itself), the scanner cannot get the geometry information. As a result, some parts of an object are left unscanned, leading to geometrical errors including holes, non-manifold, isolated elements etc. Small cracks, deep concavities and multiple layers can cause the same problem. To solve the problem, a repairing process must be applied to fix a raw mesh and eliminate these geometric errors.

1.3 Mesh Defects

Polygon meshes are representations of three-dimensional models. A polygon mesh consists of a collection of geometric elements including vertices, edges, and triangles. Mesh defects are the erroneous geometry in a polygon mesh, and can be divided into two categories: geometric defects and topological defects. Geometric defects are the errors within a polygon mesh that against the requirements for being geometrically correct. For example one of the requirements says that a polygon mesh must represent a closed surface. If a polygon mesh represents a surface that is not closed, this mesh is



(a) hole



(b) non-manifold vertex

(c) self-intersecting face

Figure 1.1: mesh defects

not geometrically correct/valid. Topological defects such as undesired handles are the unnecessary geometry that wrongly describe the shapes of an object. For instance, a torus has only one handle; and if a torus mesh has two or more handles, then it is not topologically correct because it does not represent the shape it is supposed to be representing, although the mesh may be geometrically correct.

Geometric correctness/validity is especially important in practical applications. A geometrically correct/valid mesh must have none of the above-described geometric defects. In fact, a polygon mesh should be closed, manifold and free of self-intersections to be used in most applications. By definition, each point of an n-dimensional manifold has a neighbourhood that is homeomorphic to the Euclidean space of dimension n. Each point of a manifold polygon mesh should have a neighbourhood that is homeomorphic to a two-dimensional Euclidean surface. J. Tao[10] summarizes that “a non-manifold geometric elements include non-manifold edges(Figure 3.8) that are not contained in exactly two polygons and non-manifold vertices that are not contained in a disk-like neighbourhood(Figure 1.1 b) ”.

Topological correctness further ensures that a polygon mesh does not misrepresent the shape of an object. Topological correctness is a criterion from the shape point of view. Geometric correctness, on the other hand, focuses on geometric details and ensures that every single vertex and every single triangle follow the geometric requirements.

Mesh-cleaning and Hole-filling

Mesh-repairing is a general process of removing geometric defects. It has two sub processes: mesh-cleaning and hole-filling. Most geometric defects are removed during the mesh-cleaning process. Holes are the only defects left after the mesh-cleaning process and will be fixed during the hole-filling process. In general, fixing holes is much more difficult than fixing other geometric errors because the shape of holes could be very irregular and unpredictable. In chapter 3, we describe the approaches of removing various types of geometric errors. Starting from chapter 4, we explain some hole-filling methods and apply them on 3D objects.

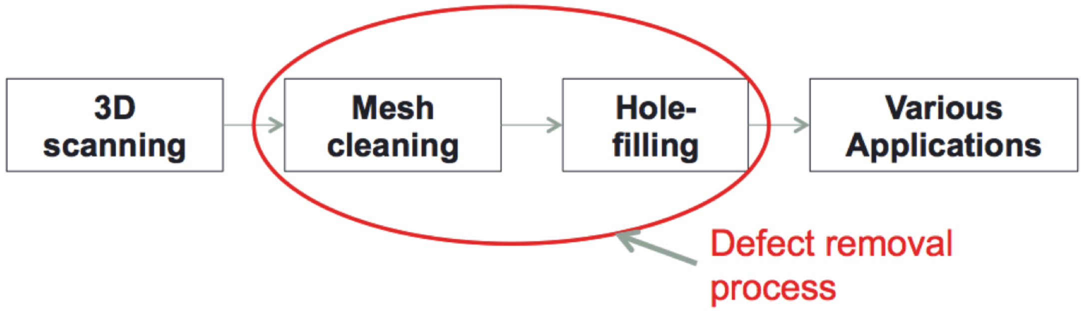


Figure 1.2: mesh-repairing process

1.4 Organization of Thesis

This thesis is divided into six chapters. Following this introductory part is the background, which presents research history and background information on hole-filling algorithms. Chapter 3 describes the definition and representation of polygon meshes and explains a complete process of performing mesh-cleaning. Chapter 4 describes some existing hole-filling approaches. We show and compare the implementation results of these approaches. In chapter 5 we introduce a hybrid hole-filling algorithm. We compare it with a surface-based approach and a volumetric approach, and show how this algorithm works for different digital models. Chapter 6 is a summary of this thesis, including the conclusions we have drawn from our research, and future work that can be done to refine our experimental results.

Chapter 2

Background

A hole is the missing geometry on the polygonal surface; it is composed of a closed cycle of boundary edges within a three-dimensional mesh. In recent years, many approaches were proposed to solve the hole-filling problem. Most of them can be categorized into two types: surface-based and volumetric. A surface-based method fills a hole by triangulating the hole boundary; different surface-based approaches differ in their ways of performing triangulations. A volumetric method, on the other hand, is a process of building a new polygon mesh to replace the original mesh. A good volumetric algorithm effectively simulates the surface shapes and restores the topological features of the original mesh.

2.1 Surface-based Algorithm

A surface-based method operates directly on hole boundaries. It detects all holes explicitly and fixes each of them on a 'case by case' basis. For each hole, a surface-based method creates a patch which is a collection of vertices and triangles used to cover a hole. A patch can be created by performing a set of mesh operations including adding new vertices/triangles, removing vertices/triangles, changing triangle connectivities and repositioning vertices. Because a surface-based method only works on hole boundaries, generally the complexity of a surface-based algorithm depends on the size of holes, rather than the size of the whole mesh.

In fact, a hole, defined as a closed cycle of boundary edges, is a polygon in three-dimensional space. A surface-based method is a process of finding a triangulation for the polygon. However, unlike traditional triangulating methods which use only polygon vertices, here new vertices can be created and added to a triangulation. Because the ways of triangulating a 3D polygon are numerous, the approaches of surface-based filling are various.

In 1995, Barequet and Sharir[2] suggest a 3D hole-closing method. The algorithm builds a patch for a hole by finding the minimum area triangulation. Liepa[11], in 2002, introduces another algorithm by minimizing the combination of dihedral angle and the sum of triangles areas. The new algorithm avoids generating non-manifold edges.

Liepa's algorithm is a well-regarded surface-based method. It improves the hole-closing method by eliminating possible non-manifold edges. Furthermore, minimum area triangulation method tends to render big and flat triangles. With the use of fine fairing and smoothing techniques, Liepa's hole-filling algorithm solves this problem and creates triangles with similar density to peripheral triangles.

Zhao, Gao and Lin [16] propose another surface-based algorithm using the advancing front mesh technique. In this method, the position of a new vertex is determined by calculating the angle between two adjacent boundary edges. If the angle is less than 75°, then no vertices are added. If the angle is between 75° and 135°, only one vertex is added; and if the angle is bigger than 135°, two vertices are added. Starting from the boundary vertices, new vertices and new triangles are added 'layer by layer' until the outermost new vertex has distance to a boundary vertex smaller than a pre-defined threshold. In this case, the outermost vertex and the boundary vertex are merged. Because the positions of newly added vertices are determined by the hole boundaries (boundary vertices and boundary edges), the density of the new triangles are similar to the density of the hole boundary.

2.2 Volumetric Algorithm

Volumetric methods are quite different from surface-based methods. Surface-based methods operate on hole boundaries and fill holes one by one. Volumetric methods, however, operate in the space containing the model and fix all holes at once. For volumetric methods, a mesh surface is first put into volumetric grids. Each grid point is labelled as either 'inside' or 'outside'. In fact, a closed mesh surface partitions the space into two parts. For each grid point, it is either inside the mesh surface or outside the mesh surface. If all the position information of grid points are known, a new mesh that separates all 'inside points' from 'outside points' can be built. Notice the smaller the grid is, the better accuracy the new mesh will have, and the more similar shapes the new mesh will have compared with the original mesh.

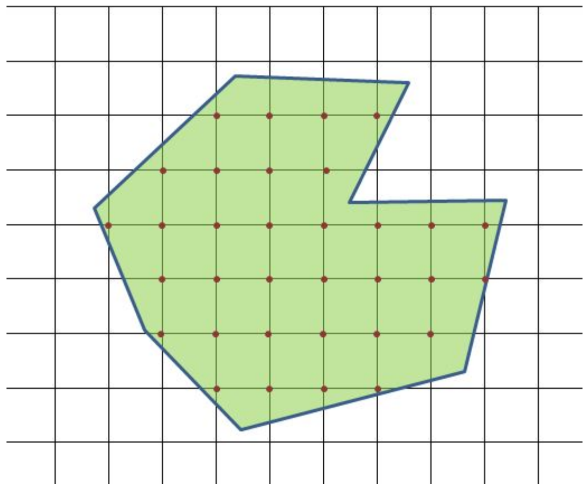


Figure 2.1: inside grid points are in the green region

So the important step of a volumetric method is to determine if a grid point (in 3D space a cube) is inside or outside. Nooruddin and Turk [13] use the 'ray stabbing' method to explicitly list all outside grid points, and mark the rest of the grid points as inside. Podolak and Rusinkiewicz [15] propose a normal-dependent method. That is for a cube, we find a nearest triangle from all mesh triangles. We check the normal of that triangle to determine if the cube is inside or outside. Marschner, Davis, Garr and Levoy [4] and others use a 'flood diffusion' process to extract surface. In Ju's [9] algorithm, cubes that have intersections with the mesh are extracted; and based on the extracted cube, we reconstruct the mesh surface using iso-surface techniques.

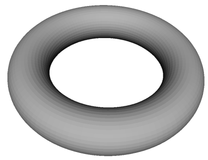
2.3 A Comparison of Surface-based and Volumetric Methods

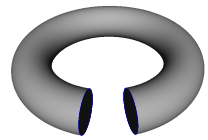
As discussed above, surface-based methods are fast, straightforward and, more importantly, they preserve structure. Geometrical information of the original mesh is preserved after holes are filled. Volumetric methods, on the other hand, use a new mesh to replace the original one. Although a new mesh may have similar shapes, the geometric details of the original mesh are lost.

The main advantage of the volumetric methods is the quality and robustness of output meshes. Meshes created by volumetric methods have good quality in terms of mesh density, triangle interior angle, number of slender triangles etc. Also, volumetric methods are robust because the geometric correctness of an output mesh is always guaranteed.

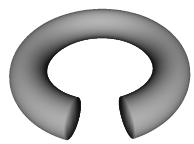
The main drawback of surface-based methods is the lack of robustness. Surface-based methods do not guarantee that outputs are always geometrically correct. Generally a surface-based method extracts a hole boundary from a mesh and fills it by triangulations. Because only the boundary of a hole is considered, the triangulation operations are blind to the rest of the mesh. That means a surface-based method will not know what the geometry is like for the rest of the mesh. A naive triangulation of the boundary may intersect other geometric elements in the mesh. Although in practice the chances of having such self-intersections are slim, users need to manually check the correctness of a output mesh after the hole-filling process. For volumetric methods, output meshes are always geometrically valid. We say that volumetric methods are fully automatic because no user interference is needed throughout the process.

Furthermore, volumetric methods are better in distinguishing between holes and handles. As shown in Figure 2.2(a), torus is an example of a handle. Figure 2.2(b) shows an inconsistent handle, which consists of two hole boundaries. Surface-based methods see it as two unrelated holes and will fill them separately, which generates undesired geometry. A volumetric method, on the other hand, knows the positions of every grid point, and will rebuild a new mesh with correct geometric inside/outside information.





(a) a torus(handle) (b) a broken handle



(c) after the surface-based hole-filling

Figure 2.2: fixing a handle

However, because volumetric methods use volumetric representation(grid points) to represent a mesh, the time and space complexity for building and storing a grid and knowing the position of every grid point is much more costly than a triangulation method. Also, when considering a complex polygon mesh with only one mesh triangle missing, the volumetric method will still need to build a grid system and reconstruct a new mesh. However, a simple surface-based triangulation can locate the missing triangle/hole quickly and find a good triangulation in an efficient way.

As described above, we have an initial idea of surface-based methods and volumetric methods. Surface-based methods focus on geometric details, while volumetric algorithms can see the 'big picture' of a polygon mesh. However, because volumetric algorithms build a mesh based on the positions of all grid points, it is generally more costly to perform.

Depending on different applications, people can choose between surface-based algorithms and volumetric algorithms. In some applications, the topology/shape of a model is more important than its geometric details, then volumetric methods are good options because the output mesh preserves topological features(shapes) of the original mesh. In other application fields like medical related manufacturing where geometric details are crucial and should not be discarded, surface-based algorithms are more preferable.

Chapter 3

Mesh-cleaning

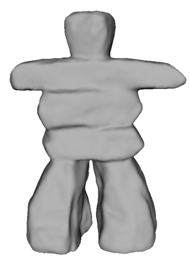
As stated in chapter 1, most geometric defects are removed in the mesh-cleaning process. In this chapter, we describe various types of geometrical defects and the ways of removing them in a polygon mesh. We start with the concept of polygon mesh and discuss some useful polygon mesh representations.

3.1 Preliminaries

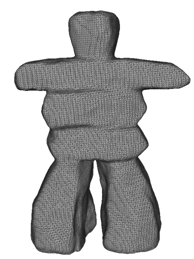
3.1.1 Polygon Mesh and Triangle Mesh

Polygon meshes are used to describe the shapes of three-dimensional models; A polygon mesh is a collection of vertices that are connected by line segments. By definition, a polygon mesh is made up of three types of geometric elements: vertex, edge, and face. Edges are the line segments that connect vertices; faces are the two-dimensional polygons made up of edges. In a polygon mesh, mesh edges are connected by sharing common vertices and mesh faces are connected by sharing common edges.

A triangle mesh is a special type of a polygon mesh where all faces are triangles. Triangle meshes are widely used in various applications thanks to the simplicity and flexibility of representing 3D models. In fact, by triangulating all polygonal faces, a polygon mesh can be easily converted to a triangle mesh. In this thesis, except where indicated, assume that all polygon meshes are composed of triangles.



(a) the smooth view of inukshuk



(b) the triangle view of inukshuk

Figure 3.1: an example of triangle mesh

3.1.2 Mesh Representations

In computational geometry, a triangle mesh can be represented in a number of ways. Different mesh representations use different methods to store geometric information. For instance, a triangle in a mesh can be represented by either three vertices or three edges. A vertex based method is a simple representation that uses vertices to represent edges and triangles. That means an edge is represented by the two vertices it contains, and a triangle is represented by the three vertices it contains. For each geometric element, we create a list. As a result, we will have a vertex list, an edge list and a triangle list. Vertex-based representation is simple, but from storage point of view, it is not space efficient as it takes more space to store a triangle mesh than other representations. In addition, it is not time efficient in terms of mesh operations. For instance, it takes linear time to find the adjacent vertices of a certain vertex.

In general, a good mesh representation is space efficient and time efficient. For example, in the vertex based representation, an edge list is unnecessary if a complete list of triangles are known. That is because every edge of a triangle mesh belongs to a triangle; if all triangles are known, all edges will be known.

Mesh operations are the basic geometry operations on mesh elements, which include vertex addition and removal, vertex repositioning, vertex traversal, face traversal, face split, face addition and deletion, etc. In fact, the mesh-fixing process is a set of mesh operations performed in a sequential way. A time efficient mesh representation lowers the complexity of doing mesh operations and, as a result, lowers the complexity of the whole mesh-fixing process.

Depending on different applications, one may use vertex-based, edge-based or face-based data structures to represent a mesh. The following are some popular mesh representations:

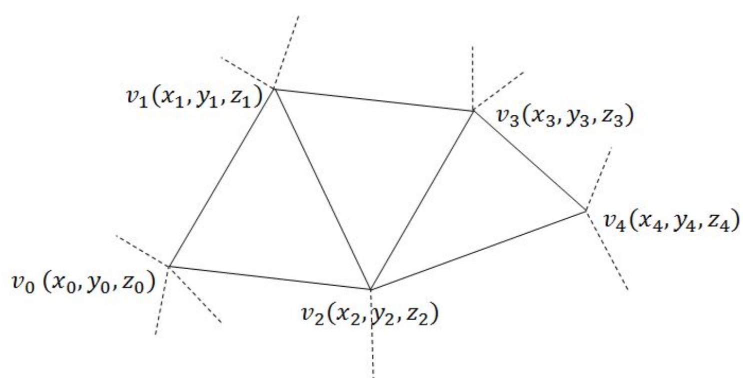


Figure 3.2: a part of of a triangle mesh

- Face mesh: a face mesh only has a face list. Each face contains the coordinates of its three vertices.

Face Mesh	
F_0	$v_0\ (x_0, y_0, z_0)$ $v_1(x_1, y_1, z_1)$ $v_2(x_2, y_2, z_2)$
F_1	$v_1(x_1, y_1, z_1)$ $v_2(x_2, y_2, z_2)$ $v_3(x_3, y_3, z_3)$
F_2	$v_2(x_2, y_2, z_2)$ $v_3(x_3, y_3, z_3)$ $v_4(x_4, y_4, z_4)$

Figure 3.3: face mesh

- Vertex-Vertex mesh: a vertex based mesh only has a vertex list. Each vertex has a list of its adjacent vertices.

Vertex-Vertex Mesh

Vertex List		Adjacent Vertices
v_0	(x_0, y_0, z_0)	v_1, v_2
v_1	(x_1, y_1, z_1)	v_0, v_2, v_3
v_2	(x_2, y_2, z_2)	v_0, v_1, v_3, v_4
v_3	(x_3, y_3, z_3)	v_1, v_2, v_4
v_4	(x_4, y_4, z_4)	v_2, v_3

Figure 3.4: vertex-vertex mesh

- Face-Vertex mesh: face-vertex mesh has a face list and a vertex list. Each vertex in the vertex list is given an index and each face is made up of indices of three vertices.

Face-Vertex Mesh

Vertex List	
v_0	(x_0, y_0, z_0)
v_1	(x_1, y_1, z_1)
v_2	(x_2, y_2, z_2)
v_3	(x_3, y_3, z_3)
v_4	(x_4, y_4, z_4)

Face List	
F_0	v_0, v_1, v_2
F_1	v_1, v_2, v_3
F_2	v_2, v_3, v_4

Figure 3.5: face-vertex mesh

- Winged-edge mesh: winged-edge is an edge-based representation. All the connectivity information is stored in edges. A winged-edge has the information of its adjacent vertices, connecting edges and adjacent faces.

- Half-edge mesh: half-edge mesh splits a full edge into two directed half edges. In this manner each edge contains two half edges and each triangle contains three half edges. Every half edge belongs to only one triangle.

In a half-edge mesh, a triangle contains three half edges. Every half edge is only used by one triangle. The order of vertices in a half edge indicates the direction of the edge. In a geometrically correct triangle mesh, each half edge has a pair edge. A pair edge is an edge composed of the same vertices but has opposite direction. However, if an edge is on the hole boundary, it does not have a pair edge.

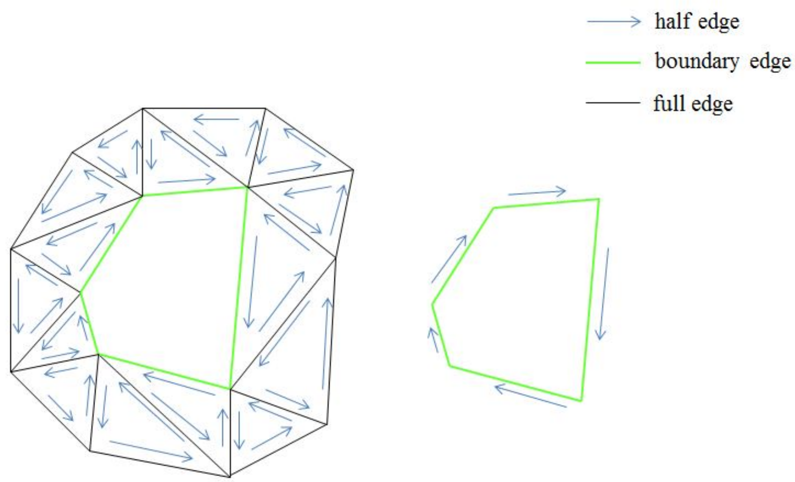


Figure 3.6: a hole and its boundary edges in a half edge mesh

3.2 Geometric Defect: Mesh-cleaning

A geometrically correct polygon mesh must be closed, manifold and free of self-intersections. As explained in previous chapters, a mesh must satisfy a set of rules and criteria to be geometrically correct. Mesh defects are the geometry that violate these rules. We list the types of mesh defects as below:

- *Duplicate Vertex:* two vertices are duplicate if they are in the same position in 3D space.
- *Duplicate Triangle:* two triangles are duplicate if they are composed of the same vertices.
- *Isolated Vertex:* a vertex that is not used by any mesh triangles is an isolated vertex.
- *Isolated Triangle:* a triangle that is not connected with the core part of a mesh

is an isolated triangle.

- *Self-intersection:* self-intersections are two or more faces that intersect with each other.
- *Boundary Edge:* a boundary edge is an edge that is only used by one mesh triangle.
- *Hole:* a hole is a collection of boundary edges; these edges form a closed cycle in 3D space.
- *Non-manifold Element:* If a polygon mesh is non-manifold, that means there exists at least one point on the mesh surface that has a neighbourhood that is not homeomorphic to 2D Euclidean space.

Note that the above described types of mesh defects are not self-exclusive. For example, a boundary edge belongs to a hole; it is also a type of non-manifoldness. Two duplicate vertices can also be isolated vertices.

3.2.1 Duplicate Element

Duplicate elements are vertices or triangles that convey the same geometric information. Two vertices are duplicate if they are in the same position in 3D space. In this case, one of them must be removed. In some papers, two vertices that are too close to each other are also labeled as duplicate. In practice, close vertices tend to create slender triangles, which leads to bad mesh qualities. So in those papers, if the distance between two vertices is smaller than a threshold value, they are marked as duplicate.

An unreferenced vertex is a vertex that is not used by any mesh triangles. If the vertex to be removed is an unreferenced vertex, we can simply remove that vertex from the vertex list. However, the same cannot be done for a referenced vertex. Assume a referenced vertex is removed from a vertex list, then the mesh triangles that used to contain that vertex cannot reference it any more. However, because two duplicate vertices are in the same position, we can replace the removed vertex with its duplicate vertex. That means for each triangle that used to reference the removed vertex, we let it reference the duplicate vertex instead. After that we can safely remove that vertex from the vertex list.

3.2.2 Isolated Element

Isolated elements include isolated vertices and isolated triangles. By definition, an isolated vertex is in fact an unreferenced vertex. As described above, we can remove an unreferenced vertex without concerning the rest of the mesh elements.

An isolated triangle is a triangle that is not connected with the core mesh. A core mesh is the major part of a mesh that defines the profile of a 3D object. Core mesh must be preserved while isolated elements must be removed. If we view isolated triangles in graphical software, visual wise they seem like pieces ‘floating’ in space.

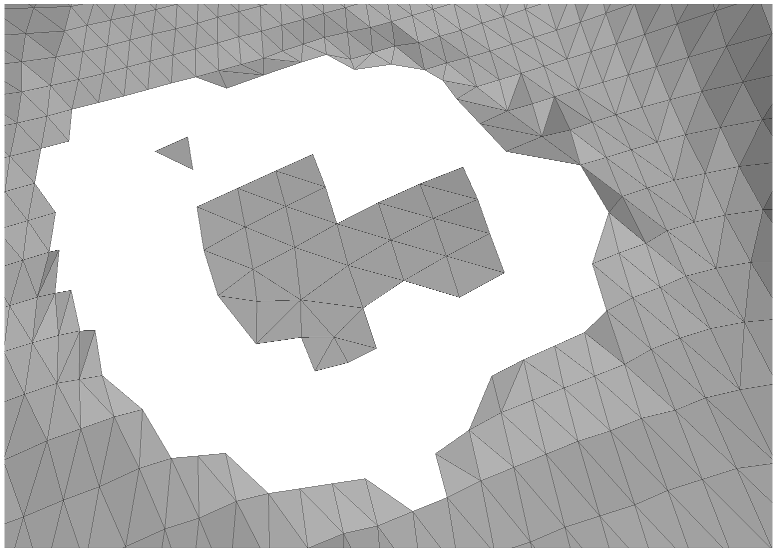


Figure 3.7: isolated element

For isolated triangles, we need to remove the triangles as well as all the vertices they use. Because isolated triangles do not share any mesh elements with the core mesh, the removal of them does not affect the geometric correctness of the core mesh.

3.2.3 Self-intersection

Self-intersection is a type of geometric defect where two or more triangles intersect with each other. Usually a scanner generates self-intersections in regions where object surfaces are complex and twisted.

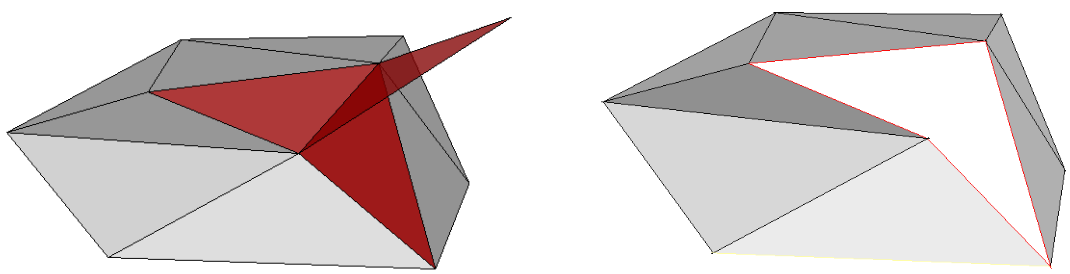
Given a triangle mesh, we must check if it contains self-intersections. A naive algorithm checks all possible pairs of triangles. This can be done in $O(n^2)$ time, where n is the number of triangles within a mesh. In fact, given two triangles, it takes constant time(that is, $O(1)$) to determine whether they intersect or not. Each time we choose two triangles from a total of n. As a result, the time complexity in total is quadratic. A more efficient approach is inspired by the sweep line algorithm.

The sweep line algorithm is used to find all intersections for a set of planar line segments. Instead of line segments, here we have triangles in 3D space. At first all triangles are sorted. To be specific, every triangle has a minimum $x$value and a maximum x value. All $2n$ x-axis endpoints are sorted. After that a one-time x-axis sweeping is applied just like the 2D line sweeping. By using this algorithm, a lower complexity, i.e. $O(nlogn)$, can be achieved.

If two triangles are found self-intersecting, both triangles must be removed. After the removal, the regions where triangles are removed become holes. Usually the hole created after the removal has relatively small size. We will see in the next chapter that a small-sized hole can be filled effectively using surface-based hole-filling algorithms. In this way, all self-intersections of a triangle mesh are removed.

3.2.4 Non-manifold Element

By definition, each point of an n-dimensional manifold has a neighbourhood that is homeomorphic to the Euclidean space of dimension n. As we know, a polygon mesh is a two-dimensional surface embedded in three-dimensional space. So each point of a manifold polygon mesh has a neighbourhood that is homeomorphic to a two-dimensional Euclidean space. J. Tao[10] summarizes that "a non-manifold geometric elements include non-manifold edges that are not contained in exactly two polygons and non-manifold vertices that are not contained in a disk-like neighbourhood".



(a) non-manifold edge and non-manifold face (b) removal of non-manifold edge

Figure 3.8: non-manifold edge and non-manifold face

First, we discuss the ways of removing non-manifold vertices. As discussed above, we cannot delete a referenced vertex directly from a vertex list. This may cause errors as the triangles that used to contain the removed vertex will not be able to reference it any more. For a non-manifold vertex, the solution is to remove the vertex as well as the triangles that contain the vertex. Like what we see in the self-intersection removal process, the removal of mesh triangles leads to the creation of holes. These newly created holes will be fixed in the proceeding hole-filling process. Similarly, for a non-manifold edge, not only the edge needs to be removed, the triangles that contain the edges need to be removed as well. As shown in Figure 3.8 three triangles that use the non-manifold edge are removed, and the geometry after the removal is a hole of four boundary edges.

3.3 Implementation of Mesh-cleaning

We apply the above mesh-cleaning techniques to raw 3D models and see the changes on the numbers of vertices, edges, and triangles before and after the mesh-cleaning.

Model	No. of vertices	No. of faces
Bunny (before cleaning)	35947	69451
Bunny (after cleaning)	34834	69451
Dragon (before cleaning)	437645	871414
Dragon (after cleaning)	436846	870887
Mummy (before cleaning)	717086	1426014
Mummy (after cleaning)	713567	1419480

Table 3.1: the number of vertices and faces before and after mesh-cleaning

For the bunny model, the number of vertices decreases and the number of faces remains the same. In fact, the only geometric defect in the bunny model is duplicate vertex. The mesh-cleaning process fixes all the duplicate vertices, and no faces are removed. In the Dragon model, both the number of vertices and the number of faces decrease, that means not only vertices are removed, some triangles are removed as well. The same applies to the mummy model where thousands of vertices and faces are removed during the mesh-cleaning process.

Chapter 4

Implementation of Hole-filling Algorithms

In 1996, Barequet, Sharir and Eppstein [1] found an interesting research result on 3D polygon triangulation. They have proved that deciding whether a three-dimensional polygon can be triangulated without self-intersection is NP-Complete. In fact the number of candidate triangulations is exponential. However, as is described in [11], "we concern ourselves with the lesser problem of finding weight-minimizing (but possibly self-intersecting) triangulations". A simple triangulation method triangulates the 3D polygon randomly. In this case, we create a random triangle inside the polygon and this triangle splits the polygon into two separate sub-polygons; the same is applied recursively to the sub-polygons until the no new polygons, except triangles, are created. In practice, a random triangulation without applying any other restrictions is less satisfactory. Better results can be achieved by applying some restrictions on the triangles created. For example, if a new random triangle has a large interior angle, then this triangle will be discarded. Other restrictions are also applied to get better triangulations. Here we start with the minimum weight triangulation.

4.1 Minimum Weight Triangulation and Its Implementation

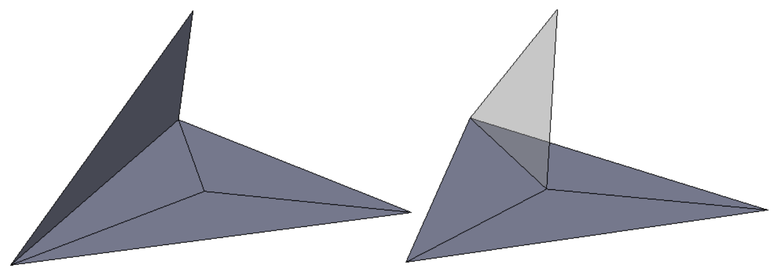


Figure 4.1: two types of triangulation in 3D space

The minimum weight triangulation is the process of finding the triangulation that has the smallest measure (area, edge length, or dihedral angles, etc). There are many ways in which to define the weight that is minimized, for example, sum of triangle areas, sum of edge lengths, max or min edge length or area, etc.

In the paper "Filling gaps in the boundary of a polyhedron"[2], Barequet and Sharir use the minimum area triangulation to fill a hole. By applying a dynamic programming method, the optimal triangulation can be found in $O(n^3)$ time. In the minimum weight triangulation, because new faces are created using the vertices of the original mesh, the number of mesh faces increases but the number of vertices stays the same. The minimum area triangulation is a popular method because it is easy to implement and reasonably fast. In our experiments we find this method works best for models that have relatively small-sized holes. It works less satisfactory for large holes because it is memory-intensive, and with the increase of hole sizes, memory needed to run this algorithm grows considerably big. In addition, the sizes of the newly generated triangles tend to be big and incompatible with the rest of the mesh.

4.2 Liepa's Hole-filling Algorithm and Its Implementation

The minimum area triangulation algorithm is a passable hole-filling algorithm, but it is not always effective especially for large and complex models. To solve this problem, Liepa[11] proposes another triangulation method. Instead of finding a minimum area triangulation, this algorithm looks for a triangulation with 'best' (i.e., smallest) dihedral angle. And among all the triangulations with the same dihedral angle, it chooses the one with the minimum accumulated area. After the refined triangulation is found, a 'mesh-refining' process is applied. To be specific, new mesh vertices are created to break big triangles; a Delaunay triangulator is then used to 'relax' patch edges. That means for the two triangles adjacent to an edge, if the one of the two opposite vertices of the edge lies inside of the circumsphere of the opposing triangle, then this edge must be swapped. The final mesh-smoothing step repositions every newly created vertex. The purpose of the mesh-smoothing process is to improve mesh qualities and create visually comfortable and reasonable meshes. More mesh-smoothing details will be discussed in chapter 5. But like many other surface-based methods, Liepa's hole-filling algorithm does not exclude the possibility of self-intersections. However, in practice it is a very well-regarded algorithm that can be applied to most 3D models.

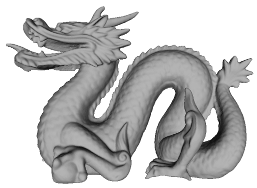
4.3 Robust Repair and Its Implementation

"Robust Repair" is a volumetric method proposed by J. Tao[9]. This approach divides space into a regular grid, and detects the grid edges being cut by a raw mesh. It keeps a list of grid edges that link between the inside grid points and the outside grid points. According to the grid edges, the method 'guesses' a boundary in 3D space that separates the inside grids and the outside grids. After all inside grids are determined, it applies the Marching Cubes [12] method to reconstruct a complete polygon surface. The output mesh is a completely new mesh in terms of geometric details(vertex positions, triangle connectivities, etc). Topological attributes are preserved while geometric details are not.

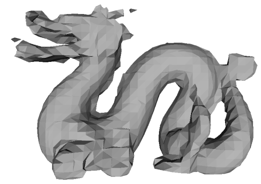
In the robust repair method, space is divided into a regular grid; each grid point is represented by a cubic volume. The size of the cubic volume is determined by the depth of the division performed, which is a pre-defined value set by the user. The deeper the space is divided, the more grid points will be generated, and the more accurate the output meshes will be. Also, if the division depth increases, the number of output mesh triangles increases as well. In this method, a mesh model with more triangles means higher resolution, and better accuracy. As shown in Figure 4.2, the mesh created at depth 4 is not good enough to represent the original mesh. But with the growth of the depth, better accuracy is achieved. On the other hand, however, the division depth should be controlled to get a reasonable result. As shown in Table 4.1, every time the division depth increases by 1, the number of mesh vertices and mesh triangles increase by approximately four times; when the depth reaches to 12, the number of triangles becomes extremely big. In practice, these large mesh models are heavy duties for 3D processing softwares, they add to the complexities of performing all kinds of mesh operations. So a good output mesh should restore the geometric features of the original mesh, and also have reasonable number of geometric elements. In our example, the output dragon mesh with depth 8 or 9 is more preferable because they are accurate and have reasonable sizes.

Depth	Torus		Bunny		Dragon	
	Vertices	Triangles	Vertices	Triangles	Vertices	Triangles
Input Mesh	4794	9576	35947	69451	437645	871414
4	557	1096	822	1632	532	1052
5	2421	4824	3436	6880	2400	4796
6	9413	18808	14090	28180	10112	20192
7	36893	73768	56724	113444	41468	82940
8	149373	298728	227818	455632	167022	334040
9	600837	1201656	912314	1824624	670144	1340288
10	2397053	4794088	3651422	7302840	2683316	5366624
12	38328415	76656808	58467868	116935740	42976066	85952036

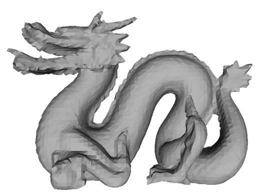
Table 4.1: the number of geometric elements of different division depths



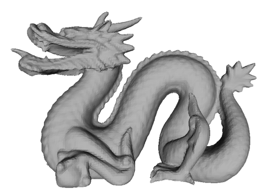
(a) the original mesh



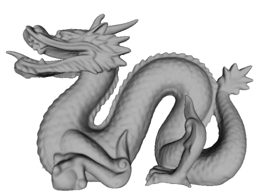
(b) depth = 5



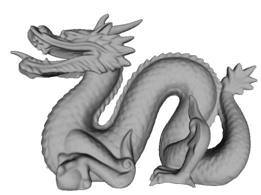
(c) depth = 6



(d) depth = 7

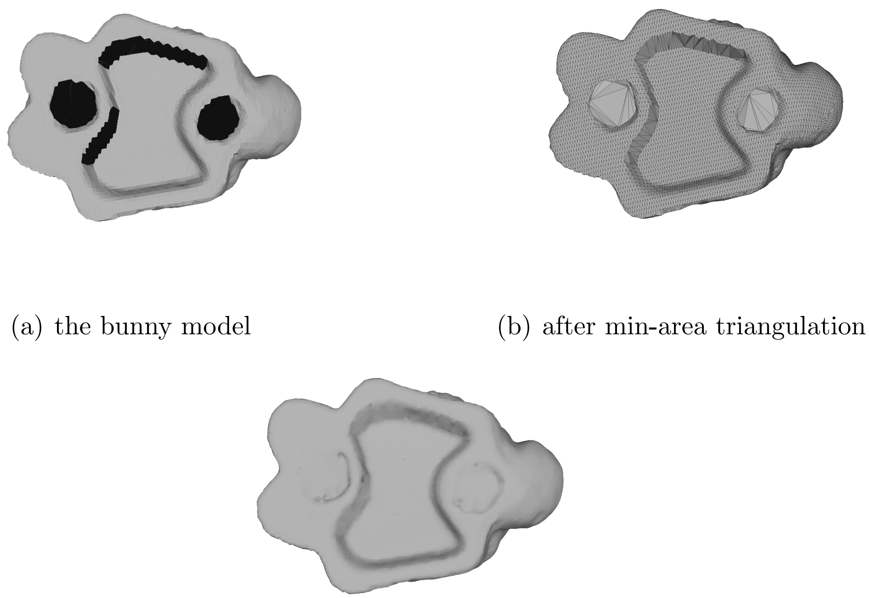


(e) depth = 8



(f) depth = 10

Figure 4.2: the dragon model after robust repair



(a) the bunny model

(b) after min-area triangulation

(c) after robust repair(depth = 7)

Figure 4.3: the bottom of the Stanford bunny model before and after hole-filling

Chapter 5

The Hybrid Hole-filling Algorithm

5.1 A Combination of Volumetric and Surface-based Method

As discussed above, there are two main ways of solving the hole-filling problem. One is surface-based done by triangulation, and the other is volumetric based on mesh reconstruction. Each has advantages over the other. In 2005, Podolak and Rusinkiewicz [15] published a paper that introduced a new volumetric method called the atomic volume algorithm. Like most volumetric methods, this method does operations on 3D volumes, but unlike many other volumetric methods, it is geometry-preserving. Instead of reconstructing a new mesh to replace the original mesh, it builds surface patches specifically for holes. In this method, an adaptive space grid is created, and only the volumes near the hole boundaries are divided. In practice, a raw polygon mesh has much fewer boundary edges than non-boundary edges. Compared with other volumetric methods that divide space into a regular grid, the number of grid points created in this algorithm is much fewer than that in a regular division. Furthermore, because it keeps the geometric information of a polygon mesh, the atomic volume method is seen as a good alternative to other volumetric methods especially in applications like medical and aerospace materials production where geometric consistency is seen as an essential criterion.

In this chapter, we propose a new, hybrid hole-filling approach based on the atomic volume algorithm. We explain our method in a step by step manner. The first four steps are derived from the paper “Atomic Volumes for Mesh Completion” [15]. The original paper goes through these steps without providing implementation details. We add the details and describe some background information for each of these steps. Starting from step five, we modify the atomic volume algorithm. The motivation of the new method is not to replace any of the existing hole-filling algorithms. Instead, we try to combine the advantages of both surface-based and volumetric methods and hope to provide a new perspective for solving the hole-filling problem. At the end of the chapter, we apply the hybrid algorithm to some 3D models and see how the new algorithm performs on these models.

5.2 Step By Step Process

5.2.1 Detect Boundary Edges

As described in “Atomic Volumes for Mesh Completion” [15], holes are explicitly located before being fixed. A hole is made up of a closed chain of boundary edges. If we can find all boundary edges, all holes can be located. In a triangle mesh, a boundary edge is an edge used by only one triangle. So for an edge, by counting the number of triangles using it we can know if the edge is on the boundary. In a half edge mesh, this can be decided by looking up its pair edge. Pair edges use the same vertices but have opposite directions. If a half edge has no pair edge (a pointer points to null), it is a boundary edge. Thus, holes can be detected by traversing all half edges. The computational complexity of doing this is $O(n)$, which is linear, where n is the number of mesh edges. Because the number of vertices is linear with respect to the number of edges, the complexity is also linear with respect to the number of vertices.

Another method can also be used. In a half edge mesh, given an edge, the edges that share a vertex with it can be determined in constant time per edge. This attribute is especially useful because it tells us that once the first boundary edge is found, the next edge on the boundary can be determined in constant time. As shown in Figure 5.1, starting from the initial boundary edge e_1, we check if the edges that share a common vertex v_2 with e_1 are boundary edges. The search stops when the boundary edge is found; the whole process stops when all the boundary edges of the hole are found. So once the initial boundary edge is known, the rest of the boundary edges can be found in constant time per edge. However, the computational complexity of finding an initial boundary edge is $O(n)$, where n is the number of mesh edges. In practice, it has slightly better performance than the first method, but in theory, they are of the same computational complexity.

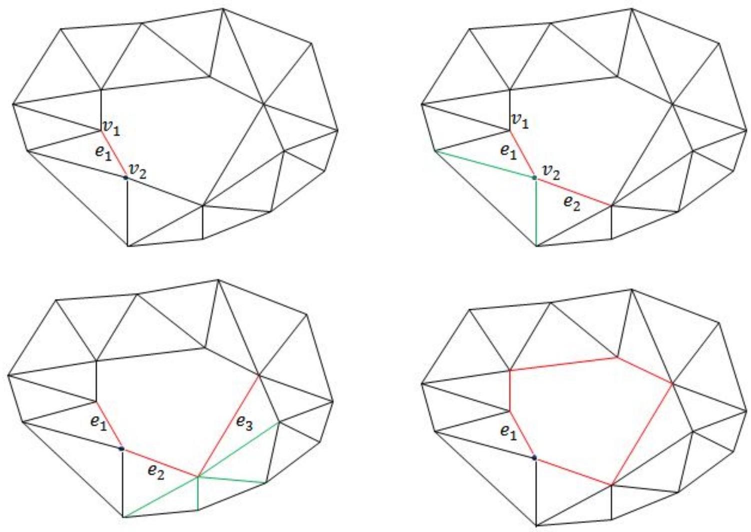


Figure 5.1: detect a hole by edge traversal

5.2.2 Divide Space

A closed mesh divides 3D space into two parts: the inside part and the outside part. Typically, a volumetric method first converts a polygon mesh into a volumetric representation. That means a polygon mesh is converted to a collection of cubic volumes. A cubic volume, or a cube, is the volume element representing a value on a three dimensional grid. The analog of a voxel (volume element) in two-dimensional space is a pixel (picture element), which represents a 2D image element in a bitmap. Figure 5.2 explains a 2D mesh on a regular grid; the grid points inside the polygon are marked as 'inside', and the grid points outside the polygon are marked as 'outside'. At the very beginning, there is only one volume in space, which is called minimum bounding box. Here we describe the concept of minimum bounding box.

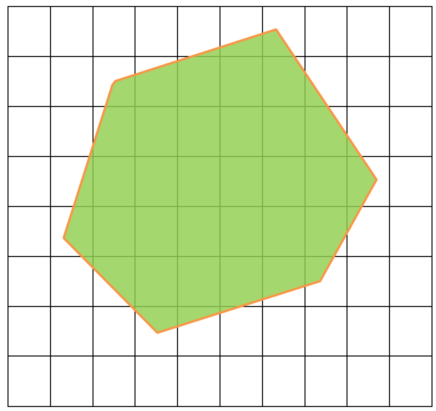


Figure 5.2: inside grid points are contained in the green region

***Minimum bounding box*: for a point set in three-dimensional space, the minimum bounding box is the cube with the smallest measure(area, volume, perimeter, etc) that encloses all vertices.**

In our hybrid hole-filling process, patches are created to fill holes. A patch is a collection of connected mesh triangles; it is within a bounding box. In fact, all mesh operations are done inside a bounding box. Note that a minimum bounding box is different from an 'arbitrarily oriented minimum bounding box'. Finding the smallest bounding box that has sides parallel to the coordinate axes is easy and straightforward. However, the problem becomes much more difficult if the orientation is not fixed. In this thesis, we only need to consider a minimum volume bounding box with fixed orientation.

A minimum bounding box needs to be divided and two ways are available: uniform division and adaptive division. Uniform division splits 3D space into a regular grid where all cubic volumes have the same size. Adaptive division, on the other hand, does not require all volumes being the same size; different volumes may have different sizes depending on how many times it has been divided.

Before looking into the details, we introduce an useful data structure: the octree data structure.

***Octree*: an octree is a tree data structure. The root of an octree represents a bounding box. Every volume, except for the root volume, has a parent volume at its upper level. Every divided cube has eight child cubes. An atomic volume is a leaf node in the octree that is not divided.**

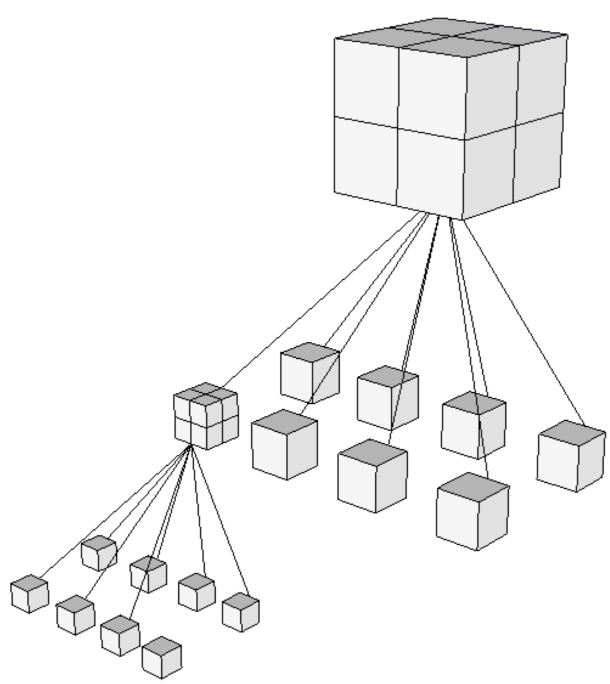


Figure 5.3: octree data structure

An octree is the representation of a space division. The 'division depth' of a space division is defined as the maximum depth of an octree. For a uniform division, with the increase of the division depth the number of grid nodes grows exponentially. For adaptive division, however, the number of grid nodes grows much slower. Every time the division depth increases by one, only some nodes are divided; and most nodes are not divided. As a result, the number of total grid nodes is much fewer than that of an uniform division.

In the paper of "Atomic Volumes for Mesh Completion"[15], Joshua Podolak and Szymon Rusinkiewicz propose an adaptive method for dividing a bounding box. They specify that "*a bounding volume of the mesh is adaptively split into cubes until each cube contains a trivial (for hole-filling purposes) portion of the mesh. Cubes that do not contain the boundary of a hole generally need not be split any further*". Also, inconsistent volumes must be divided in order to define a complete mesh within a volume. In our method, we change the division rules as below:

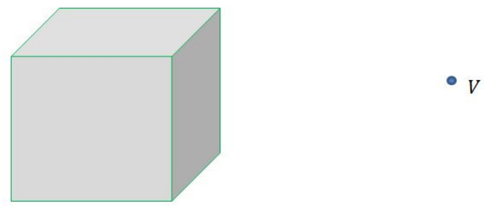
Starting from the root cube, if a cube contains the boundary of a hole, plus it has not reached a pre-defined maximum depth, this cube must be divided.

The division depth is defined by the user, and with the increase of division depth, more and smaller volumes are created. In our method, we don't exclude the possibilities of having inconsistent atomic volumes in space. In fact, for an inconsistent atomic volume, we cannot define a complete surface for it using a volumetric method, and that is the reason why atomic volume method requires that every volume must be consistent, and keeps dividing a volume if it is not. In the hybrid algorithm, the boundary of an inconsistent volume will be constructed in the surface-based hole-filling process.

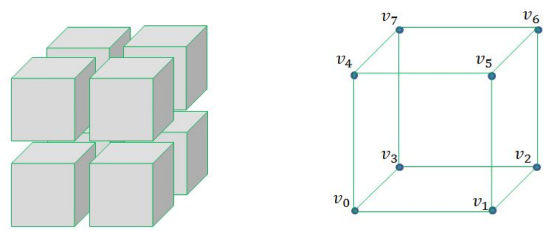
5.2.3 Acquire Adjacency Information

After a bounding box is divided, the space contains a set of volumes of different sizes. For each volume, we create an "adjacency list". An adjacency list stores a list of 'adjacent' volumes of a volume. Here, two volumes are adjacent if they share a common face in 3D space.

The following example is a one-time division of a cubic volume. At first, volume V is divided into eight child volumes, denoted by $v_0, v_1, v_2, ..v_7$. Each child volume shares three faces with other child volumes. To be specific, v_0 shares faces with v_1, v_3 and v_4; v_1 shares faces with v_0, v_2 and v_5, etc. Based on this, an adjacency list is created for each volume as shown in Table 5.1.



(a) a volume in space



(b) dividing a volume

Figure 5.4: a simple division of a volume

Volume	Adjacency List	Volume	Adjacency List
v_0	v_1, v_3, v_4	v_4	v_0, v_5, v_7
v_1	v_0, v_2, v_5	v_5	v_1, v_4, v_6
v_2	v_1, v_3, v_6	v_6	v_2, v_5, v_7
v_3	v_0, v_2, v_7	v_7	v_3, v_4, v_6

Table 5.1: adjacency list of a simple division

At the very beginning, V is the only volume in space. After V is divided, new volumes together with the adjacency lists of these new volumes, are created. In this example, because no volume has common faces with the divided volume V, we need not update the adjacency lists for other volumes.

Now consider another case where volume W is adjacent to volume V, as shown in Figure 5.5. After the division, V is no longer atomic and should be removed from the adjacency list of W. As shown in Table 5.1(b), W still has V in its adjacent list. We need to notify all the volumes adjacent to V, in this case W, to update their adjacency lists. After that, W has a new adjacency list containing the right adjacency information, as shown in Table 5.1(c).

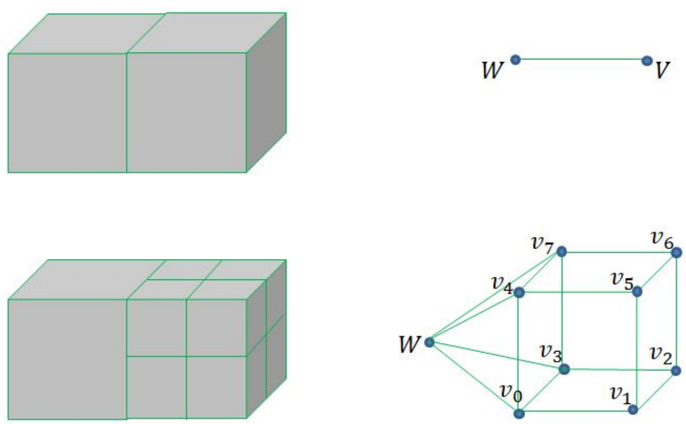


Figure 5.5: two adjacent volumes

(a) adjacency list before V is divided

Vertex	Adjacency List
V	W
W	V

(b) V is divided, children volumes are added

Volume	Adjacency List
v_0	W, v_1, v_3, v_4
v_1	v_0, v_2, v_5
v_2	W, v_1, v_3, v_6
v_3	v_0, v_2, v_7
v_4	W, v_0, v_5, v_7
v_5	v_1, v_4, v_6
v_6	v_2, v_5, v_7
v_7	W, v_3, v_4, v_6
W	V

(c) Updating the adjacency list for W

Vertex	Adjacency List
v_0	W, v_1, v_3, v_4
v_1	v_0, v_2, v_5
v_2	W, v_1, v_3, v_6
v_3	v_0, v_2, v_7
v_4	W, v_0, v_5, v_7
v_5	v_1, v_4, v_6
v_6	v_2, v_5, v_7
v_7	W, v_3, v_4, v_6
W	v_0, v_2, v_4, v_7

Table 5.2: updating adjacency list

5.2.4 Mark Volume Types

As mentioned above, a mesh divides 3D space into two parts. If a volume does not intersect a mesh surface, it must be either completely inside or completely outside the mesh. On the other hand a volume that intersects a mesh must have some parts inside, and some parts outside of the mesh. We categorize all volumes into three types:

- A **blank** volume is a volume that does not intersect the mesh surface
- An **in/out** (i/o) volume is a volume that contains mesh surface but does not contain the boundary of a hole
- A **hole** volume is a volume that contains the boundary of a hole. To be specific, a hole volume is a volume that intersects with one or more boundary edges

For a blank volume in space, it is either inside a mesh or outside a mesh, and whether it is inside or outside is determined using a minimum cut method which is described in the next two steps. For an i/o volume, the inside part is separated from the outside part by a consistent mesh surface. That means a boundary is completely defined within a i/o volume. Similarly, for a hole volume that intersects one or more boundary edges, some parts are inside and some parts are outside. However, only an incomplete surface is defined in a hole volume. The partial surface fails to define a complete boundary that separates inside parts from outside parts. In the hybrid algorithm, a complete boundary is built for a blank volume using volumetric methods; and a complete boundary is built for a hole volume using surface-based methods.

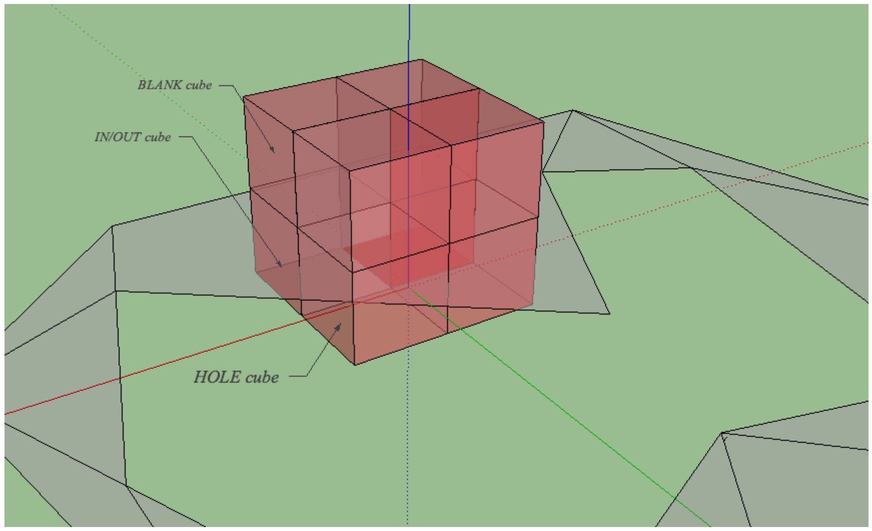


Figure 5.6: three volume types

5.2.5 Build Adjacency Graph

According to the adjacency list and the type of each volume, we construct a weighted adjacency graph. In the graph, every blank volume is represented by a single node and every i/o volume is represented by two nodes: an 'inside' node that represents the inside part and an outside node that represents the outside part of an i/o volume. Now consider a hole volume.

According to the atomic volume algorithm[15], a hole volume is split into several tetrahedrons, and each tetrahedron is represented by a node in the adjacency graph. A hole volume is divided into different numbers of tetrahedrons depending on how a boundary edge intersects a volume. Every tetrahedron is labelled as either inside or outside. If all inside tetrahedrons are known, a complete surface will be defined. In our method, however, we use only one node to represent a hole volume. That is because we build the boundary for a hole volume in the surface-based process which is applied right after the volumetric process. In our method, a node that represents a hole volume is seen as a 'virtual' node because no surface is created for a hole volume during the volumetric process. However, this node cannot be removed because we need this node to be connected with its adjacent volumes to build a connected adjacency graph. The rules for creating a weighted adjacency graph are described as follows.

- If a blank volume is adjacent to a blank volume, an edge is created between the nodes they represent. The weight of that edge is defined as the area of their common face.

- If a blank volume is adjacent to an i/o volume, the node representing the blank volume connects to either the 'inside' node or the 'outside' node of the i/o volume. If the blank volume is set as inside, an edge is created between the blank node and the inside node of the i/o volume. Otherwise, the edge links between the blank node and the outside node of the i/o volume. The weight of the edge is measured by the area of the common face between two volumes.

- If a blank volume is adjacent to a hole volume, an edge is created between the nodes they represent.

- If a hole volume is adjacent to another hole volume, an edge is created between the nodes they represent.

- If a hole volume is adjacent to an i/o volume, two edges are created: one is the edge between the hole node and the outside node of the i/o volume. Another edge connects the hole node and the inside node of the i/o volume.

- If an i/o volume is adjacent to an i/o volume, no edges are created.

Note that if a blank volume is adjacent to an i/o volume, the node representing the blank volume should connect to either the 'inside' node of the i/o volume or the 'outside' node. This is explained in [15] that "we may determine whether any point is inside or outside by checking the normal at the closest point on the surface. This lets user determine if that atomic volume should be connected to to the inside node of the i/o cube or the outside node".

5.2.6 Minimum Cut and Initial Patch

Once an adjacency graph is created, the next step is to separate all the inside nodes from the outside nodes. At first the adjacency graph is connected, every node can reach another node within a finite number of steps. In graph theory, a cut is a defined as the removal of edges that splits a connected graph into two sub-graphs. After a cut is applied, graph nodes are partitioned into two separate groups. Our goal is to find a minimum cut with the minimal measure so that the outside nodes are separated from the inside nodes.

In "Atomic volumes for mesh completion"[15], the author specifies two approaches of defining 'weight'. In the first approach, every edge has the same weight and the minimum cut is the cut with minimal number of cut edges. In the second approach, the weight of an edge is measured by the area of the common face shared by two volumes. So the minimum cut is the cut that has minimal sum of areas. As described in [15], "It is important to note that no matter what edge-weights are given, as long as they are finite, the algorithm will yield a correct, watertight surface". Here we use area as our weight measurement.

In graph theory, many methods are known to be able to solve the minimum cut problem. The *Ford-Fulkerson* method [8] is a method that computes the minimum cut in a directed graph. The *Edmonds-Karp* algorithm [7] is an implementation of the Ford-Fulkerson method in $O(VE^2)$ time, where V is the number of graph nodes and E is the number of graph edges. In the paper of "Algorithm for Solution of a Problem Of Maximum Flow in Networks with Power Estimation"[6], E. A. Dinic uses breadth-first search to find the shortest paths and reduces the time complexity to $O(V^2E)$. In "An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision" [3], Yuri Boykov and Vladimir Kolmogorov introduce a new minimum cut algorithm which has worst-case time complexity of $O(V^2EC)$, where C is the cost of the minimum cut. Theoretically this is worse than the standard Edmonds-Karp algorithm; however, good performance has been shown in practical experiments using this method.

In our adjacency graph, we find that the number of edges in the graph is approximately linear with respect to the number of graph nodes. So the time complexity of finding the minimum cut using the Edmonds-Karp algorithm is the same as that of Dinic's algorithm. The time complexity is $O(V^3)$, where V is the number of graph nodes. In this thesis, we use the Edmonds-Karp algorithm.

The Relation Between V and n

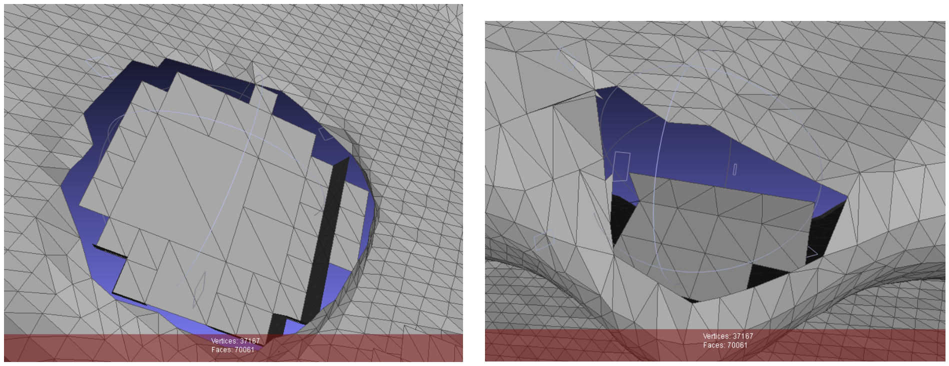
Notice that we have used two types of inputs to represent the time complexity. One is the number of triangles within a mesh, denoted by n; and the other is the number of nodes within an adjacency graph, denoted by V. As we already know, an atomic volume is represented by either one or two nodes in an adjacency graph, so the number of graph nodes is linear with respect to the number of atomic volumes in space. On the other hand, the number of atomic volumes created is determined by the depth of a division as well as the input mesh. So the size of the adjacency graph is determined by both the input mesh and the division depth. However, the number of non-boundary edges has no effect on the number of graph nodes. In the adaptive division, every time a division is performed, only the volumes that contain the boundary of a hole are divided. That means only the number of the boundary edges has impact on the number of graph nodes. When a mesh has larger number of boundary edges, we expect the size of the adjacency graph to be larger as well. Because the number of atomic volumes created by an adaptive division is much less than that in a uniform division, it is interesting to explore the relations between the number of graph nodes and the division depth. The following table describes the growth of graph nodes when division depth increases.

			Depth							
Model	Mesh Triangles	Boundary Edges	3	4	6	7	8	10	12	16
torus 1	9598	4	25	34	56	65	81	185	584	8380
torus 2	9576	14	25	34	81	134	248	953	3561	54911
inukshuk 1	4453	17	34	52	130	215	392	1337	4970	77219
inukshuk 2	4402	44	26	47	218	438	921	3483	13435	210158
bunny 1	70303	39	35	61	112	179	331	1289	4861	74010
bunny 2	69451	223	50	110	391	769	1718	7334	28203	433142

Table 5.3: the number of adjacency graph nodes in different models

As shown in Table 5.3, with the increase of division depth, the number of graph nodes grows. Notice that when the depth increases by 1, the number of graph nodes nearly doubles. However, the number of graph nodes is much fewer than that of a uniform division. For example, when the depth reaches to 16, the number of volumes in a uniform division would be 8^{16}, which is larger than 2.8×10^{14}. In addition, because a volume is represented by at least one graph nodes, the number of adjacency graph nodes is even bigger than the number of atomic volumes created. For many volumetric algorithms, a depth of around 13 or 14 is good enough for practical uses. Assume the division depth is set as 14, and the edge of a bounding box is 1 meter. Then the length of a minimum atomic volume will be less than 0.062 millimeter, which is small enough even for huge mesh models. So in the adaptive division, although the number of graph nodes grows exponentially with respect to the division depth(with base 2), the number is still reasonable especially when considering the division depth does not exceed 14 in most cases.

In an adjacency graph, a cut edge corresponds to a face that separates an inside blank volume from an outside blank volume. This face defines a complete boundary between the two adjacent blank volumes that lie on different sides. All such faces are added to a mesh, and the collection makes an "initial patch". An initial patch is a preliminary cover for a hole that describes the framework of a complete patch. An initial patch is incomplete and unable to cover a hole because it defines surfaces only for blank volumes. It does not define surfaces for hole volumes. In the next step, we use surface-based methods to build a complete boundary for every hole volume.



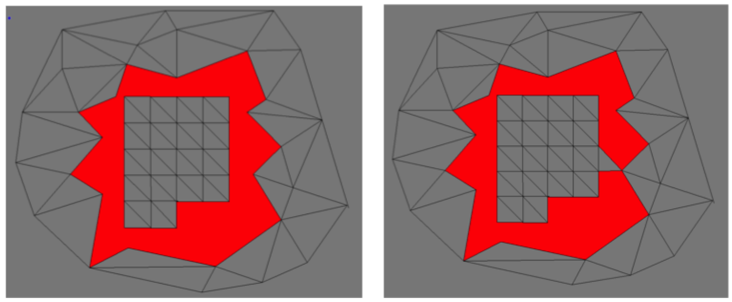
(a) initial patch example 1 (b) initial patch example 2

Figure 5.7: two initial patches in the bunny model

5.2.7 Stitch Patch

An initial patch defines a preliminary boundary and covers a part of a hole. In this step, our goal is to extend the initial patch to make it cover a complete hole.

By definition, a hole is a closed cycle of boundary edges. An initial patch also has a closed cycle of boundary edges (edges that only have one adjacent triangle). To get a complete mesh surface, new triangles must be added to the space between the two boundaries. As shown in Figure 5.8, the red area must be filled. Notice that the geometry between the hole boundary and the patch boundary is not a simple polygon. In fact, the hole boundary is not even connected to the patch boundary. In order to get a hole-like geometry, we 'manually' add one triangle:



(a) (b)

Figure 5.8: an example of an initial patch in 2D

First a vertex is randomly chosen from the patch boundary. Then we choose an edge(line segment) from the hole boundary that is the closest to the chosen vertex. By using the vertex and the edge, a new triangle is created, which leads to the creation of a new hole. The boundary edges of the new hole are made up of the hole boundary, the patch boundary and two edges of the new triangle. In Figure 5.9, the red triangle is added to the mesh.

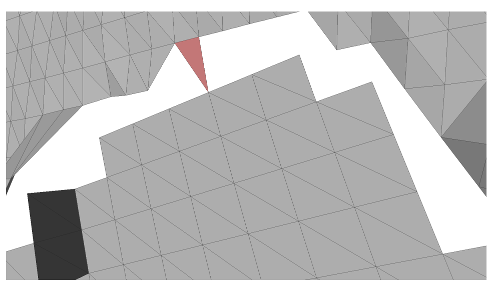


Figure 5.9: manually create a hole

After a hole is created, a surface-based hole-filling algorithm can be used. Here, we use the well regarded Liepa's algorithm.

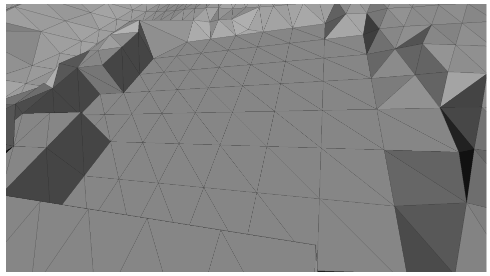


Figure 5.10: gap-stitching using Liepa's algorithm

A special case: As mentioned above, no surface is created for hole volumes in the volumetric process; new triangles are only created for adjacent blank volumes on opposite sides. But what if there are no such adjacent blank volumes in space? Consider a mesh that has very small holes. After the space division, only hole volumes are created around the hole boundary, and no two blank volumes created are on opposite sides. That means no initial patch is created. In this situation, our hybrid hole-filling algorithm becomes a pure surface-based algorithm. The original hole is passed to the surface-based process without adding any triangles. The original mesh is filled using surface-based, in specific, Liepa's algorithm.

5.2.8 Smoothing

After applying Liepa's algorithm, the new mesh surface is now closed and has no more geometric defects. However, the patch consisting of new vertices and new triangles does not render good mesh qualities. Notice that an initial patch is a collection of faces where every face is a part of an atomic volume. Each new face stands either horizontally or vertically in space. In our experiments, we can see apparent differences between an input mesh and a hole patch. The triangles of a patch, for example, may seem too dense or too sparse compared with other mesh triangles. Or the patch may have sharp corners or deep recessions at places where a hole has high curvature. To solve these problems, we apply a mesh smoothing process done by repositioning vertices and faces of a patch.

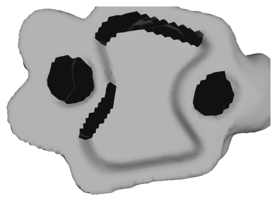
For 3D polygon meshes, mesh smoothing is one of the most important mesh processing operations. Mesh smoothing is used in various applications to remove noise and improve mesh qualities. A variety of mesh smoothing techniques have been published. Among these, Laplacian smoothing is a fast and easy to implement algorithm. In this method, every vertex is moved to to a new position by averaging the positions of its adjacent vertices. It requires a very low computational cost but the disadvantage is that it does not always move a node to the position that provides the best quality.

The goal of a mesh smoothing algorithm is to improve mesh qualities, and, at the same time, produce minimal damage to the geometric features within a mesh polygon. In practice, Laplacian smoothing tends to 'flatten' sharp shapes of an object especially after multiple iterations are applied. A slightly modified Laplacian smoothing method uses weights proportional to the distance between the vertices. This method is also known as the scale-dependent umbrella operator method. In this method, a close vertex has significant impact on determining the position of a new vertex.

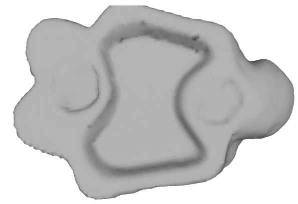
Curvature flow smoothing is a feature-preserving smoothing. In the paper "Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow", face normals are computed and vertices are repositioned into the new normals. This umbrella operator considers the angles opposite the common edge that two triangles share. According to Desbrun et al. [5], curvature flow provides the best smoothing with respect to mesh shapes. Based on this method, Liepa suggests a recursive application of the umbrella operator to get a second-order weighted umbrella operator. To get a second-order umbrella operator, we need to compute the umbrella operators of the direct adjacent vertices of a certain vertex. Liepa's method uses the geometric information (vertex position, edge length, dihedral angle) of the core mesh; the outputs are reasonably smooth and blend well visually with peripheral triangles. In this thesis, we apply Liepa's smoothing method to the patches.

5.3 Results and Comparison

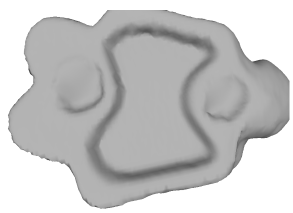
We apply the hybrid algorithm to some mesh models. The output meshes created by different hole-filling algorithms are shown below.



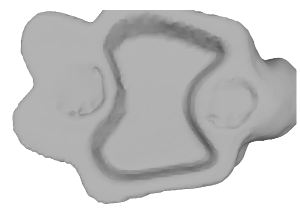
(a) the original bunny model with holes



(b) hybrid algorithm

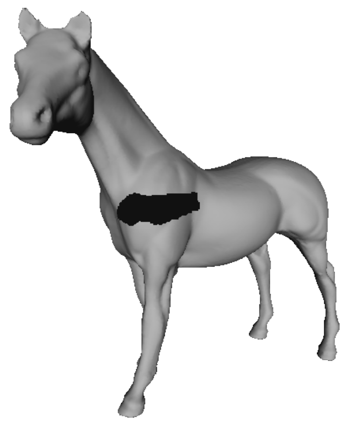


(c) liepa's algorithm

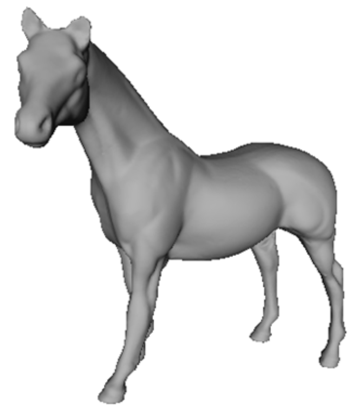


(d) robust repair algorithm

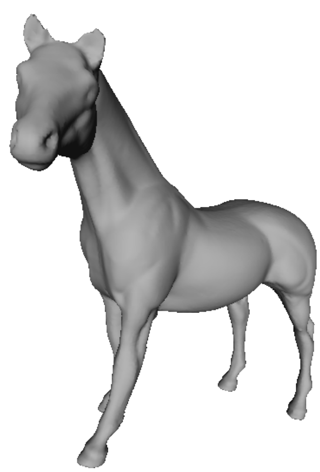
Figure 5.11: the bottom of the Stanford bunny model fixed by different methods



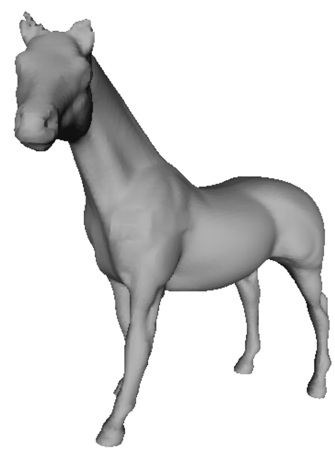
(a) the original horse model with holes



(b) hybrid algorithm



(c) liepa's algorithm



(d) robust repair algorithm

Figure 5.12: the horse model fixed by different methods

5.4 Advantages and Limitation

The hybrid algorithm combines the atomic volume method and Liepa's surface-based method. An initial patch, created by a modified atomic volume method, defines the frame of the hole patch. By using Liepa's surface-based algorithm, we further expand the patch and make it fit to a hole. In general, a surface-based algorithm is reasonably fast because it focuses only on boundary edges instead of the whole mesh. Similarly, our hybrid method only focuses on the regions around holes instead of the whole space. Our method divides a bounding box adaptively where only volumes intersecting a boundary edge are divided. As most volumetric methods divide space uniformly, when the division depth is the same, the hybrid algorithm takes less memory than a volumetric uniform division.

But unlike many volumetric methods, our hybrid algorithm does not guarantee a geometrically correct output. In the hybrid hole-filling method, a surface-based algorithm is applied in the mesh-stitching process, which may lead to the creation of self-intersections. However, because the surface-based method is applied after an initial patch(frame) is created, the risk of producing such self-intersections is greatly reduced. In fact, for most polygon meshes except some deliberately devised ones, Liepa's algorithm seldom yield self-intersecting meshes. Our method only reinforces the robustness derived from Liepa's algorithm.

However, compared with other hole-filling methods, the hybrid algorithm runs much slower. The bottleneck of our algorithm is on computing the minimum cut of an adjacency graph. As described in 5.2.6, the complexity of finding a minimum cut takes approximately $O(V^3)$ time. The larger the division depth is, the more nodes the adjacency graph has, and the more time is required to compute the minimum cut. Also, when compared with Liepa's algorithm that triangulates hole boundary edges directly, the hybrid algorithm adds the patch boundary to the hole boundary, which makes the size of the boundary larger and makes the surface-based process more costly.

Chapter 6

Conclusion and Future Work

In this thesis, we described a complete mesh-repairing process that fixed diverse types of defects within a polygon mesh. The mesh-repairing process contains two sub-processes, the mesh-cleaning process and the hole-filling process. The goal of the mesh-cleaning process is to remove most(except holes) mesh defects and provide a qualified, passable mesh to the proceeding hole-filling process. In chapter 3, we discussed the method of removing each type of mesh defect. After the mesh-cleaning process, the only type of mesh defect left within a mesh is the hole.

Holes are repaired in the hole-filling process. There are two main hole-filling approaches. The surface-based methods create mesh patches by triangulating hole boundaries. Different restrictions can be applied in the triangulating process to improve mesh quality. Because the ways of performing triangulations are diverse, the types of surface-based methods are various. The volumetric methods, on the other hand, generate a new mesh to replace the original mesh. We implemented Liepa's surface-based algorithm and the robust repair algorithm. For both algorithms we discussed the advantages and the disadvantages.

Inspired by the two hole-filling approaches, our work suggested a new, hybrid hole-filling algorithm by combining the atomic volume method and Liepa's surface-based method. In this hybrid algorithm, a modified atomic volume method is applied to get the frame of a hole patch. After that, Liepa's method is used to extend the patch frame to complete the remaining hole area. After a hole is filled, the mesh-smoothing techniques are used to polish patch vertices and improve mesh qualities. We compared the output meshes created by multiple hole-filling methods. We find that the hybrid algorithm has proved to be capable of filling holes with reasonable patches.

However, there is still much research to be done. A limitation of our hybrid algorithm is not being time-efficient for large mesh models. As described in 5.4, the bottleneck of our hybrid algorithm is on computing the minimum cut of an adjacency graph. When dealing with large models, the hybrid algorithm takes an unaffordable amount of time in solving the minimum-cut problem. In fact, repairing polygon meshes with large-size holes is a nontrivial problem for most existing hole-filling algorithms. In our experiments, we find that Liepa's hole-filling algorithm tends to crash memory when dealing with large hole boundaries; and the robust volume method tends to create a deformed replacement of the original mesh. In our hybrid method, the Edmonds-Karp algorithm, which has time complexity of $O(VE^2)$, is not efficient enough. There are many known minimum cut algorithms with better time complexities than the Edmonds-Karp algorithm. The best known algorithm, as described by Orlin James in the paper of "Max flows in O(nm) time, or better"[14], solves the problem in time $O(VE)$, where V represents the number of graph nodes and E is the number of graph edges. Because this is a much more complicated algorithm, future work can be done in analyzing the feasibility of implementing these algorithms.

We also believe that the surface-based process can be improved. In the surface-based process, new triangles are created to cover holes. A good triangulation should be geometrically correct and topologically plausible. We find that a good triangulation is not necessarily a minimum dihedral angle triangulation, nor a minimum area triangulation. To find a minimum triangulation, Liepa's hole-filling algorithm runs a dynamic programming process which has space complexity of $O(n^3)$. In order to lower the space complexity, we can try to find a 'good enough' triangulation instead of a weight-minimizing one. We believe that the random triangulation carries a lot of potential, and could be combined with new restrictions to produce improved results. To be specific, a random triangulation can be created under some certain conditions; if the triangulation created is unsatisfactory or does not meet some quality requirements, we discard it and create a new random triangulation. We keep building new triangulations until an acceptable one is found. It would be interesting to explore what contributes to a 'good enough' mesh model, and how to find one in a more efficient way.

Bibliography

[1] Gill Barequet, Matthew Dickerson, and David Eppstein. On triangulating three-dimensional polygons. In *Proceedings of the twelfth annual symposium on Computational geometry*, SCG '96, pages 38–47, New York, NY, USA, 1996. ACM.

[2] Gill Barequet and Micha Sharir. Filling gaps in the boundary of a polyhedron. *Comput. Aided Geom. Des.*, 12(2):207–229, March 1995.

[3] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(9):1124–1137, September 2004.

[4] James Davis, Stephen R. Marschner, Matt Garr, and Marc Levoy. Filling holes in complex surfaces using volumetric diffusion. In *1st International Symposium on 3D Data Processing Visualization and Transmission (3DPVT 2002), 19-21 June 2002, Padova, Italy*, pages 428–438. IEEE Computer Society, 2002.

[5] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 317–324, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[6] E. A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Math. Dokl*, 11:1277–1280, 1970.

[7] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, April 1972.

[8] L. R. Ford, Jr. and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics = Journal canadien de mathématiques*, 8(3):585–591, 1956.

[9] Tao Ju. Robust repair of polygonal models. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 888–895, New York, NY, USA, 2004. ACM.

[10] Tao Ju. Fixing geometric errors on polygonal models: a survey. *J. Comput. Sci. Technol.*, 24(1):19–29, January 2009.

[11] Peter Liepa. Filling holes in meshes. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, SGP '03, pages 200–205, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[12] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, August 1987.

[13] Fakir S. Nooruddin and Greg Turk. Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):191–205, April 2003.

[14] James B. Orlin. Max flows in $o(nm)$ time, or better. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, STOC '13, pages 765–774, New York, NY, USA, 2013. ACM.

[15] Joshua Podolak and Szymon Rusinkiewicz. Atomic volumes for mesh completion. In *Proceedings of the third Eurographics symposium on Geometry processing*, SGP '05, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association.

[16] Wei Zhao, Shuming Gao, and Hongwei Lin. A robust hole-filling algorithm for triangular mesh. *Vis. Comput.*, 23(12):987–997, November 2007.