AN EXPLORATION OF HOLE FILLING ALGORITHMS

A Thesis

Presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Eric Firestone

June 2008

APPROVAL PAGE

TITLE: An Exploration of Hole Filling Algorithms

AUTHOR: Eric Firestone

DATE SUBMITTED: June 2008

Dr. Zoë Wood	
Advisor or Committee Chair	Signature
Dr. Aaron Keen	
Committee Member	Signature
Dr. Chris Buckalew	
Committee Member	Signature

Abstract

An Exploration of Hole Filling Algorithms

by

Eric Firestone

Laser range scanning is one of the leading methods for the acquisition of 3D models from real world objects. This process, however, introduces significant excess topological handles which increases the complexity of future processing, and lowers the quality of the acquired models. Previous research has shown that the hole filling step of the model creation pipeline is the primary cause of excess handles. We explore the hole filling process in detail and discuss the limits of hole fillers that work on the reconstructed surface and of those that work in the volumetric setting. In addition, we present our algorithm which aims to reduce the excess handles by adapting and improving filters that work in the volumetric domain to fill holes in the scanned data. Using these filters we are able to reduce the topological noise by 47% and to improve the output appearance of surfaces processed by existing hole fillers.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Computerized 3D models are increasingly being used in fields such as motion pictures, video games, and medical research, among others [14, 17]. A common goal in these applications is the creation of as realistic a model as possible, and this is often accomplished by scanning physical objects and digitizing that data. Various methods exist for accomplishing this, but we will focus on the most common one: structured light sensing [3].

In the structured light sensing method of scanning, a focused beam of light, generally a laser, is swept across the model as a point of reference for a conventional video camera, which can relate the illuminated area to a scanline in the 3D representation. Using this data and the camera's known line of sight, correspondence between the viewpoints of the laser and the camera can be triangulated, giving a distance to the object.

Scans are made from multiple angles and the collected correspondences are synthesized into a series of data points called range images, which are subsequently aligned and merged to create a single model. This model is first represented as a cube of voxels (a volume), from which a mesh can later be extracted using an algorithm such as marching cubes [13].

There is no guarantee that these scans will include data about the entire surface of the object being scanned, and more often than not some of the object is occluded from the scanner, creating areas that lack data. For most practical applications these areas must be filled using one or more hole filling techniques. It is our goal to examine the influence of these hole fillers on the final mesh product, particularly on how they affect the topology of the model. We also aim to help minimize the excess topology introduced by these hole fillers through the use of volumetric preprocessors.

1.1 Terminology

There are a number of terms which will be used heavily, and which should be clearly defined for the context of this document.

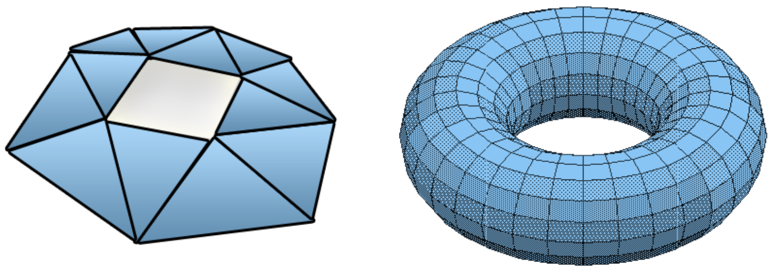

(a) A hole (b) A handle

Figure 1.1: Terminology

Boundary Edge - A boundary edge is the edge of a polygon in a mesh which has only one adjacent face. This term may be shortened to simply "boundary" or "edge" throughout the paper.

Hole - For the purposes of this document, a hole, as illustrated in figure 1.1(a), refers to a break in a surface mesh, as defined by a series of three or more boundary edges. A hole should not be confused with a handle, which is defined later.

Manifold - The term manifold describes a model with a surface that is either devoid of any holes (i.e. has exactly two faces connected to each edge in the mesh), or which has holes that are topologically equivalent to a disc (i.e. each vertex is either surrounded by a disc neighborhood or half-disc neighborhood).

Handle - A handle refers to a loop in the structure defined by a mesh, such as the handle on a coffee cup, or the area in the middle of a doughnut. An example of such a handle is shown in figure 1.1(b). Handles are an expected part of the topology of the mesh, and unlike holes, are not considered a defect in the mesh structure.

Genus - Genus is a mathematical term describing the number of handles, or watertight holes, present on a mesh. As examples, a sphere has a genus of zero, a torus a genus of one, and a double torus a genus of two. Any genus numbers used in this paper are for manifold meshes.

Hole Filling - For our purposes, hole filling refers to the process of eliminating holes defined by boundary edges in the mesh. It does not refer to altering the genus of the mesh through the elimination of handles.

1.2 Problem Description

Structured light range scanning is capable of producing high resolution, visually accurate models, however it does have a few limitations. Because the process uses a linear sensor, areas of concavity on an object can obstruct the scanner's line of sight, leaving unscanned regions. Without treatment, these regions manifest themselves as holes in the final mesh, leaving it non-manifold and visually unappealing (see figure 1.2). As these regions are common on all but the simplest of real world objects, they must be dealt with in a robust and accurate way.

As discussed in the related work section, there are many existing hole filling methods, however each still has significant shortcomings. One of the most reliable and prevalent methods is that employed by VRIP [5], which makes use of the additional scanner data present in the volumetric representation to extract an isosurface. This method works well for creating a manifold mesh, but our previous work [7] has shown that it introduces unnecessary complexity as well.

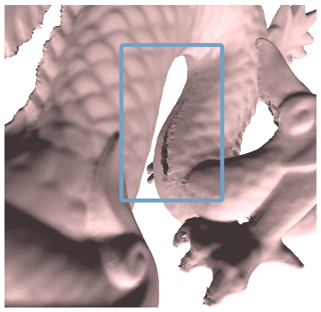
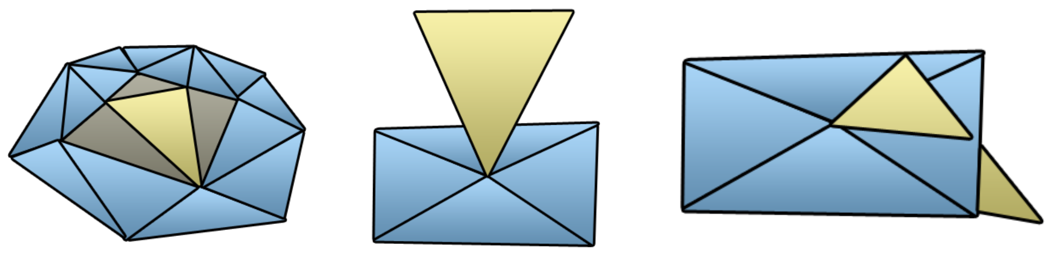


Figure 1.2: A region which is problematic to capture using range scanning.

This complexity comes in the form of excess topological handles, or loops on the surfaces of the model. These handles are generally very small and are not visible to the naked eye when the model is viewed in full. They mean, however, that extraneous data must be stored for the model, and more importantly, that any processing that is done on the scanned model must be done on the extraneous data as well, often severely degrading the results.

VRIP does its hole filling during the reconstruction phase of the reconstruction pipeline, however the other phases of the pipeline are worth reviewing as well. The pipeline begins with the data acquisition stage, during which some device (such as a laser scanner) is used to generate a data set representing the physical object being modeled. In the case of structured light data acquisition, this acquired data is stored into multiple range images, each containing a point cloud representation of the scanned model from a given viewpoint in 3D space. This stage is followed by an alignment stage during which the range images are translated or rotated so as to represent their position on the original model. During the third stage the aligned range images are merged into a single volumetric representation, on which many existing hole fillers focus their repairs. The final stage of the pipeline involves reconstructing a mesh from the volumetric representation. This mesh is the other medium on which hole fillers are commonly applied

Filling holes in a model in a satisfactory way is difficult due to a number of subtle problems, whether the model is in its volumetric representation or in its mesh representation. The volumetric representation provides additional data



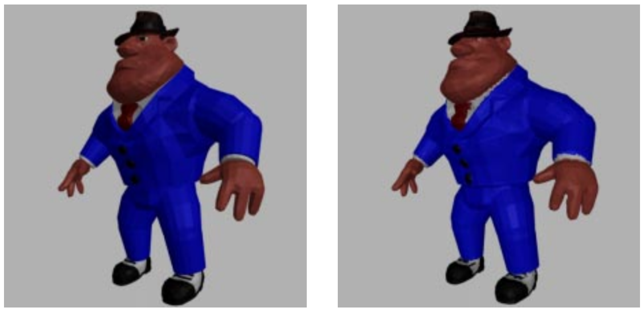
(a) Hole-spanning faces (b) Overlapping vertices (c) Self intersection

Figure 1.3: Common problems in mesh-based hole filling.

from the scanner, however, this representation occurs earlier in the reconstruction pipeline and it is difficult to determine what effects changes to the volume will have on the final mesh. Furthermore, there must be some way to determine from the volume that the extracted mesh will actually be devoid of holes.

Filling holes in an extracted mesh seems more straightforward at first, but this method too is riddled with difficulties, particularly around producing a manifold mesh. The manifoldness of the extraction is not a problem for volumetric hole fillers as the requirement to create a manifold mesh is left on the extraction mechanism, such as marching cubes [13]. When filling holes in the mesh, however, the hole filler must avoid a number of problems, including creating overlapping faces. This can happen if existing hole-spanning faces exist in a hole (see figure 1.3(a)), and faces added to fill the hole overlap them. A similar problem exists with overlapping vertices. In a manifold mesh without boundaries, a vertex must be surrounded by a disc neighborhood, and so situations such as figure 1.3(b) must be avoided. A third, and often more difficult, problem is the case of self-intersection, as shown in figure 1.3(c). The mesh should not intersect itself. Avoiding this requires a programmatic understanding of the mesh's representation in space, which is more difficult to deal with than the relatively simple edge, face, and vertex connections required for the other issues. Fortunately, self-intersection in a mesh, although visually less appealing, does not affect post-processing such as face-count reduction.

As with the problem of hole filling, a number of methods exist for reducing



(a) Before (b) After

Figure 1.4: Mesh smoothing results in an overall loss of detail. Taken from [15].

the number of handles in an existing mesh, although these too are imperfect. A common technique is to smooth the surface of the mesh [8, 15]. This has the benefit of being somewhat naive as to where handles exist, however, it results in an overall loss of detail (figure 1.4), and by no means ensures that all extraneous handles will be removed. More complex methods exist [18] which can explicitly identify the handles to remove and do so in a clean manner without loss of detail. The shortcoming of this technique is that it is complex in its execution, and therefore requires long processing times.

Our goal in this paper then, is to carefully explore the hole-filling algorithms used for surface reconstruction. We examine several different methods and present their results and weaknesses. We also propose a method to reduce the genus of the final mesh in a way which does not compromise existing detail, and which improves the overall appearance of hole filled areas. We focus on the hole filling step of the reconstruction process as that has been shown to introduce the most topological noise.

Chapter 2

Previous Work

2.1 Source Investigation

Our previous work [7] provides a solid exploration into where extraneous handles in a mesh are introduced. This work analyzes two of the four stages of the reconstruction pipeline [4]. Of the four stages: data acquisition, alignment, merge, and reconstruction, we focused on the first and last stages, eschewing the alignment stage in order to limit the scope of the paper, and judging the merge stage as an unlikely source of error.

Our analysis of the data acquisition stage in this previous work shows that any noise introduced by the scanner is likely not a source of extra topology in the final mesh. In general, removing "noisy" data points led to increased hole filling, which resulted in a *higher* genus for complex hole fillers, or significant loss of detail for simple ones.

Our evaluation of the hole filling stage provided further evidence that it was a significant contributor to a mesh's artificially high genus. We tested the influence of this stage by comparing the hole filler from the widely used reconstruction algorithm, VRIP [5], to a simple triangle fan hole filler. This simple hole filler produced meshes with a much lower genus than VRIP. Based on this result, combined with the correlation of higher genera given increased hole filling (as demonstrated by the noise filtering experiment), we determined that the hole filling stage introduces the majority of the extraneous topology. Based on this conclusion, our current work focuses on improving the results of existing hole fillers such that they introduce fewer handles.

2.2 Hole Fillers

As the hole filling stage has been determined to be the primary source of extraneous handles, it is compulsory that we investigate previous research in this area.

2.2.1 Triangulation

In order to evaluate the effects of the hole filling stage on a mesh's genus, for our previous work [7] we implemented our own simplistic hole filler which uses a triangle fan patch over holes. The patch had its center vertex located at the geometric center of all points comprising the boundary edge of the hole, with one side of each patch face aligned along one of the hole's boundary edges.

As discussed in the problem description, a number of problems must be solved for mesh-based hole fillers. Our triangulation hole filler addressed the issue of hole-spanning faces using a mark-and-sweep approach [11], where marking is done only by traversing across edges. Using this approach unmarked faces correspond to hole-spanning faces, which are removed before any hole filling is done. Overlapping vertices are handled after hole filling has completed by simply splitting the vertex into multiple vertices which coexist at the same geometric location. An additional vertex is created for each set of connected faces which touches the overlapping vertex in order to eliminate the overlap condition. As it was not relevant to our investigation, we did not address self-intersection in the mesh.

The greatest strength of this hole filling approach is its simplicity. Given that only one vertex is being added for each hole, it is extremely unlikely that any additional handles will be introduced, leading to a lower genus than is produced by other hole fillers. This algorithm also requires no manual intervention. Holes can be automatically identified based on boundary edges, and no parameters are needed to create the triangle fan patches. Finally, thanks to the methods described above to handle hole-spanning faces and overlapping vertices, the final mesh is always manifold and without boundaries.

Because the triangulation hole filler was created for evaluating other hole fillers by trying to minimize extraneous handles, the visual representation of the filled holes was not a concern. A number of features of the triangle fan patches are non-ideal. Using the centroid as the center of the patch is simplistic, and the geometry of the patch may not fit well with the surrounding mesh. Additionally, self-intersection is not prohibited, so the patch could potentially intersect existing faces of the mesh.

Aesthetically, the triangle fans create noticeable lines as seen in figure 2.1(a) and the large faces we use stand out from the much smaller faces of the rest of the mesh. The algorithm can also alter the desired topology of the model by closing holes it is not supposed to. Holes such as the Buddha's armpit (see figure 2.1(b)) can be mistakenly filled because the armpit, which is supposed to be a tunnel through the model, is viewed by the algorithm as two basic holes (one on either end), which should be patched. This is a difficult problem to avoid using only the data available in the mesh, but one which is not an issue for volumetric hole fillers.

Finally, although not relevant for most models, the triangulation hole filler cannot handle a mesh which is split into multiple distinct pieces. For a model of this sort, the mark-and-sweep phase would discard all but one of the pieces, and even without this phase, the hole identification step would fail if the hole could not be traced in a complete loop from one vertex back to itself (here, too, it would fill the hole only on one piece, ignoring any additional pieces).

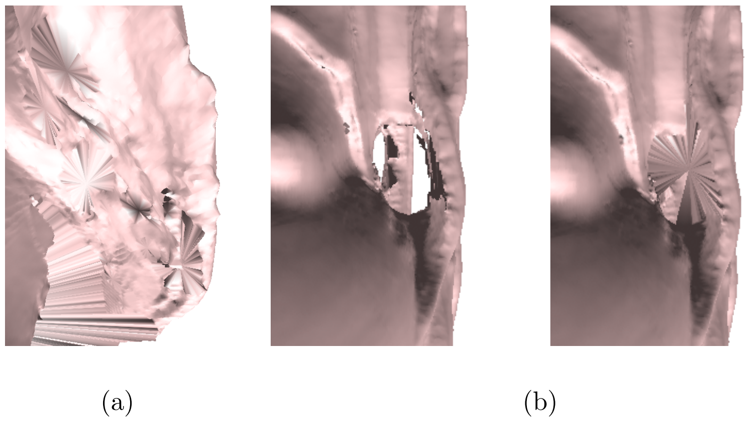


Figure 2.1: **a) Visually distinct triangle fan patches from the triangulation hole filler. b) The Buddha's armpit filled using the triangulation hole filler. This is an example of a complex hole which is difficult to analyze in the mesh domain.**

2.2.2 VRIP

The volumetric range image processor (VRIP) [5] is a well-regarded and widely used tool for research applications [10, 12, 16]. It provides both a merging algorithm as well as a volumetric hole filler. By working with the volumetric data, VRIP has more information available to it than hole fillers working with a mesh. The most important piece of additional information relates to the validity of volume areas as determined by their visibility from the range image scanner. Using the scanned data, and a process called space carving, VRIP is able to determine which areas are unseen by the scanner and thus will represent holes in the output mesh.

Space carving works by following the line of sight from an observed surface back to the scanner. As the scanner was able to see the surface along that path, the path can be assumed to be empty space and is marked as such. This creates three distinct states for the voxels of the volume: seen voxels are those representing the observed range image data, empty voxels are those that are cleared using space carving, and unseen voxels represent any remaining areas. In more practical terms, the seen voxels make up the exterior surface of the model, the empty voxels make up the area outside of the model, and the unseen voxels fill the interior of the model. A slice of the buddha volume which illustrates these voxel types can be seen in figure 3.1(a).

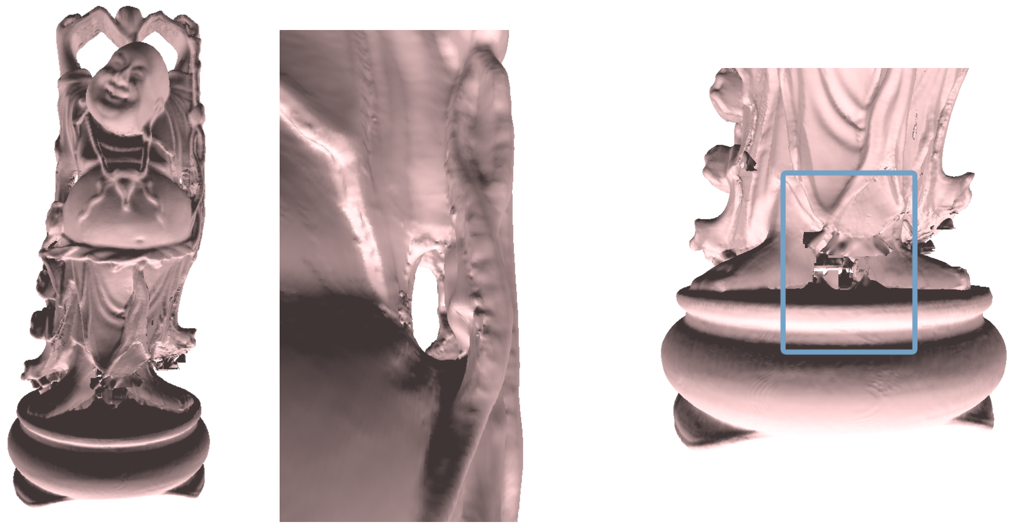
VRIP's hole filler works by extracting an isosurface from the unseen-empty boundaries as well as the observed surface. These boundaries appear in crevices and areas which the scanner has difficulty observing. This method has a number of strengths. Because VRIP works only with the volume data, it does not have to worry about problems inherent to mesh hole filling such as hole-spanning faces or overlapping vertices. Additionally, its results are always manifold as long as the extraction mechanism (marching cubes [13] in VRIP's case) always produces manifold meshes. VRIP also requires relatively little user tweaking. There are a number of parameters that can be changed, however for the majority of data the default values will produce an adequate product.

A number of optimizations are employed by VRIP to speed processing and minimize memory usage, but the noteworthy one with regard to hole filling is the use of run length encoding [9] to encode the volume. This allows for quick traversal across large homogenous areas, and also minimizes the memory footprint. We will take advantage of this optimization in our hole filling implementation as well.

The biggest shortcoming of VRIP as a hole filler is that it introduces a large number of additional handles. As shown by our previous work [7], the number of handles increases significantly with the amount of the model that VRIP is hole filling. Due to its widespread use, but relatively poor genus numbers, we use VRIP as the minimum benchmark for our results.

2.2.3 Volfill

The volumetric diffusion hole filling algorithm (commonly referred to as volfill) by Davis et al. [6] attempts to use a more intelligent approach than VRIP's to synthesize unseen surface areas of the model. It represents the model volumetrically as a signed distance function similar to VRIP, but attempts to create a surface which is continuous with the existing, seen model surface, in order to



(a) (b) (c)

Figure 2.2: a) A mesh extracted from a volume treated by Volfill. b) Volfill does a good job of handling small, complex areas, such as the Buddha's armpit. c) However, it creates significant noise and deviates into the wrong directions near large holes if other nearby holes are present. This behavior keeps it from creating the desired topology in the output mesh.

produce a more realistic surface geometry.

Using volfill, holes are filled by identifying gaps in the zero set of the signed distance function. These gaps, which would correspond to holes if an isosurface were extracted from the volume, are then closed by iteratively expanding the known surface into that area. The direction of expansion is determined by the distance to the existing isosurfaces in a specified vicinity.

In practice, this works well in small areas, as illustrated in figure 2.2(b). It successfully closes holes using a realistic looking geometry, using curves that are significantly more authentic looking than triangulation hole filling, and which also look better and have a smoother surface than VRIP filled holes. Volfill is much more robust in complex areas as well, accurately filling areas that contain chaotic surfaces, such as hair or the folds of robes.

Volfill, like the other algorithms, has its problems. The foremost issue is

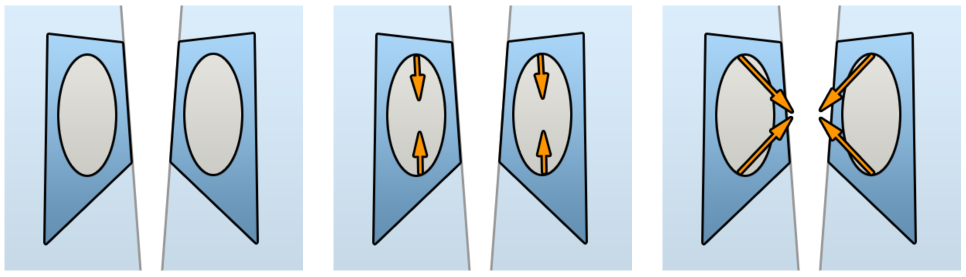

(a) (b) (c)

Figure 2.3: a) Two holes in close proximity. b) The desirable direction of expansion by volfill. c) The actual direction of expansion by volfill due to influence of the holes on each other.

that although the authors claim the algorithm is always capable of producing a volume which can be extracted to a manifold mesh, our tests showed that this mesh was often not the one desired. Large holes, such as that on the top of the Buddha's pedestal, were not closed completely (see figure 2.2(c)), even after a large number of iterations. The direction of expansion indicates that the hole was to be filled by merging the top of the pedestal with the bottom of the Buddha's robes, therefore eliminating desired topology. This example also illustrates volfill's inability to properly close large holes. The direction of expansion is influenced by all surrounding isosurfaces, so if an additional gap exists near one side of a large gap, the side of the gap will converge toward the smaller gap, not toward the other side of itself. An illustration of this situation is provided in figure 2.3, and a real-world example can be seen in figure 2.2(c), where the gap at the bottom of the Buddha's robe is converging toward the large gap in the pedestal. The two holes which are converging can be seen in the volume slice of figure 3.7(a). It is likely that much of this problem could be avoided by using the line of sight constraints as outlined in the volfill paper, however the implementation made publicly available by the authors does not provide this capability.

Also, unlike the previously mentioned hole fillers, volfill requires manual input. The user must specify the number of iterations to expand, where an insufficient number would keep it from filling even the most simple holes. Additionally, the distance for which expansion will proceed into a hole must be specified, thus requiring the user to have knowledge of how large the holes in the model's surface are. These, along with other parameters which are not easily obtained by the user, can have significant influence on the success or failure of the algorithm to fill holes.

2.3 Handle Reducers

Improved hole filling is one approach to reducing the number of handles in a mesh. An alternate approach is to try to remove the handles from a model after the holes have been filled. A number of methods have been devised for accomplishing this, some which operate on the mesh itself, and some which operate on the volume data used to create the mesh. We explore these works as we incorporate some of their techniques into our own implementation.

2.3.1 Smoothing

A methodology by Nooruddin and Turk [15] helps reduce handles by applying traditional 2D morphological operators to a volumetric representation of a model. The aim of their work is to produce more accurate models of reduced face counts than existing simplification methods. A key to their approach is that they do not preserve the topology of the original mesh. Although the aim of our work is not to simplify the mesh (in fact it is to preserve as much detail of the original as possible), Nooruddin and Turk's methods do effectively reduce handles, and we can adapt their algorithm to fit our needs.

The morphological operators are applied to the volumetric representation, which they obtain from existing meshes using two mechanisms unique to their work. As we are concentrating on improving the entire reconstruction pipeline, we have access to the volumetric data before a mesh has been extracted, and so these mechanisms are not relevant to this paper. What is relevant is that the volume they extract is necessarily solid (not thin-shelled), which is similar to the volumes we will be using from VRIP. The differences between these volume types can be seen in figure 2.4.

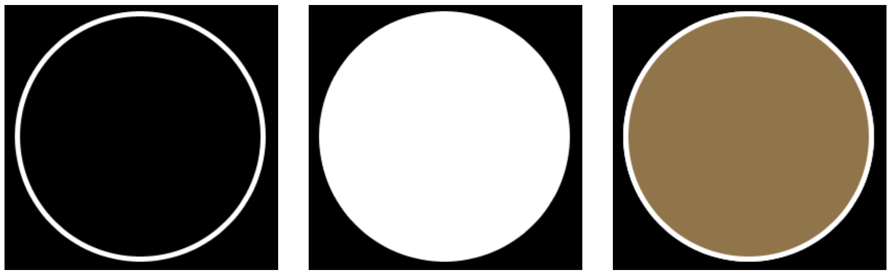

(a) Thin-shelled (b) Solid (c) VRIP

Figure 2.4: Slices through various types of volumes. White represents seen voxels, black represents empty voxels, and brown represents unseen voxels.

Specifically, their work makes use of the erosion and dilation morphological operators to reduce the topology of the model. The erosion operator contracts the volume within a given threshold, effectively shaving layers off of the volume. It is here that the solid model becomes necessary, as thin-shelled volumes would be destroyed once the shell has been eroded. The dilation operator is the complement of the erosion operator. This operator expands the volume by adding layers to it. The operators are generally used in conjunction, first contracting then expanding the volume, or vice versa. During this operation, small topological artifacts are smoothed out.

A key problem with this algorithm is that the process of voxelizing the model, then re-extracting a mesh, creates a product which is drastically different from the original. For Nooruddin and Turk, this is desirable as it produces a guaranteed manifold mesh, however it creates a significant loss of detail that is against the interests of our paper. This process is also complicated and expensive. For our work, non-manifold meshes are not a problem as we are addressing the data before it has been extracted into a polygonal representation, and so we can alleviate this problem by simply working with the volume data early on in the pipeline.

2.3.2 Handle Identification

One of the most successful handle removal tools is that of Wood, et al. [18]. Their approach utilizes both the extracted isosurface, and the underlying volume. Handles are identified by encoding the data as a Reeb graph and looking for cycles. The identified loops (handles) are then filled with a triangle fan. Additional precautions are used to preserve existing mesh detail and to avoid self-intersection in the resulting mesh.

This algorithm is the most accurate of any mentioned. Because it explicitly identifies handles, it can precisely remove them and can guarantee that they are all removed. Additionally, this precision allows for minimal modification of the mesh, thus preserving detail.

One of the main disadvantages of Wood's algorithm is that it is very slow. The process of identifying handles takes significant time, and so this algorithm takes orders of magnitude longer than the other algorithms. Also, by necessity, it requires the user to specify the maximum size of the handle to close, thus preserving handles which are part of the desired topology of the model (such as the Buddha's armpit).

It is also worth noting that this algorithm is a sort of last resort. It is used because the stages of the traditional reconstruction pipeline have failed to create a mesh without imperfections, and so these imperfections must be removed. It is the goal of this work to help improve these traditional reconstruction stages to help reduce the handles they introduce, thus minimizing the need for post processing tools such as this algorithm.

Chapter 3

Implementation

3.1 Inspiration

The aforementioned works present a wide variety of ideas for reducing the genus of a mesh, however none of these is perfect. We aim to combine the strengths of these approaches while minimizing their shortcomings.

One common theme for the existing bodies of work is their use of the volume data instead of the mesh data. The one approach that uses a purely mesh-based repair mechanism [7] hits a number of limitations which cannot be reasonably overcome without additional data. As an example, the problem of differentiating holes in the mesh on the ends of a tunnel from holes that happen to be aligned and need to be filled is nearly impossible to solve without some concept of how the original scan was made. By comparison, the volumetric data clearly represents the area as a tunnel through the model, or as an area unseen by the scanner. By modifying the volume rather than the mesh, we can take advantage of this additional data, such as normals and line of sight information provided by the original range images.

Working in the volumetric domain also eliminates the need to do complex mesh surgery. Changing the mesh requires taking care to preserve manifoldness and avoid self-intersection. These are two problems that have many caveats, as outlined in the problem description section. Modifications to the volume do not require taking these precautions, as a well-behaved mesh extraction mechanism [13] is guaranteed to produce a manifold, non-intersecting mesh.

Unlike some existing works, we do not bind ourselves with the requirement of being able to work with an existing mesh, and so we have the freedom to work on the volume data as desired. We also note that even with this requirement, research such as [15] still converts the mesh to the volumetric domain in order to leverage the advantages of this representation. Our goal is to improve hole filling to produce a cleaner mesh after a single run through the reconstruction pipeline, and as the data passes through a volumetric representation in the pipeline, we will focus on this period.

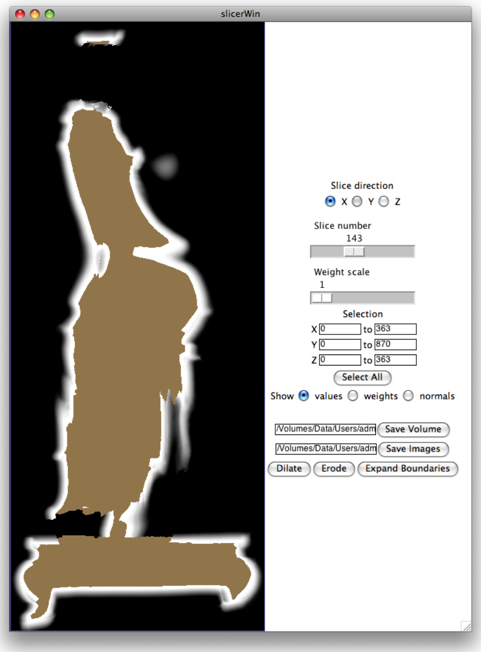
Finally, we choose to work in the volumetric domain because it is earlier in the pipeline, and so we can leverage additional hole filling techniques at later stages. By applying volumetric hole filling techniques, we do not preclude ourselves from applying existing volumetric hole fillers, or mesh based hole fillers. Given this ability, our work is not required to fully solve a problem which has already been partially solved, it only needs to improve the product of the existing techniques.

We build our solution upon the discussed solutions, supplementing them as necessary. In particular, we make heavy use of VRIP since it provides the merge and reconstruction phases of the reconstruction pipeline. For the raw data and its alignment we use range images from the Stanford 3D Scanning Repository [2], which come pre-aligned by previous researchers using a tool called Scanalyze [1]. By interspersing our techniques into the steps taken by VRIP we have access to the data in its volumetric representation, and are also provided with a marching cubes implementation to extract a manifold mesh from our modified volume.

In order to create our improved hole filler, we start by modifying the technique of Nooruddin and Turk [15]. We adapt variations on their erosion and dilation operators to work with the volume data provided by VRIP. Unlike their solid volume, the VRIP volume has voxels of three types: seen, unseen, and empty. Because the unseen voxels comprise the inner content of the volume, with the seen voxels providing the volume's shell, we can effectively operate on the volume as if it were solid by treating both of these types of voxels as model voxels. We do not neglect the differentiation between seen and unseen voxels however, as this provides a valuable piece of information which is absent in the Nooruddin and Turk work. Their original implementation applied the morphological operators to the entire volume, thus losing detail uniformly across all areas, even where smoothing was not necessary. Using our knowledge of the unseen areas of the volume, we can apply the operators only to the unseen-empty boundaries, as these areas will correspond to areas that will require hole filling (and which have the most extraneous handles) later in the pipeline. By limiting the scope of the operators we avoid the loss of detail in areas which do not correspond to holes.

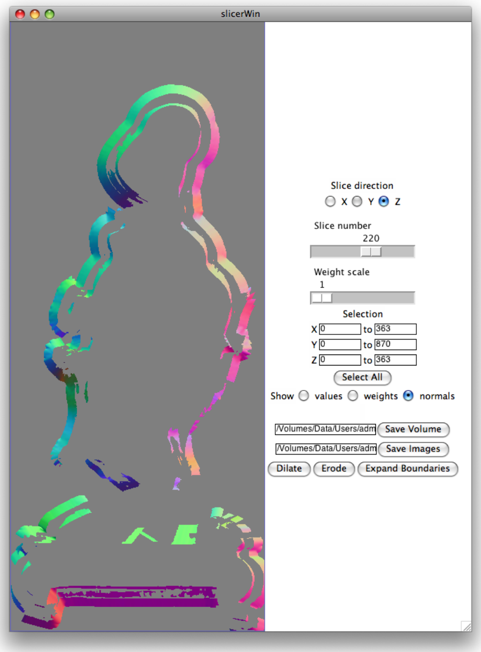
We derive a third operator that uses the surface normals to expand the isosurface into the hole areas. This operator is inspired by volfill, and has the similar aim of expanding the isosurface based on the existing surface, however it uses a different implementation. Volfill expands the isosurface by slowly blurring it outward into gaps. The direction of the blur is based on the distance to the nearest existing isosurface. Our approach expands more concretely by expanding perpendicular to the normal of a voxel as determined by the scanner line of sight. This method avoids interference from nearby gaps as the volfill algorithm is prone to, since the direction of expansion is based on the known surface, not on the gaps. Unfortunately, this implementation too has issues which will be discussed in the results section.

Even using constraints such as line of sight and unseen-empty boundaries, there are times when the lack of scanner information limits the effectiveness of an algorithm due to interference by the surrounding data. Additionally, there are times when applying an algorithm is more detrimental in certain areas than it is helpful (such as a smoother which might erase detail in an already filled area). To help combat this, we provide the ability to apply our operators only to selected areas of the volume. This requires user knowledge and interaction with the filling process, but is optional and can lead to improved output.



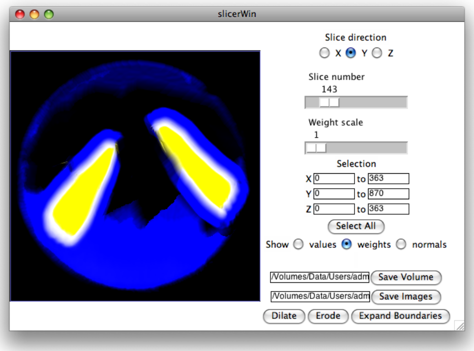
Slice direction [✓ X] Y Z
Slice number 143
Weight scale 1
Selection
X 0 to 363
Y 0 to 870
Z 0 to 363
Show [✓ values] weights normals
/Volumes/Data/Users/adm
/Volumes/Data/Users/adm

(a)



Slice direction X Y [✓ Z]
Slice number 220
Weight scale 1
Selection
X 0 to 363
Y 0 to 870
Z 0 to 363
Show values weights [✓ normals]
/Volumes/Data/Users/adm
/Volumes/Data/Users/adm

(b)



Slice direction X [✓ Y] Z
Slice number 143
Weight scale 1
Selection
X 0 to 363
Y 0 to 870
Z 0 to 363
Show values [✓ weights] normals
/Volumes/Data/Users/adm
/Volumes/Data/Users/adm

(c)

Figure 3.1: Our user interface. a) An x-axis slice displaying the empty (black), unseen (brown), and seen (white) voxels of the Buddha volume. b) A z-axis slice displaying the normals for the Buddha volume. c) A y-axis slice of the Buddha's feet displaying the voxels' confidence values.

3.2 Interface

To provide the user interface with which to apply our operators, we modify the vripslicer tool provided with the VRIP [5] project. This tool provides a basic volume viewer to which we add a number of capabilities. On top of the voxel type view (figure 3.1(a)) provided in the basic tool, we add a normals view (figure 3.1(b)), and fix the implementation of the weights view (figure 3.1(c)) which gives the confidence values for each pixel. The VRIP provided implementation of the weights view is non-functional.

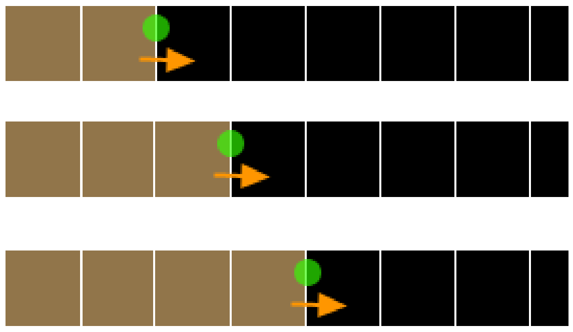
We add two capabilities for use specifically with our work. Notably, we add a selection mechanism (as discussed further below), which is available both through numeric controls and through mouse selection. Additionally, we add buttons to apply our operators to the currently selected voxels. These buttons apply one iteration of their respective operator, with the effects immediately visible in the volume view.

Finally, we provide two mechanisms to save the results of user-applied operations. Controls are provided to save either the modified volume to a specific path, or to save images for each slice of the currently viewed axis to a directory.

3.3 Morphological Operators

As discussed, there is a large body of work aimed at effectively filling holes. Our goal then, is not to create another hole filler, but to improve the results of existing fillers by limiting their use to situations where they excel. We accomplish this goal by applying the following morphological operators to the model while in its volumetric representation.

The operators are applied to a copy of the volume which is then swapped for the original when a sweep is done. This double buffering technique keeps changes that have already been made from interfering with future comparisons in the same sweep. Without this precaution, expansions or dilations would run



(a)

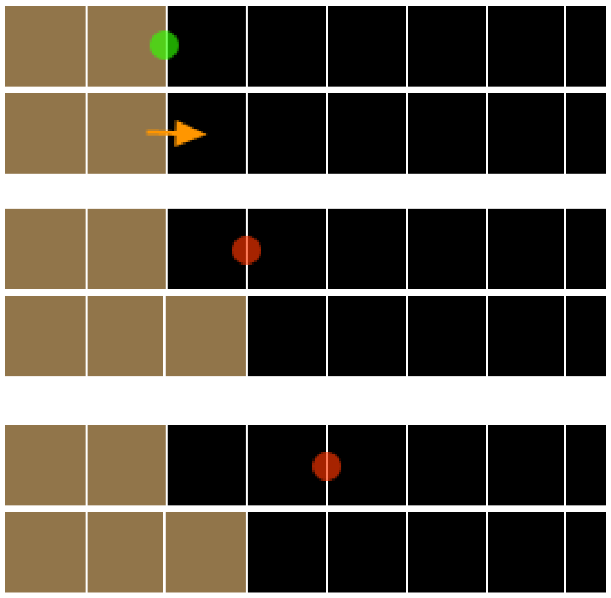
(b)

Figure 3.2: a) If updates are done in-place then as we sweep from left to right we are always on the boundary and expansion will go unchecked b) The expansion will fill the entire scanline.

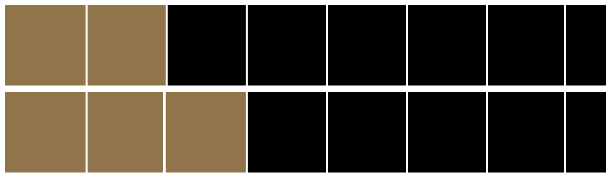
unbounded within a single sweep. Consider the simplified case of only one dimension: iterating from left to right across three unseen voxels followed by three empty voxels. If the algorithm reaches the barrier and expands the unseen voxels into the empty voxels (to create four unseen voxels followed by two empty ones), without the double buffering, the algorithm would then encounter the barrier again on the next iteration (and again make the expansion). This unbounded expansion is illustrated in figure 3.2. By updating a different volume than we are reading from, we avoid this problem, as in figure 3.3.

3.3.1 Erosion and Dilation

Our two primary operators are adaptations of the dilation and erosion operators implemented by Nooruddin and Turk [15]. These operators expand and contract the volume, respectively, which serves to merge or remove small "noise" voxels around the volume, and to smooth out rough areas. The erosion operator can be used to remove noise voxels by contracting them into nothing as shown in



(a)

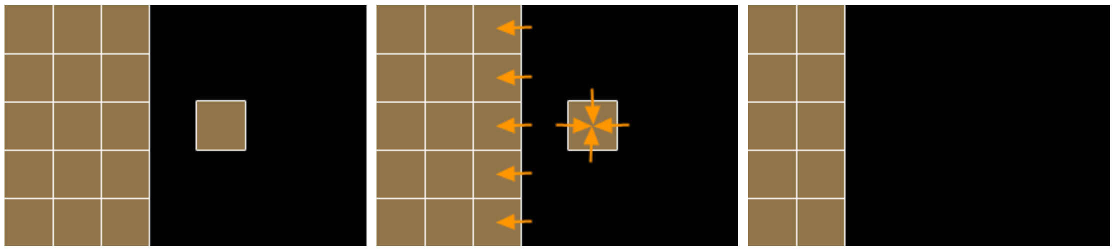


(b)

Figure 3.3: a) By using one volume for reading (tops) and one for writing (bottoms), we only make incremental updates with each sweep. c) The resulting scanline won't be expanded more than one voxel away from the volume with each sweep.

figure 3.4. Alternatively, the dilation operator can be used to remove them if they are near the main volume by expanding the volume until the noise is merged into it as in figure 3.5. In practice, both operators are usually used in tandem to avoid shrinking or growing the volume significantly. The effects of these operators is discussed further in the results section.

In the implementation given by [15], the operators are applied to the entire volume, which given our goal has the undesirable side effect of smoothing the entire model when we only want to smooth the hole filled areas. Our implementation therefore limits the application of these operators to the boundaries of unseen-empty voxels in the VRIP-generated volume, as these areas correspond to areas that will require hole filling in the extracted mesh. Additionally, because our volume has voxels of three states (seen, unseen, and empty) instead of the two-state (empty and not-empty) voxels of Nooruddin and Turk's volume, we cannot implement the dilation operator by simply inverting the volume and applying the erosion operator. Instead, we reverse our test as described below, and add voxels to the volume rather than removing them.


(a) A single application of the erosion operator

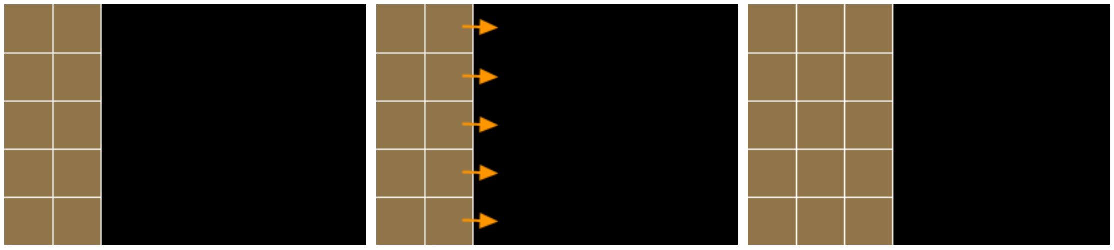
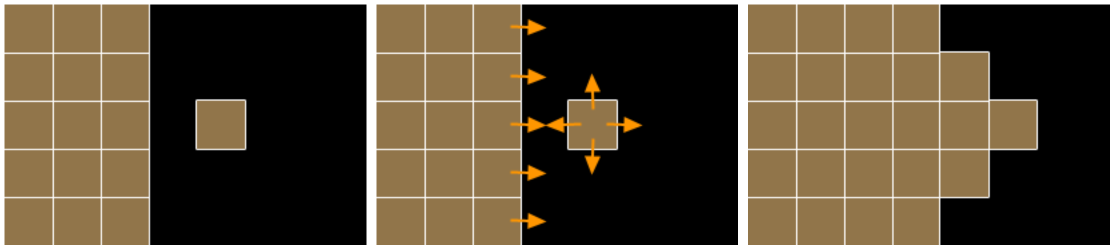

(b) A single application of the dilation operator

Figure 3.4: The removal of a floating "noise" voxel using the erosion operator followed by the dilation operator.


(a) A single application of the dilation operator

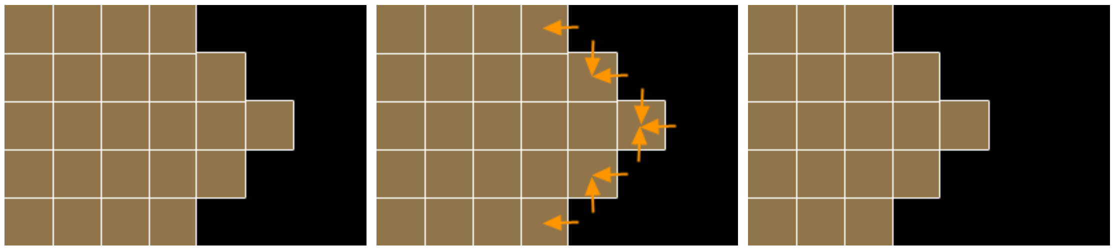

(b) A single application of the erosion operator

Figure 3.5: The merging of a floating "noise" voxel using the dilation operator followed by the erosion operator.

The erosion operator iterates the voxels of the volume and where it finds an unseen voxel adjacent to an empty voxel, it replaces the unseen voxel with an empty one. Adjacent voxels are those six voxels which share a side with the voxel being evaluated.

Erosion Operator

```
Create a buffer volume
bufferVolume = copy of volume

Iterate the voxels
for (z = 1 to dz - 1)
  for (y = 1 to dy - 1)
    for (x = 1 to dx - 1)
      if (voxel at (x, y, z) is unseen and any adjacent voxel is empty)
         set type of voxel in bufferVolume at (x, y, z) to empty

replace volume with bufferVolume
```

The dilation operator is implemented similarly, with the role of unseen and empty voxels swapped:

Dilation Operator

```
Create a buffer volume
bufferVolume = copy of volume

Iterate the voxels
for (z = 1 to dz - 1)
  for (y = 1 to dy - 1)
    for (x = 1 to dx - 1)
      if (voxel at (x, y, z) is empty and any adjacent voxel is unseen)
         set type of voxel in bufferVolume at (x, y, z) to unseen
```

```
replace volume with bufferVolume
```

3.3.2 Isosurface Expansion

The isosurface expansion operator aims to expand the existing isosurface into holes in the model, a goal similar to that of volfill. This operator expands the surface by using the normals of voxels in the existing isosurface to determine the direction in which to expand. As these normals are not available in the default implementation of VRIP, the implementation had to be augmented to calculate and store the normal for each voxel where possible. To store the normal, three unsigned chars (eight bytes each) were used, one for each dimension. This enlarged the volume's disk footprint by approximately 75% as the three bytes for the normal were added to the existing four bytes used to store value and weight (each a two byte short). In practice this percentage held true as our test Buddha grew from 113.6 megabytes to 196.5 megabytes (a 73% growth).

The operator expands the isosurface from seen voxels to voxels which are approximately perpendicular to the normal of the seen voxel, as outlined by the following pseudo-code:

Isosurface Expansion Operator

```
Create a buffer volume
bufferVolume = copy of volume

Iterate the voxels in the volume
for (z = 1 to dz - 1)
  for (y = 1 to dy - 1)
    for (x = 1 to dx - 1)

      Find a seen voxel
      if (voxel at (x, y, z) is a seen voxel and has its normal set)
         normalize voxel normal <nx, ny, nz>

         Look at the surrounding voxels
         for (zm = -1 to 1)
           for (ym = -1 to 1)
             for (xm = -1 to 1)

               Find a neighbor which is not already part of the isosurface
               otherVoxel = voxel at (x+xm, y+ym, z+zm)
               if (otherVoxel is unseen or empty)

                  Determine if neighbor is perpendicular to normal of the current voxel
                  normalize vector to surrounding voxel <xm, ym, zm>
                  dotProduct = <xm, ym, zm> · <nx, ny, nz>
                  if (cos(±112.5) < dotProduct < cos(±67.5))
                     set otherVoxel in bufferVolume to element at (x, y, z)

replace volume with bufferVolume
```

This operator is noticeably more complex than the erosion and dilation operators, and warrants further explanation. As mentioned previously, any changes are made to a separate copy of the volume being examined. This copy is created on line 1, and is swapped for the original after execution of the operator on line 16. Lines 2-4 iterate every voxel in the volume. As will be described in the Optimizations section, the efficiency of this iteration is greatly improved by taking advantage of the RLE encoding. We iterate the voxels until we find a seen voxel (line 5), as this represents the isosurface that we want to expand. We therefore ignore unseen and empty voxels during this iteration. As we require a normal to determine the direction to expand into, we also ignore voxels which do not have their normal set.

Once we have a seen voxel to work from, we examine its eight immediate neighbors (those comprising the cube around it). The iteration of these neighbors is handled by lines 7-9. For each neighbor, we first determine if it is not already part of the isosurface, and therefore is a viable voxel to expand into (line 11). If the voxel is either empty or unseen, then we need to determine if it is approximately perpendicular to the normal, and therefore is inline with the existing isosurface. We determine this by taking the dot product of the normal vector for the current voxel with the vector from the center of the current voxel to the center of the neighboring voxel being examined. Because both of these vectors are normalized (lines 6 and 12), the dot product has unity magnitude, and its value directly represents the angle between the vectors. As the normal vector is presumed to be perpendicular to the isosurface, we want to expand into voxels that are approximately perpendicular to the normal. Therefore, if the dot product of our two vectors is approximately 90 degrees (we allow for a 22.5 degree variance in either direction), then we expand our isosurface into that voxel (lines 14 and 15). A two dimensional representation of what is done for each voxel is given in figure 3.6.

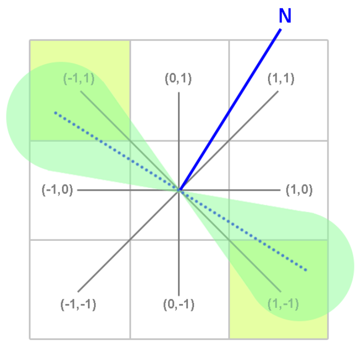
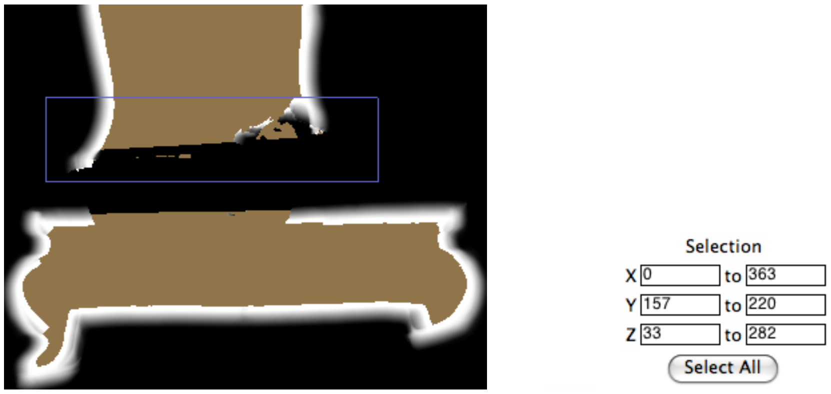


Figure 3.6: Voxels that are approximately perpendicular to the normal N will be expanded into if necessary. The dotted line represents the gradient of the current voxel, and the gray lines represent the vector from the center of the current voxel to the surrounding voxels. The yellow squares represent voxels that would be considered for expansion.

3.4 User Selection

Although the ideal hole filler can operate with minimal or no user intervention, allowing the user to limit the scope of operators to specific areas often improves results. In practice, we see improved genus numbers with this technique for our operators, but also note that this ability would aid other hole fillers as well. Specifically, the problem we describe with volfill of incorrectly determining which boundary edges to merge would be avoided if the user could limit the algorithm's scope to only look in the area of the same hole. For our example, this would involve selecting the top of the Buddha's base to fill the large hole across its top, then selecting the bottom of the Buddha's robe (similar to figure 3.7(a)) to fill the hole there.

We provide the user this ability in an easy to use fashion by allowing for selection in the volume viewer of the user interface. A basic mouse drag-and-release sets the scope in which operators will be applied. A selection rectangle is provided for visual feedback (figure 3.7(a)). The viewer allows for viewing from any of the three axes, and so the user can choose which view to make his or her selection from. The user's selection rectangle dictates the selection scope for



X 0 to 363
Y 157 to 220
Z 33 to 282

(a) (b)

Figure 3.7: a) Areas of user selection are visibly highlighted in the volume viewer. b) Numeric controls are also available for precise information and selection.

the two axes which are not the view axis. Thus, if viewing along the z-axis, the height of the selection represents the y-axis selection and the width represents the x-axis selection. The entirety of the view axis is selected within that scope. Put another way, all slices (depths) being viewed are selected. Numeric controls (as shown in figure 3.7(b)) are available if the user wants to limit this third dimension (or wants to more precisely limit the other dimensions). The selection rectangle accurately represents the selection across view axis changes, so the user can clearly see what is selected within the volume.

The implementation of the scoping is straightforward. Lines 2 through 4 of each of the operators are modified to iterate only within the bounds. The modified lines look similar to the code below, where start_x, start_y, start_z, stop_x, stop_y, and stop_z specify the selection area as taken from the user interface.

```
02 for (z = start_z to stop_z)
03   for (y = start_y to stop_y)
04     for (x = start_x to stop_x)
```

3.5 Optimizations

A naive implementation of our operators would consume considerable memory resources and take considerable processor time to complete. To limit these impacts, we employ a number of optimizations, such as caching and writing only modified scanlines.

As the volumes we use are created by VRIP, they are encoded using run length encoding (RLE) [9], which limits the memory and disk space required to store the volume. Unfortunately, this makes updating the volume a very slow process since the encoding needs to be re-evaluated after each write. To remedy this, we use a scanline cache that stores the scanline being evaluated and its eight neighboring scanlines in their raw, indexable format. Thus, the cache stores a three by three block of scanlines; this block size is arbitrarily expandable in our implementation, but as none of our operators evaluate voxels that are not their immediate neighbors, there is no need for anything larger.

The cache provides a number of significant advantages. The raw format of the scanlines in the cache means that they are directly indexable, unlike RLE encoded ones, which must be progressively evaluated in order to find the value of a given voxel. Our operators' implementations iterate the scanline linearly, so the speed advantage here is minimal over a iterating an RLE scanline (and may actually be slower, as discussed below), however the direct indexing leads to cleaner code, and is significantly faster than the default implementation of voxel indexing in VRIP which evaluates the RLE for each read.

The real strength of the raw format is its speed for writing. Writing a voxel in VRIP's default implementation of the volume object required that the existing scanline be converted to its raw format, the specific voxel value changed, the raw scanline be re-encoded using RLE, and then the original scanline be replaced with the new one. This is an expensive process both in the processor time it takes to convert the scanline back and forth, and in memory as entire new scanlines must be allocated for each write that occurs within them. Using the cache, a scanline is converted to raw once when being read into the cache, and is converted back to RLE at most once for each time it is needed in the cache. Whereas the naive implementation had the potential to read, convert, update, re-convert, and write the scanline once for each voxel in the line, this process occurs at most once for all voxels in the line using the cache.

An effort is also made to minimize the number of scanlines that need to be read into and written from the cache. Because our iterations are done one scanline at a time (iteration of the first dimension iterates voxels in a scanline, iteration of the second dimension iterates scanlines in the slice, and iteration of the third dimension iterates slices in the volume), there is strong spacial locality, and most of the cache does not require updating with each iteration. A naive implementation might flush the cache each time a new scanline is to be evaluated, but by intelligently evaluating the location of the new scanline with relation to the scanline previously centered in the cache, only a small portion of the cache needs to be refreshed in most cases. For our three by three scanline block, six of the nine scanlines need only be shifted (a very cheap operation that requires no new memory) rather than re-read the majority of the time. At times when the slice changes the cache does require a full refresh.

Similar to reads, the cache limits writes only to those times when needed. By taking advantage of the encapsulation provided by the cache object, and by maintaining a dirty bit for each current line, the cache can keep track of which lines have changed since being read in. By not writing lines that have not changed back to the volume, the process of reallocating a new RLE scanline object and reevaluating its encoding from the raw scanline is completely avoided. Additionally, writes for a scanline are only done before it is to be removed from the cache, so a given scanline could be updated during the evaluation of multiple scanlines before ever actually being written out.

As mentioned previously, updating the volume in place leads to problems, thus our cache must support our double buffering approach. The cache maintains references to both versions of the volume, and internally maintains two copies of the cache, one for reading and one for writing. Both copies are shifted and refreshed as needed. We acknowledge the opportunity to employ the copy-on-write optimization here, thus avoiding needing to create an additional write copy unless a modification needs to be made, however our current implementation does not utilize this.

Finally, we take advantage of the fact that the volume is RLE encoded. This encoding allows for very fast traversal of the volume. As an example, a scanline which contains only empty space can be skipped after reading only a single value, rather than the hundreds of values present in a raw representation. We use this in tandem with the cache, taking advantage of the RLE to quickly find a viable voxel to evaluate, then using the cache to quickly read from and make modifications to the volume.

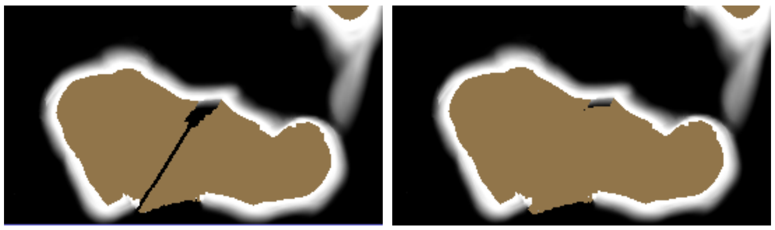
Chapter 4

Results

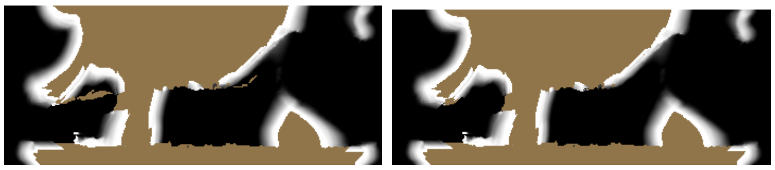
Using our operators, we successfully reduce the handle counts of the models we treat, while maintaining or improving the visual quality of the untreated version.

Most of our success comes through the use of the dilation and erosion operators. These operators serve to "clean" areas of the the volume which were partially occluded from the scanner, and therefore have jagged or irregular data. There are three distinct ways in which this cleaning occurs: merging of data, removal of noise, and smoothing of unseen surfaces.

Often times, the line of sight for the scanner is occluded for an area with the exception of a very small window. As shown in figure 4.1(a), this leads to tunnels of emptiness in otherwise unseen areas. Because this is an empty-unseen boundasry, VRIP applies its hole filler to create a surface along the area. This is an unnecessary additional surface which is an opportunity for additional handles. Furthermore, these tunnels manifest themselves as small pinholes in the model, which require a large number of polygons to represent, and which are visually undesirable. Using the dilation operator, we are easily able to remove these tunnels, thus avoiding the need for additional hole filling or topology. Because the sides of the tunnel eventually come together, future erosions can be used on the outer parts of the model without reopening the tunnel.



(a)



(b)

Figure 4.1: a) A tunnel in the dragon model before and after being cleaned. b) Unseen "noise" under the Buddha's robe before and after being cleaned.

In a similar manner to the way tunnels are eliminated, noise around the model (such as in figure 4.1(b)) can be removed. Either the dilation or erosion operator can be used here, although usually both are used in tandem in order to avoid growing or shrinking the model significantly. By starting with the erosion operator, small outlying unseen voxels will be reduced to nothing, after which the dilation operator can be used to restore empty-unseen boundaries in the main volume to their original levels. This operation was illustrated in figure 3.4. Conversely, the dilation operator can be used first to grow the noise and main volume until they merge together. After being merged, the erosion operator can be used to shrink the new boundary to a level consistent with the surrounding seen areas. This can be seen in figure 3.5.

Finally, the operators can be combined to smooth unseen-empty boundary areas. By applying the operators in alternating sessions, small imperfections in the problematic areas are smoothed out.

We had a lesser degree of success with the boundary expansion operator. The issue with this operator is that it requires the normals to be present on

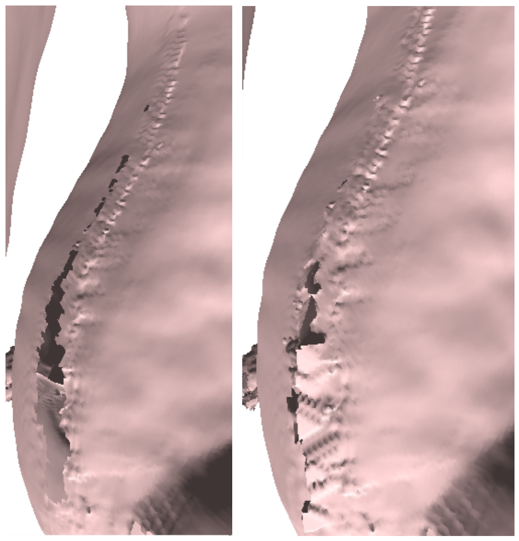


Figure 4.2: The rear of the Dragon's body with no hole filling or treatment (left) and after treatment using only the boundary expansion operator (right).

the surrounding seen areas in order to do expansion, and this is often not the case in areas where expansion is needed. Unfortunately, the majority of areas that are out of view of the scanner (and therefore require fixing) are out of view because they are occluded by a sharp edge. This type of area, such as under the Buddha's robe (see figure 3.7(a)), is discontinuous with the normals field present on the outer part of the edge, and so cannot be filled in using these normals.

The expansion operator did perform decently well in areas where one part of the model occluded part of a smoother surface. Examples of this type of area are the top of the Buddha's pedestal, which is flat, but blocked from the scanner by the Buddha's body, and the front part of the Dragon's rear underbelly, which is blocked by the front of the Dragon's body. As seen in figure 4.2, the operator was able to close small holes successfully, but because only the normals closest to the hole's edge are used to determine the direction of the newly created surface, it is not the best suited for filling surfaces with curvature.

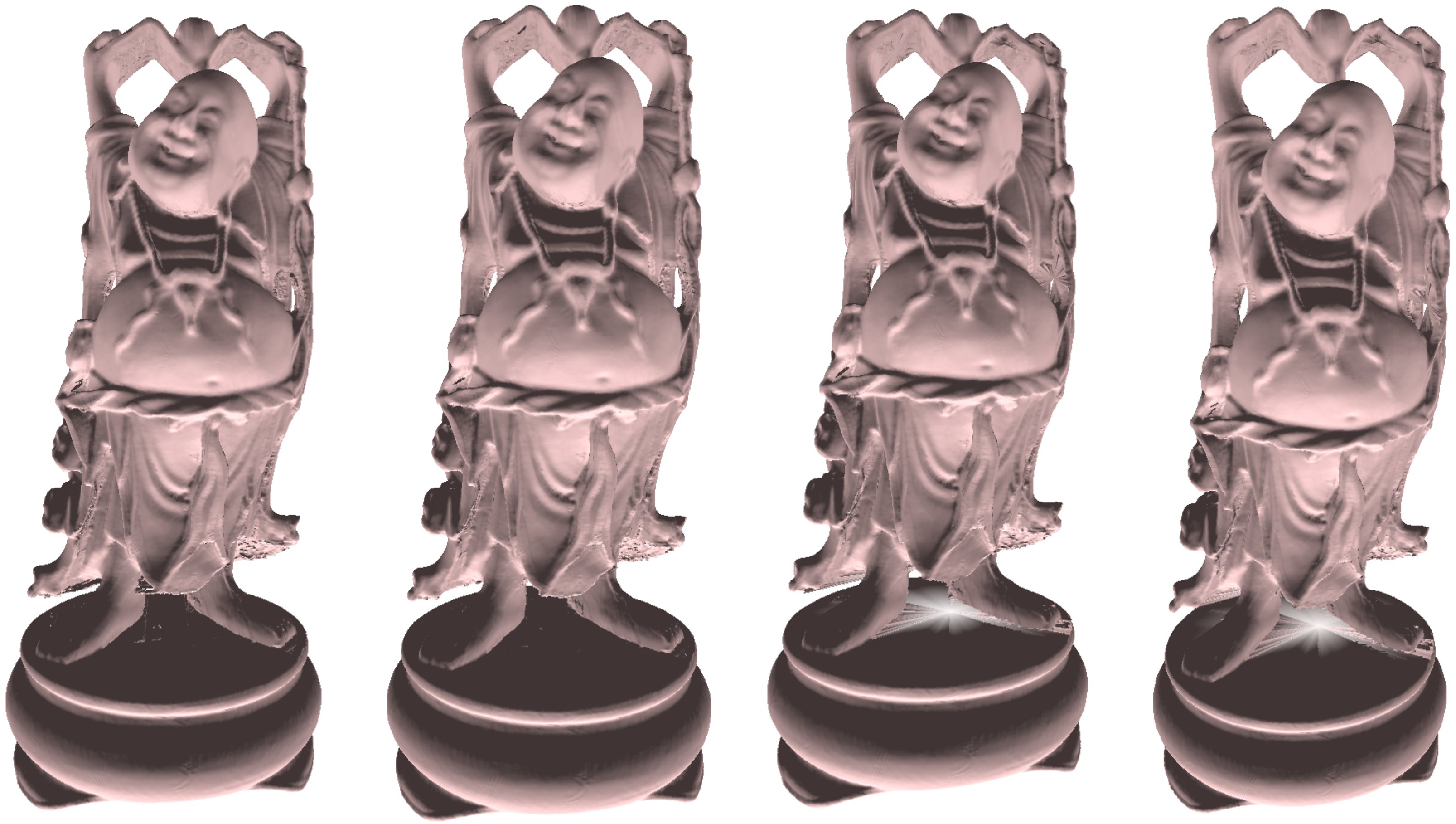
By combining the operators we were able to consistently generate meshes of a lower genus than when no treatment was used. As seen in table 4.1, we were able to reduce the genus by 47.8% for the Buddha using the VRIP hole filler, and

Model	Our Operators	Hole Filler	Vertices	Faces	Genus
Buddha	No	VRIP	1564955	3130174	67
Buddha	Yes	VRIP	1551328	3102890	35
Buddha	No	Triangulation	1436056	2872220	28
Buddha	Yes	Triangulation	1436536	2873188	28
Dragon	No	VRIP	1406194	2812540	39
Dragon	Yes	VRIP	1392210	2784548	33
Dragon	No	Triangulation	1354531	2709078	5
Dragon	Yes	Triangulation	1353775	2707562	4

Table 4.1: Genus counts for treated and untreated volumes using two different hole fillers.

15.4% for the Dragon using the VRIP hole filler. Because our operators reduce handle counts by removing noise around holes, they had little effect on the genus count when using the triangulation hole filler. This is because this hole filler first removes junk faces, then creates the simplest patch possible, and so is not influenced by noise.

There was also no loss of visual quality given the use of our operators. As shown in figure 4.3, the treated and untreated versions appear very similar, and for the triangulation hole filler, our operators improved the ability of the hole filler to fill the Buddha's armpit without patching it (figure 4.4). Figure 4.5 shows in detail the differences between the VRIP and triangulation hole fillers, where the first leaves a slightly uneven surface across the top of the pedestal, while the second has the distinct starburst pattern. For the VRIP hole filler, the treated mesh is slightly smoother across the top of its pedestal. Similar to the Buddha, figure 4.6 shows that we were able to reduce the genus count of the Dragon model without reducing its visual appeal.



(a) genus 67 (b) genus 35 (c) genus 28 (d) genus 28

Figure 4.3: a) Unmodified, VRIP hole filler b) Cleaned, VRIP hole filler c) Unmodified, triangulation hole filler d) Cleaned, triangulation hole filler

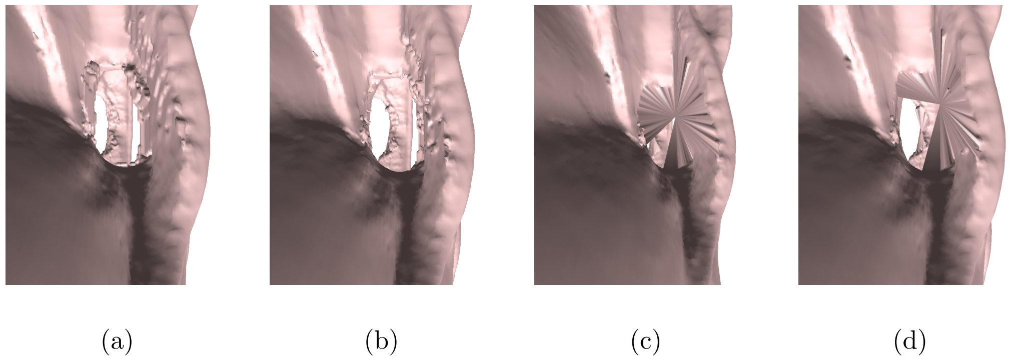


Figure 4.4: a) Unmodified, VRIP hole filler b) Cleaned, VRIP hole filler c) Unmodified, triangulation hole filler d) Cleaned, triangulation hole filler

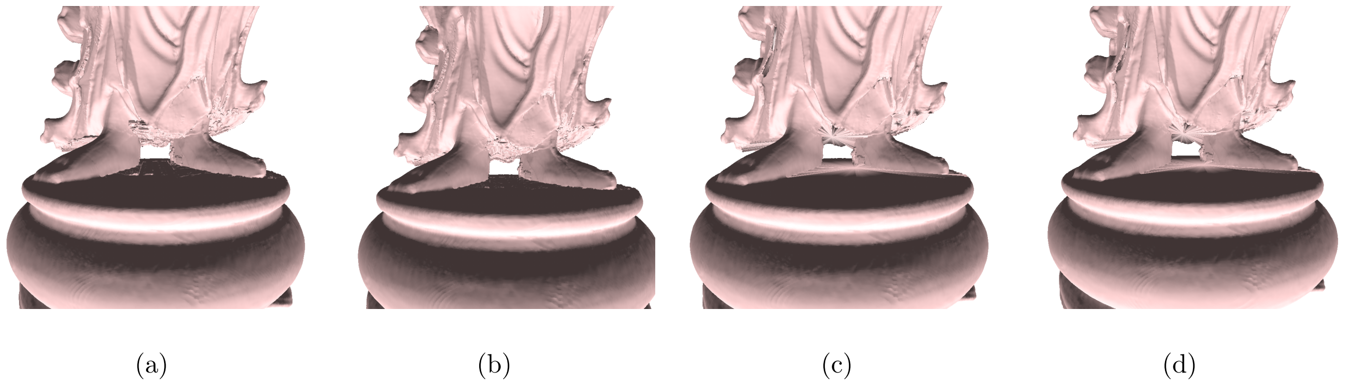
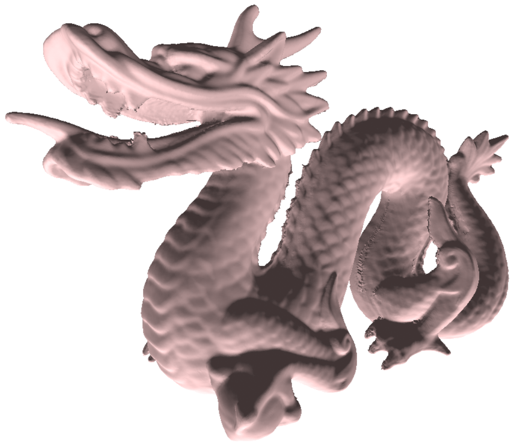
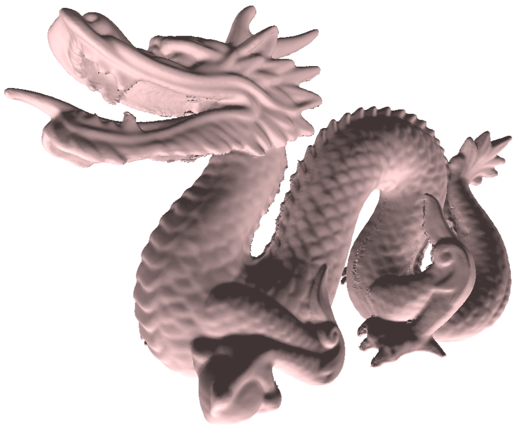


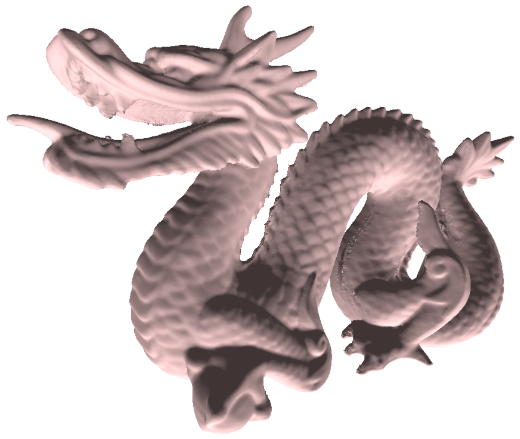
Figure 4.5: a) Unmodified, VRIP hole filler b) Cleaned, VRIP hole filler c) Unmodified, triangulation hole filler d) Cleaned, triangulation hole filler



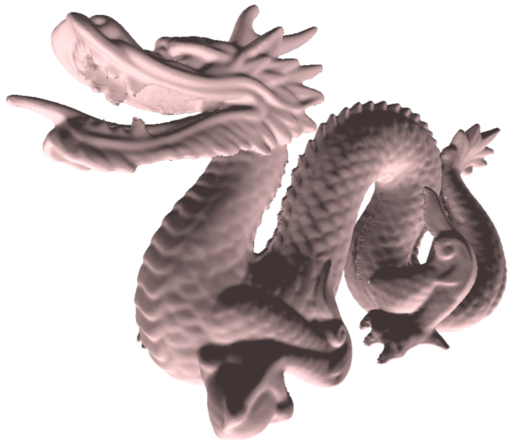
(a) genus 39



(b) genus 33



(c) genus 5



(d) genus 4

Figure 4.6: a) Unmodified, VRIP hole filler b) Cleaned, VRIP hole filler c) Unmodified, triangulation hole filler d) Cleaned, triangulation hole filler

Chapter 5

Future Work

Our work furthers the progress toward the creation of meshes without topological noise, however we acknowledge that there is still much research to be done.

The operators we apply to the volume are aimed at cleaning up the volume for hole filling later in the pipeline, but at this point there is already some extra topology which we are not addressing. Using the triangulation hole filler we still produce Buddha meshes of genus 28, which is significantly higher than the desired six. As this hole filler is incredibly simple, this indicates that the extra handles are in areas other than the unseen-empty areas which we are treating. The likely culprit for introducing this topology is the alignment stage, which warrants further analysis. When aligning the range images ourself we note a 21% decrease in genus (22 vs. 28) over the Stanford aligned images using the triangulation hole filler, indicating that investigation of this stage would be worthwhile.

We also believe that the expansion algorithm used in volfill [6] carries a lot of potential, and could be combined with our research to produce improved results. We suggest two specific enhancements: creating a volfill operator, and enhancing that operator by incorporating the normals as our boundary expansion operator does.

Unfortunately, the existing volfill does not integrate well with our operators. Volfill cannot read our volumes that have been enhanced with the normals data, and so we cannot first apply our operators, then run volfill on the volume. Conversely, we cannot run volfill then apply our operators because the volumes produced by volfill use two-state (seen and empty) voxels and strip the information about which voxels are unseen, an essential element for our operators. An operator similar to our existing operators could be created which employs the volfill algorithm. This would allow integration with our other operators, and also allow for the use of user defined scoping of the operator's application. This would greatly enhance volfill as it currently confuses the areas of one hole with another, causing the results discussed previously.

Finally, volfill could be enhanced to take advantage of the normals data which we have provided with our existing boundary expansion operator. By weighting the data perpendicular to the normals more heavily, volfill could avoid being influenced by holes other than the one it is currently examining.

Bibliography

[1] Scanalyze: a system for aligning and merging range data. `http://graphics.stanford.edu/software/scanalyze/`.

[2] The stanford 3D scanning repository. `http://graphics.stanford.edu/data/3Dscanrep/`.

[3] P. J. Besl. Active, optical range imaging sensors. *Machine Vision and Applications*, 1:127–152, 1988.

[4] B. Curless. From range scans to 3D models. *Computer Graphics*, 33(4):38–41, Nov. 1999.

[5] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *SigGraph-96*, pages 303–312, 1996.

[6] J. Davis, S. R. Marschner, M. Garr, and M. Levoy. Filling holes in complex surfaces using volumetric diffusion. In *3DPVT02*, pages 428–861, 2002.

[7] E. Firestone, C. Povey, and Z. Wood. Locating the source of topological error in reconstructed 3d models. In *Proceedings of Electronic Imaging '08*, San Jose Convention Center, San Jose, CA, Jan. 2008.

[8] L. Freitag, M. Jones, and P. Plassmann. An efficient parallel algorithm for mesh smoothing. Technical report, Nov. 30 1995.

[9] E. Furuta. Run length encoding and decoding methods and means. United States Patent 4,096,527, June 1978.

[10] H. Hoppe. Progressive meshes. *Proceedings of SIGGRAPH '96*, pages 99–108, 1996.

[11] R. Jones. *Garbage Collection: Algorithms for Automatic Dynamic Memory Management*. John Wiley and Sons, July 1996. With a chapter on Distributed Garbage Collection by Rafael Lins. Reprinted 1997 (twice), 1999, 2000.

[12] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. E. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The digital michelangelo project: 3D scanning of large statues. In *SIGGRAPH*, pages 131–144, 2000.

[13] W. Lorensen and H. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics*, 21(4):163–169, 1987.

[14] J. B. A. Maintz and M. A. Viergever. A survey of medical image registration. *Medical Image Analysis*, 2(1):1–16, 1998.

[15] F. S. Nooruddin and G. Turk. Simplification and repair of polygonal models, May 09 2003.

[16] S. M. Seitz and C. R. Dyer. Photorealistic scene reconstruction by voxel coloring. *International Journal of Computer Vision*, 35(2):151–173, Nov. 1999.

[17] F. M. Weinhaus and V. Devarajan. Texture mapping 3D models of real-world scenes. *ACM Computing Surveys*, 29(4):325–365, Dec. 1997.

[18] Z. Wood, H. Hoppe, M. Desbrun, and P. Schröder. Removing excess topology from isosurfaces. In *ACM Transactions on Graphics*, volume 23(2), pages 190–208. ACM press, 2004.