LOCATING THE SOURCE OF TOPOLOGICAL ERROR IN RECONSTRUCTED 3D MODELS

A Thesis

Presented to

The Faculty of Cal Poly State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Craig Ethan Povey

February 2007

AUTHORIZATION FOR REPRODUCTION

OF MASTER'S THESIS

I reserve the reproduction rights of this thesis for a period of five years from the date of submission. I waive all reproduction rights after that time span has expired.

_____

Signature

_____

Date

APPROVAL PAGE

TITLE: Locating the Source of Topological Error in Reconstructed 3D Models

AUTHOR: Craig Povey

DATE SUBMITTED: February 2007

Dr. Zoë Wood _____
Advisor or Committee Chair

_____
Signature

Dr. Aaron Keen _____
Committee Member

_____
Signature

Dr. Lew Hitchner _____
Committee Member

_____
Signature

Abstract

Locating the Source of Topological Error
in Reconstructed 3D Models

Craig Povey

Although laser scanning (or range scanning) technology has offered great improvements to 3D model creation in recent years, it has also introduced some new concerns. Specifically, recent work shows that topological errors such as microscopic "handles" can significantly lower the overall quality of range-scanned models for down-stream applications (such as simplification and parameterization). Our goal in this project was twofold: to determine the specific source of this topological error in the range scanning process, and in doing so, to also develop a method to alleviate the error. We concentrated our investigation on the following two possibilities: (1) signal noise or calibration error in the laser scanner (resulting in bad data points) and (2) error during the model reconstruction phase. Although we were ultimately unable to remove any error by modifying the input data directly, we found that by modifying the reconstruction phase of the pipeline, we were able to effectively reduce the amount of topological noise in the resulting 3D model by up to 60%.

Table of Contents

Table of Figures

1 Introduction

One of the most active areas of research in 3D computer graphics is that of creating 3D model representations of physical objects in the real world for use in a vast array of different applications including: computer animation, video game development, rapid prototyping, medical research, archaeology, art history, and fashion design [9][23]. This model creation can be accomplished in a number of different ways; in this paper we focus on the realm of active or structured light sensing. In this method, a focused beam of light (a laser) is used to illuminate a line on the surface of the physical object, the reflected light from which can then be captured by a conventional video camera. Once captured, the center of each recorded scan line is computed, and the line of sight is traced through that pixel until it intersects the illumination beam at a point on the surface of the object. This process is called optical triangulation and produces a correspondence between the viewpoints of the laser and the camera. This correspondence then allows the object's physical shape to be captured by sweeping the laser beam slowly over the object's surface [6].

This entire process, depicted in Figure 1, is generally referred to as "range scanning" (or often simply "laser scanning"), and yields a "point cloud" of 3D data (called a range image) that can then be aligned and combined with other range images captured from different view angles and then processed in order to produce a 3D surface representing the physical object that was scanned. At this point, a surface representing the scanned object has been created, and may subsequently be transformed, edited, deformed, or rendered as desired.

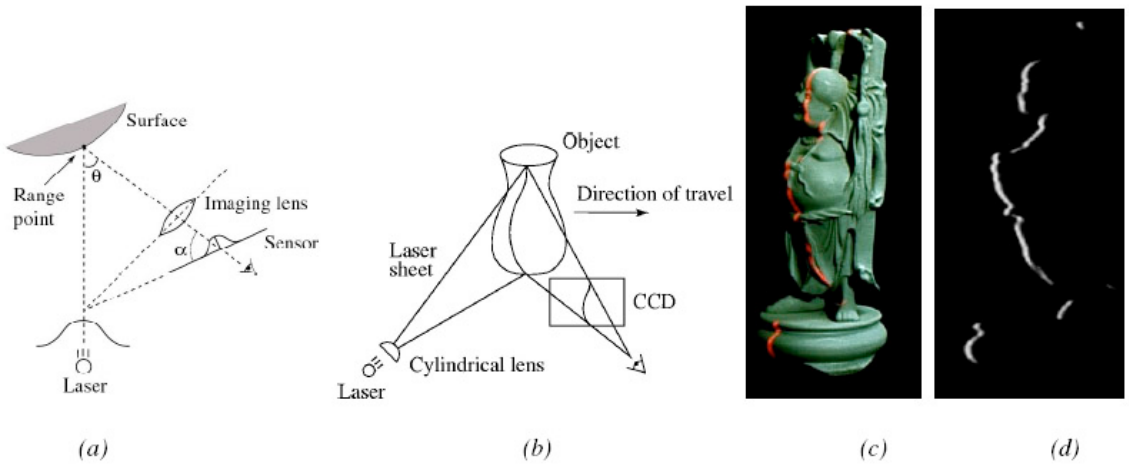

Figure 1: The structured light range scanning process. (a) 2D depiction of optical triangulation using a laser beam for illumination. (b) Extension of (a) to 3D. (c) Red laser line projected onto small Buddha statue. (d) Reflected light seen by the camera. (Figure taken from [6])

The final step in the process described above is generally referred to as surface reconstruction, and has itself been the topic of much recent research in 3D graphics [5][17][27]. One popular approach to the surface reconstruction stage merges each individual *range surface* (a tessellated version of a range image - see Figure 2) into a single surface embedded in a volumetric grid (containing elements commonly referred to as *voxels*), which is then sampled in order to derive a continuous implicit function representing the scanned surface. This implicit function is then used to extract a manifold (defined below) triangle mesh, called an isosurface, from the volumetric grid. In this paper, we utilize a well-known and commonly used reconstruction tool called VRIP [5]. Also note that we sometimes use the terms *range image* and *range surface* synonymously, although they actually represent different forms of a given data set (technically, a range image is purely a set of data points, whereas a range surface is a tessellated, continuous mesh representation of a range image).

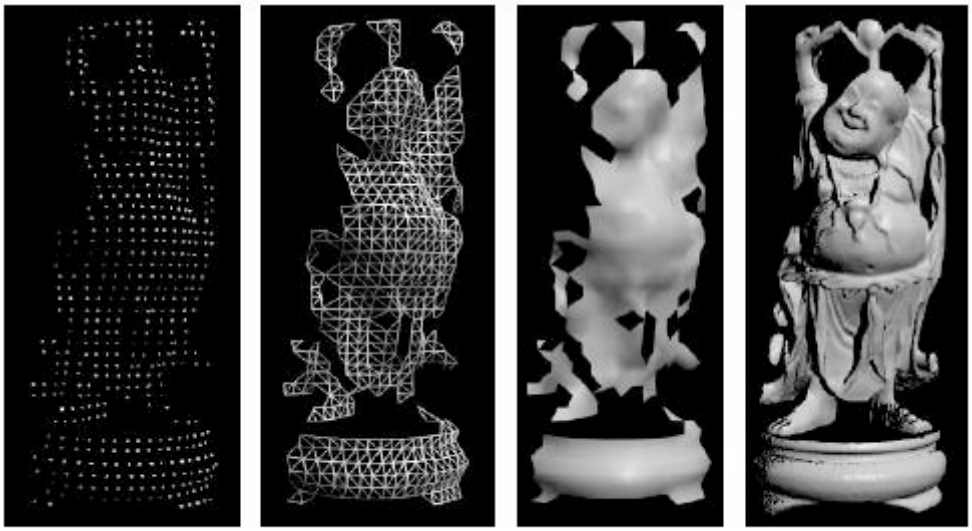

Figure 2: Stages of conversion from range image to range surface. (a) Range image "point cloud." (b) Mesh tessellation. (c) Shaded rendering of triangle mesh in (b). (d) High resolution range surface. (Figure taken from [6])

The terms manifold and manifoldness in the context of our project describe a surface or mesh and generally refer to its overall correctness or validity. Specifically, as defined by Jules Bloomenthal in [2], a closed manifold surface is one which is "embedded in [3D space] such that the infinitesimal neighborhood around any point on the surface is topologically equivalent ('locally diffeomorphic') to a disk." Intuitively, this means that the surface is "watertight" and contains no holes or dangling edges. For more information regarding the particulars of determining whether a given mesh is manifold, see Sections 3.3 and 4.3, where we describe our design and implementation of a two-part algorithm for determining an input mesh's manifoldness.

1.1 Problem Description and Motivation

Although range scanning technology has contributed greatly to 3D model creation, it has also introduced some concerns that did not previously exist. Specifically, recent work shows that topological errors such as microscopic “handles” can significantly lower the overall quality of range-scanned models, especially if the models are simplified after reconstruction [13][37]. In order to define what a topological handle is, we must first define the mathematical term genus. For our purposes in this paper, the genus of a closed surface is analogous to the number of “watertight holes” in the surface; for instance, a sphere has genus 0, while a torus has genus 1. We then define a handle (also known as a tunnel) as [11][38]. A handle can be imagined as a portion of the surface that would be topologically analogous to the handle on a coffee cup model (or the center of a donut model, etc.). Therefore, throughout this paper, we use the genus of a surface as an expression of the “handle count” of that surface.

One important distinction in terminology regarding our project is that between handles and holes. Handles in a mesh are part of the expected topology for a given model; for example, the handle of a coffee cup model. A hole, however, is instead a break or fissure in the mesh geometry itself, and is generally considered a defect in the mesh. A hole is defined as an area bounded by a continuous loop of *boundary edges* in the mesh, where a boundary edge is simply an edge with only one adjacent face. For this reason, these holes may be referred to synonymously as “boundaries.” In a more general sense, holes may be thought of as anything that would compromise a model’s water-tightness. For the remainder of this paper, we refer to these simply as holes, and the correcting of these

holes is referred to as the process of “hole-filling”, which is not to be confused with altering or changing the genus (or number of handles) in a mesh.

Motivation

In most cases, the handles found on the final reconstructed models are extremely small; in fact, the vast majority are completely imperceptible to any user viewing the model from a reasonable distance. Since these topological defects seem not to degrade the appearance of the final model, it may at first seem pointless to attempt to remove them. However, although topological handles may not directly degrade the quality of a 3D model, they do in fact degrade it indirectly by complicating subsequent geometry processing procedures, such as model simplification, smoothing, and parameterization [38]. In addition, topological artifacts hinder any type of mesh processing that require parameterization of the surface (such as texture mapping and remeshing) [14][20][30]. For instance, the geometry image parameterization technique described in [12] only works with low-genus models; models with a large number of handles (and therefore a high genus) were prohibitively difficult to parameterize, and thus need to have all extraneous handles removed before their parameterization technique could be applied effectively. Finally, some applications (such as the fitting of organ templates to medical MRI data) strictly require topologically correct models [18][31]. To illustrate the effects of topological handles on the quality of a reconstructed 3D model, consider the example of mesh simplification systems. Most traditional mesh simplification algorithms preserve the topology of the original mesh (including any extraneous handles); as a result, many

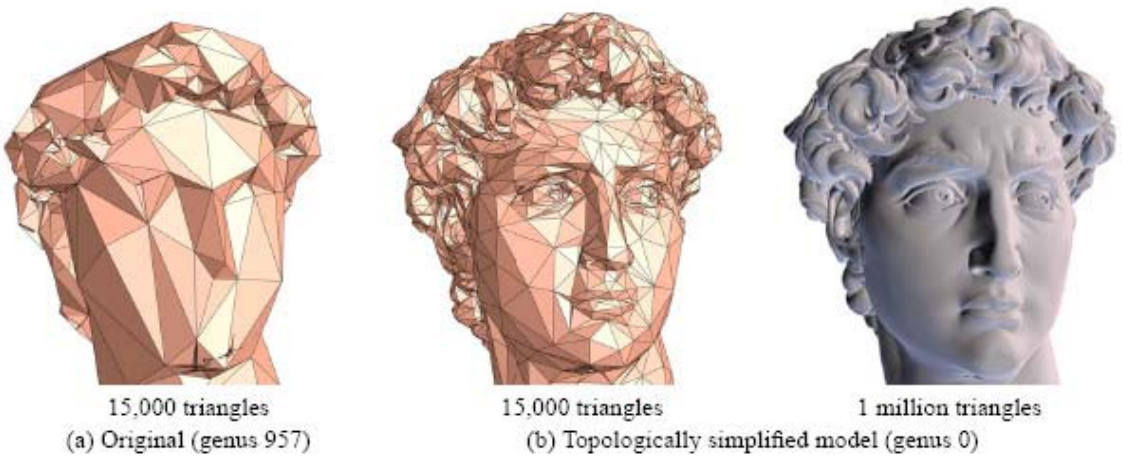

Figure 3: Comparison of simplified meshes with and without topological handles. (a) Poor detail in the facial area due to many triangles being wasted representing invisible topological handles. (b) High and low resolution versions of the model without any triangles wasted on extraneous topological handles. (Figure taken from [38])

triangles on the simplified mesh are wasted on preserving minuscule handles on the surface that should not have existed in the first place (see Figure 3). By removing these unwanted handles, the surface may be more accurately and efficiently simplified, deformed, animated, and rendered.

1.2 Contribution

As a result of the clear problem presented by topological handles, the issue has garnered a fair amount of attention in the 3D graphics research community (described in more detail in the Related Work section). However, while reasonable progress has been made toward removing the defects from the constructed model, much of this work offers little to no insight regarding the fundamental cause of those handles, much less how to alleviate or even prevent their creation. The primary goal of this project, therefore, is to determine the source of this particular type of topological error, with the intent of paving the way for

future research in the area of topological error removal, particularly as a pre-processing step.

Finding the source of the error

The first step toward finding the source of this error was to consider all stages of the basic 3D model generation pipeline and make some educated hypotheses regarding stages at which the error could likely be introduced (see Figure 4).

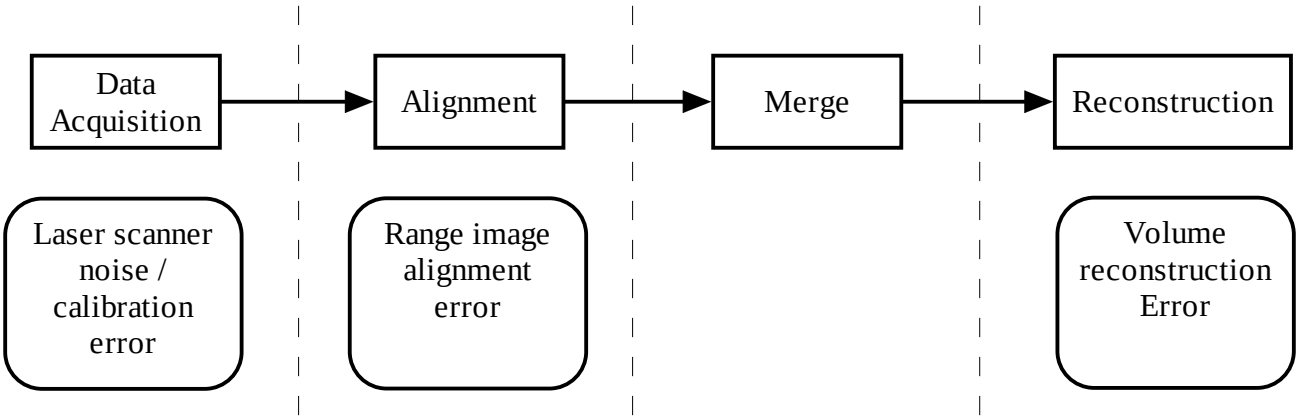

Figure 4: The basic 3D model pipeline and corresponding opportunities for error.

The first main stage of this pipeline is the *data acquisition* stage; this is the stage during which some device (such as a laser scanner) is used to generate some (usually large) 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. Any error introduced at this stage would be due chiefly to either signal noise or calibration error in the scanner device itself, subsequently resulting in the generation of "outlier" data points, and therefore extraneous handles in the final model.

The second basic stage in the pipeline is the *alignment* phase. In this stage, also referred to as “registration”, the range images generated in the first stage are inserted into a shared 3D space and then translated and rotated (either manually or automatically) so that they collectively represent the entire object being modeled. Various tools have been developed to aid the user in this process, most consisting of some combination of manual user input and automatic alignment assistance (such as Stanford’s Scanalyze tool [41]). Despite the amount of work put into this area, the alignment of range images continues to be a topic of much research interest [3][4], and given the reasonably “unsolved” nature of this stage, it is possible that even slight range image misalignment could generate topological defects later in the pipeline. For this project, we decided not to include this stage in our investigation; this was for no reason other than to limit the scope of the project in order to make it more manageable.

The third step in the pipeline is that of *merging* the data. This stage is very closely tied to the alignment phase, and basically consists of combining all the aligned range images into one cohesive data set representing the modeled object. Although there is certainly some potential for the introduction of topological error at this stage (perhaps due to a poorly written merge algorithm or other software defects), we decided that this was a reasonably unlikely possibility and thus chose not to concentrate our investigation on this particular step of the pipeline.

The fourth stage in the generation of a 3D model is called *reconstruction*, and is the final step in the pipeline. In general terms, this stage is responsible for transforming a raw or

basic representation of the model (such as a point cloud or isosurface equation) into a closed, (usually) manifold, 3D mesh that can be easily rendered or manipulated by a third party system. One common reconstruction tool widely used in the research community is the VRIP application developed at Stanford University [5]. VRIP uses the *marching cubes* algorithm [22] to volumetrically build a 3D triangle mesh from a set of aligned and merged range images. In addition to performing model reconstruction, VRIP also has a built-in hole-filling tool intended to guarantee that the system will always produce a water tight manifold model. However, we found that in the process of filling holes in a mesh, VRIP also created some bumpy, jagged extrusions in and around the hole-filled regions of the model. This gave rise to the suspicion that VRIP's hole-filling mechanism could possibly be playing a part in the creation of handles in reconstructed 3D models.

By examining two of the previous four stages, namely the acquisition and reconstruction phases, we were able to reduce the amount of erroneous topology by over 60%. In particular, we found that the surface reconstruction phase contributed strongly to the presence of excess topology; see Section 5 for more details.

2 Related Work

Our main intent in this project is to investigate two different stages in the 3D model creation process with a focus on identifying the primary contributor to the creation of erroneous topological handles in the output model. In doing so, our project has a strong basis on recent work in the areas of surface reconstruction, topology simplification, noise removal, and repair. We are also interested in and making use of recent work in the area of mesh hole-filling techniques (usually employed as part of the reconstruction process). Finally, our work also has a strong correlation to biomedical visualization, particularly in the area of MRI scanning and brain cortex modeling and visualization.

The pioneering work of Turk and Levoy [33] and Curless and Levoy [5] set the stage for our research by illuminating the problem of topological defects to the 3D graphics community. Their mesh “zippering” [33] and VRIP [5] systems (respectively) were widely accepted and utilized in the research community and as a result, the excess topological handles in the models they produced became more evident to the users of these systems. We chose to use the work of Curless and Levoy in our investigation; specifically, we chose VRIP and its built-in hole-filler as the subject of our analysis of possible topological error introduction in the reconstruction phase because of the wide-spread popularity of this software and the Stanford 3D repository.

More recent research has focused specifically on the problem of topological noise removal. Guskov and Wood [13] propose an innovative approach to removing small, extraneous handles from an extracted 3D mesh through a local wave front traversal

algorithm. This algorithm identifies handles by repeatedly growing "ϵ-balls" over the surface and removes them by making defined cuts in the mesh (i.e. "mesh surgery"). While this approach is simple, straightforward, and reasonably effective, it has a number of drawbacks; for instance, the algorithm's ability to detect a handle is constrained by the predefined size of the ball and therefore is unable to detect or remove long thin handles in some cases. Additionally, handle detection is very slow for large values of ϵ, and mesh surgery can create surface self-intersections in some cases as well. A final drawback is that this system requires a reconstructed mesh; that is, it is unable to work directly on the volume data or isosurface representation of the model. Wood, et al. [37][38][39] build on this work by developing a system that modifies the volume data itself, rather than the extracted mesh. Their algorithm makes an axis-aligned sweep through the volume to locate handles, compute their size, and remove them. Rather than growing ϵ-balls over the surface, this method finds handles by incrementally constructing and analyzing a Reeb graph [28]; handle sizes are computed by finding a short, non-separating cycle in the graph. This technique is therefore not constrained by ϵ in any way, and also is not susceptible to missing large, thin handles (as in the previous case). Additionally, handles are removed by directly modifying the volume rather than performing mesh surgery, which avoids the previous problem of creating self-intersections in the final mesh.

Bischoff and Kobbelt [1] take a similar volume-centric approach to removing erroneous handles. In their solution, an initially small set of voxels producing correct topology is gradually expanded by adding one voxel at a time until it fits the target isosurface. By only adding voxels that do not introduce a handle to the isosurface, this technique

guarantees that no handles are created in the reconstruction process, and therefore that the final surface will have the desired topology and genus. This differs from previous approaches in that it is primarily preventative rather than curative in nature with regard to handle creation. However, this method tends to result in more overall smoothing of geometry and loss of fine detail compared to a more targeted approach.

Recent work by Szymczak and Vanderhyde [32] works on volume data in a similar manner in order to extract an isosurface of a user-defined topological simplicity. In their method, a value is assigned to each voxel and then the voxels are ordered based on their distance from the target isosurface. The values of certain voxels are then altered, allowing only a limited number of topology changes during the extraction process. While this technique is fast, it doesn't allow much control over the manner in which handles are removed; for instance, a long, thin handle might be filled in rather than removed altogether, which in most cases is not desirable. A recent paper by Tao Ju [19] describes a new, innovative approach to repairing polygonal models by guaranteeing a closed surface that partitions space into disjoint internal and external volumes. This approach takes a "polygon soup" as input, constructs an intermediate volume grid, and then generates an output surface by dual contouring this grid. Although it appears promising, this work does not directly address our problem because we are interested in beginning with range images rather than a "polygon soup"; also, this technique sometimes produces topological defects in the reconstructed mesh. In these cases the author simply applies the topology simplification technique of Wood, et al. [37] to the intermediate reconstruction.

In our investigation of the hole-filling portion of the reconstruction process, we wanted to implement our own simplistic hole-filling algorithm so that we could more accurately assess the possibility that other existing hole-fillers (such as the one integrated into the VRIP tool) could be generating handles in the final mesh. In doing so, we are interested in learning from and building on any recent work in the realm of hole-filling 3D surfaces. Curless and Levoy [5] incorporate a hole-filling approach called "space carving" into their VRIP tool which interpolates points across non-sampled surfaces in concave regions of the model. These added surfaces serve to produce watertight models but may or may not (based on various input parameters) produce a truly manifold mesh. Davis et al. [8] propose a slightly different approach to hole-filling called volumetric diffusion intended to address situations in which holes are too geometrically complex to fill using traditional triangulation algorithms. This technique consists of converting a surface into a volumetric (voxel-based) representation with a given signed distance function where the zero set of this function defines the target isosurface. The function is initially defined only near observed regions of the surface, and then through alternating blurring and compositing steps, the function is "diffused" through the volume until its zero set covers all the existing holes. After the holes are filled, marching cubes is used to extract the final mesh. A final hole-filling approach proposed by Wang and Oliveira [35] is based on a *moving least squares* (MLS) algorithm and is intended to recover both geometry and shading information for the hole by using an interpolation procedure based on the context of the surrounding surface. MLS is a class of algorithms that addresses the generic problem of fitting smooth functions to a given set of scattered data. The basic idea of this technique is to first find existing holes (again, a hole is defined as any region surrounded by a loop

of inter-connected boundary edges where a boundary edge is any edge with only one adjacent face). For each hole, an MLS algorithm is used to repetitively resample and refit a surface to the hole until a reasonable fit is reached.

A final area of related work is in the domain of human brain cortex modeling. The reconstruction of the brain's cortical surface from *magnetic resonance* (MR) images is a very important goal in the biomedical and neuroscience fields, and therefore it is desirable to be able to reconstruct a topologically accurate 3D model of the brain from MRI data. The pioneering work done by Dale, et al. [7] was one of the first tools to offer a means of correcting visible topological error in a reconstructed cortical surface. However, their approach was simplistic, slow, and repetitive; the topological errors were identified simply by the user visually inspecting the inflated model and were corrected by manual hand-editing of this model. This process was very slow (approximately 30 minutes per brain hemisphere) and also only addressed larger, visible handles while ignoring smaller ones.

Fischl, et al. [10] proposed an improvement to the manual editing strategy in [7] wherein handles are detected and removed automatically by first inflating the surface to a sphere and then searching for any region in which overlapping triangles exist; this is based on the premise that overlapping triangles correspond to a handle on the un-inflated surface. The handles are then removed simply by removing the overlapping triangles from the mesh and re-tessellating the removed portion. Although an interesting approach to the problem, this method ends up being very slow, mostly due to the surface inflation step.

Also, removing all overlapping triangles from the mesh usually removes more of the surface than is actually necessary to obtain the desired topology; it can also lead to slight distortion of the surface regions surrounding the removed handles.

Shattuck and Leahy [31] also proposed a faster, automated alternative to that of [7] which provides a genus-zero model of the human cortex from MRI scans. Their method examines the connectivity of the white matter segmentation in order to find regions that contain incorrect topology. Rather than removing the handles directly from the mesh itself, this technique edits the underlying volume data in order to most efficiently correct the topology. While this solution is reasonably efficient and effective, it does have a number of drawbacks; for instance, it will always remove all handles in the model (that is, it always produces a surface with genus 0). This works well in the case of brain cortex modeling since a real brain is known to have a genus of 0, but greatly restricts the application of this algorithm to other domains and problems in which the target model may have “true” handles that should be preserved. In addition, this algorithm is only able to make cuts aligned with the Cartesian axes, which means that it will not always find the most natural or expected cut to remove a handle; Figure 5 shows an example case for which Shattuck and Leahy’s approach would fail to find the shortest, intuitive cut to break the handle.

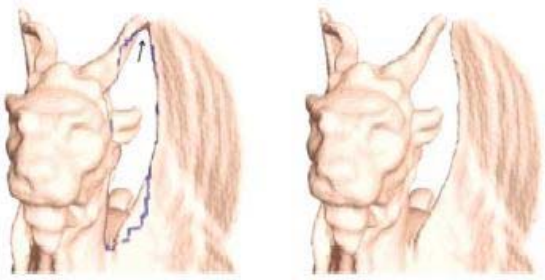

Figure 5: Cutting a topological handle. The left image shows an erroneous extension between the ear and wing that creates a topological handle. The right image shows the same model corrected by making a small cut around the extension. (Figure taken from [38])

Han, et al. [16] have extended the work of [31] in order to improve on some of the inherent weaknesses in their approach. This method is similar in that it is volumetric and graph-based (relies on a connection graph to detect handles), but differs in two main ways (both of which offer improvements on the methods of [31]). First, their approach is intrinsically three-dimensional and as a result, the "cuts" made to remove handles are not constrained to alignment along the Cartesian axes. This allows for more intelligent cut choices and therefore more efficient and realistic handle removal. Secondly, this approach allows the user to choose between using foreground or background filters during handle removal, which correlates to handles being either cut (removed completely) or filled in, respectively. The authors note that while the latter option may never be desirable in the context of cortical modeling, it may be a desired feature in other application domains.

3 Project Overview

As stated previously, the intent of this project is to conduct an analytical investigation of the 3D model creation pipeline with the intent of discovering how and where topological defects (specifically handles) are introduced into the final reconstructed model. Specifically, two main paths were taken: first, an analysis of the raw range data involved in the data acquisition stage, and second, an examination of hole-filling as a main component of the model reconstruction process. By exploring both of these paths, we hope to gain some valuable insight into the strengths and weaknesses that exist in the standard process for generating 3D models from scanned data sets.

3.1 Path 1: Raw Range Image Analysis

Our initial investigation involves taking a closer look into the raw range image data that serves as the input to the entire model creation process. The main basis for our decision to focus on this stage first was no more than simple intuition; it seemed most logical and likely that the topological handles observed were caused by erroneous points in the input data set generated by the laser scanning device. Based on this decision, the first problem that we had to solve was the simple question of *how to identify* erroneous points in a set of input range images.

To answer this question, we first made the hypothesis that if the range image data truly was responsible for creating topological handles, the error would likely occur in areas where “outlier” data points existed; that is, areas where the distance from one or more

points to their neighboring points is somewhat greater than the average neighbor-to-neighbor distance. Furthermore, we hypothesized that the most likely candidates for regions such as this were areas of overlap between adjacent range images.

Our reasoning behind this was twofold; first, these overlapping regions correspond to the edges of individual range images, which are generally known to be the least accurate portions of any given range image. This lower accuracy is due mainly to the fact that the angle between the laser scanner viewing vector and the normal vector of the physical object's surface is generally higher around the boundary of a given scan since the object's surface often curves away from the scanner in these areas. This causes the laser's beam to intersect the object at a grazing angle, and thus produces less accurate data points. This problem has been acknowledged in existing research, and was addressed in [5] and [33] by down-weighting certain vertices based on the dot product between the vertex normal and the viewing direction. However, we hypothesized that removing such outliers rather than simply down-weighting them might have an effect on the genus of the resulting model.

The second reason that we focused on the overlapping region between range images is because these regions are the most susceptible to inaccuracies in the alignment process. Consider, for example, two slightly overlapping finite 2D curves (depicted in Figure 6). If perfectly aligned, as shown in the top image of the figure, the overlapping portion of these curves will be completely coplanar, as if they were actually one congruous curve. However, if either curve is even slightly misaligned to the other, as in the bottom image,

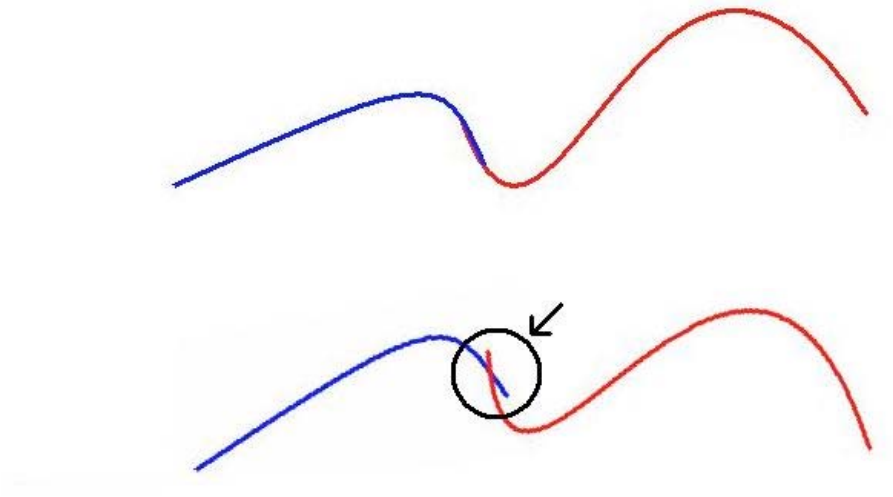

Figure 6: Error due to misalignment of range surfaces

part of the overlapping edge of each curve will be incorrectly offset from the surface of the adjacent curve. This offset region along range image overlaps could very likely have the same effect (i.e. creating surface handles in the final model) as the inaccurate data coming from the laser scanner, discussed previously. Thus, the areas of overlap between range images correspond not only to the areas with the highest margin of error from the laser scanning device itself, but also to the areas most affected by even slight misalignments between adjacent range images. With the above justification for concentrating our investigation on range image overlaps, the next step was determining a reasonable corrective process. We decided that a reasonable corrective approach would be to simply remove outlier data points in the border regions of range images that likely resulted from a poor laser-to-object scanning angle in the data acquisition process (described here as Path 1).

In theory, we would like to be able to examine the border region of each range image independently and remove its erroneous outlier points without any a priori knowledge of

other range images in the set. However, in practice, our only method of estimating whether a data point might be invalid is by comparing it not only to the closest points in its own range image, but also to those in any adjacent overlapping range images. Since we cannot gain any sense of overlapping regions or range image adjacency without first aligning all the range images, we actually perform our data point analysis and removal algorithm *after* the range image registration phase. However, we maintain that this analysis still operates on the raw input data, tests whether the data produced by the laser scanner is responsible for creating topological handles in the final model, and therefore occurs logically at the data acquisition stage in the pipeline.

The basic idea of our algorithm for identifying and removing erroneous data points from a set of input range images is that we want to simply find the distance between overlapping range image points and then remove those points whose distance is beyond a given threshold value (the details of this algorithm are described in Section 4.1). By removing these points, we expect that we will eliminate the surface geometry that eventually gets reconstructed into handles on the final model; therefore, our test will consist of the following steps: (1) run our algorithm to identify likely erroneous points, (2) remove the flagged points and write out a "filtered" version of the input data set, (3) input the filtered data set into the reconstruction phase (in place of the original data), and (4) compare the genus (handle count) of the model produced from our filtered data to that of the model produced from the original, unmodified data. An improved genus in this case demonstrates that erroneous points in the input data set do indeed contribute to producing topological handles.

One small clarification is that technically, removing bad data points is only solving one problem (eliminating the erroneous data) by creating another (introducing missing data). Therefore, removing this data doesn't actually solve our problem per se, but rather forwards it ahead to the hole-filling stage of the pipeline, where the holes that we created earlier by removing data points will be filled, effectively replacing the original erroneous data with more topologically accurate data generated by a hole-filling algorithm.

3.2 Path 2: Hole-Filling Analysis

In addition to looking at the raw range data as a possible source of topological error, we also speculated that the process of filling holes in the mesh during model reconstruction could contribute to topological noise in the final model. This hypothesis was based on our own experiences using the VRIP reconstruction tool; specifically, we observed that VRIP's hole-filler sometimes added noticeable extraneous topology to models when large gaps in the input data are present. We acknowledge that this phenomenon presents a slightly different problem than that of the miniscule handles we are interested in, but it does indicate that VRIP's hole-filler is completely capable of generating extraneous topology in a model which, if nothing else, at least provides a rationale for conducting an investigation into this particular section of the pipeline.

The following summarizes the functionality of VRIP's hole-filling mechanism, as described in [5]. Their hole-filler takes a volumetric approach, rather than a mesh-based approach; that is, it does not work directly on the reconstructed mesh but rather operates on the grid of voxels and then extracts surfaces from this volume to fill the holes. The

basis of this method is to classify all voxel points as having one of three states: (1) unseen, (2) empty, or (3) near the surface. Holes, then, are defined simply as boundaries between unseen and empty regions in the volume. Thus, the following describes VRIP's entire reconstruction process, including hole-filling:

1. Set all voxel points as "unseen"
2. Update voxels near to or containing the surface with continuous signed distance and weight values
3. Follow the lines of sight back from the observed surface and mark the corresponding voxels as "empty" (this step is known as "space carving")
4. Extract an isosurface made up of: (1) the zero-crossing of the signed distance function (i.e. the scanned surface) and (2) the surfaces corresponding to boundaries between unseen and empty voxel points (i.e. filled holes)

Thus, their approach creates the actual surface and fills the holes in the same surface extraction step (rather than creating the mesh first and then modifying it to fill the holes later).

In order to test the effect of the VRIP hole-filler on the generation of surface handles, we decided to write our own simplistic, mesh-based hole-filler. Our hole-filler is simplistic in that it does not aim to produce a realistic or aesthetically pleasing result; our goal here is rather to simply close open boundaries (i.e. holes) in the most straight-forward manner with the intent of reducing the introduction of excess topology. Our hole-filler is also mesh-based rather than volume-based; that is, unlike VRIP, we fill the holes by inserting geometry (i.e. points, edges, and faces) directly into the basis mesh rather than modifying

a voxel grid and then extracting a final, hole-filled mesh in one step. We chose this mesh-based approach for a few reasons: first, because our range data analyzer (see Section 3.1) was already mesh-based, it was much more straightforward to simply extend this implementation to a mesh-based hole-filler than to either convert it to a volumetric representation or to implement it volumetrically from scratch. A second reason that we chose to implement a mesh-based approach was that we wanted to be able to have more control over the manner in which the mesh was modified. In a volumetric methodology, only the voxel points are modified, and then the mesh is extracted from this as an isosurface; that is, there is something of a level of abstraction between the volume and the mesh it produces. Therefore, when hole-filling is done volumetrically, the actual geometry is modified in a less straight-forward way which creates a higher potential for adding more geometry and connecting disconnected regions compared to a mesh-based approach. In contrast, by operating directly on the mesh, we gained better control over each point, edge, and face that was inserted or removed from the mesh, and therefore reduced the chance of adding extra topology (such as that we observed in the VRIP output) during hole-filling.

The basic idea behind our hole-filler is to simply fill each hole in the mesh with a triangle fan; while this will not produce a visually realistic result in most cases, it is a simple solution that is not likely to add any extraneous topology. We first find holes in the mesh by finding all the *boundary edges*; that is, edges with exactly one adjacent face. Once all the boundary edges have been located, we simply begin with the first one, and trace along adjacent boundary edges until we arrive back at the beginning edge; when this occurs, we

have found a hole in the mesh. Once we have found all the holes in the mesh, we simply iteratively fill each one in turn. This is done by first placing a new vertex in the geometric center of the hole, and then generating a triangle fan around this center vertex to fill the hole. Despite the simplicity of this approach, there are some important considerations that had to be met in order for all possible cases to be handled correctly; the detailed description of our algorithm in Section 4.2 addresses these considerations.

Our investigation of VRIP's hole-filler is based on constructing two analogous models from the same set of range data whose only difference is in the hole-filling technique applied. We use our hole-filler to produce a "control" model and then compare the genus of this model to its counterpart generated using VRIP's volumetric hole-filler. Thus, if we observe a higher genus in the VRIP version, we may conclude that the volumetric hole-filler does indeed contribute to the problem of topological handles.

3.3 Manifoldness Test

As a validation of our solution, we wish to demonstrate that the mesh produced by our hole-filler is manifold in nature. As a corollary to the hole-filling mechanism described above, we also implemented a two-part test for evaluating a mesh's manifoldness which we applied both to our own hole-filled models as well as to those generated by VRIP. One of the main motivations for implementing this test was that we observed in multiple cases that models generated by VRIP occasionally had negative genera (i.e. plural of genus), a property generally corresponding to non-manifoldness. This could be the result of a faulty implementation of the Marching Cubes algorithm within VRIP, since the

original Marching Cubes algorithm [22] has been shown to produce topologically ambiguous (i.e. non-manifold) results in some cases [26]. The main problem with the original algorithm is based on ambiguities within the lookup table used to construct the isosurface; certain configurations have more than one possible tiling, and depending on the tiling chosen, the resulting surface may contain cracks [21]. Subsequent work has addressed this issue in various ways [24][25][34]; however, it is entirely possible that VRIP's Marching Cubes implementation uses a faulty, ambiguous lookup table, which would explain the negative genus results we observed in some cases.

The first part of our manifoldness test is a simple faces-per-edge evaluation; that is, we verify that every edge in the mesh has exactly 2 faces adjacent to it. Edges with exactly one adjacent face are classified as boundary edges, as described above, and are

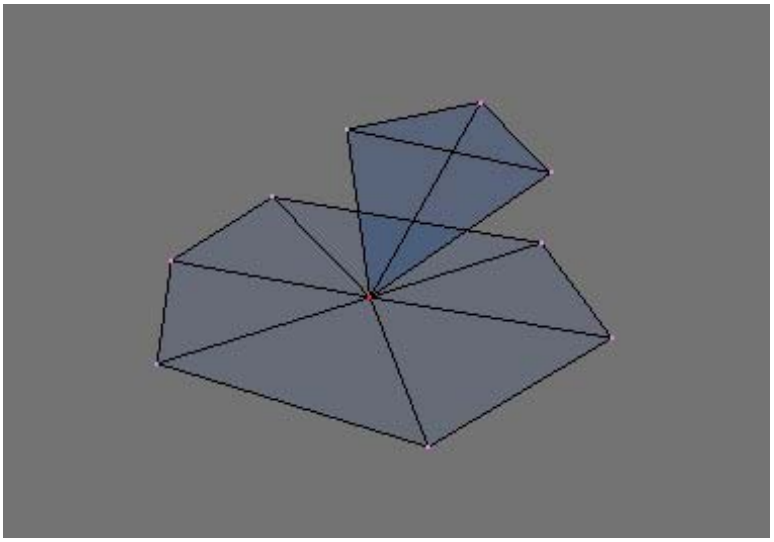

Figure 7: A non-manifold mesh that would incorrectly pass the faces-per-edge test. Assuming that the bottom portion of the mesh is actually interconnected with a larger mesh, every edge in the figure has exactly two faces adjacent to it. However, the red vertex would not pass the vertex disk test, and therefore is not manifold.

generally referred to as being *manifold-with-boundaries* [2]). Therefore, if any edge in a given mesh has either less than one or greater than two adjacent faces, the mesh is decisively non-manifold. However, this evaluation is technically incomplete, since it sometimes produces false-negatives; that is, there are cases in which a non-manifold mesh will pass this test (see Figure 7).

As a result of this shortcoming, we introduce our second algorithm for evaluating manifoldness: what we call a "disk/half-disk" test. This test is technically complete and sufficient for correctly identifying any non-manifold mesh. The basis for this evaluation stems from the definition of manifoldness proposed in [2], where every vertex in the mesh must be topologically equivalent to a disk (or a half-disk in the case of manifold-with-boundaries). Note that a closed (or water-tight) manifold will have only vertices which are equivalent to disks, while a manifold with boundary will have a combination of both types of vertices. Therefore, in accordance with this definition, our algorithm simply iterates over every vertex in the mesh and verifies that each is indeed equivalent to a disk

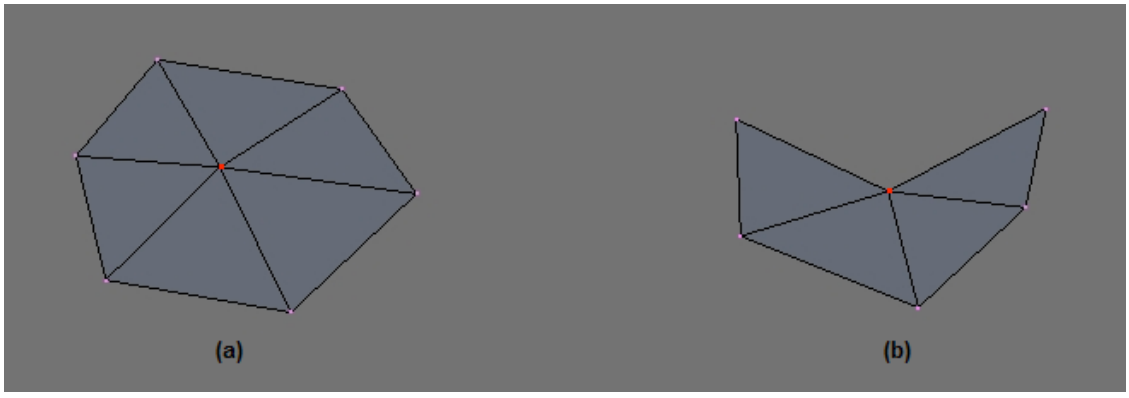

Figure 8: Basic disk and half-disk topologies, respectively. Every vertex in a manifold mesh must be topologically equivalent to the red "disk" vertex in (a), and similarly, each vertex in a manifold-with-boundaries mesh must be equivalent to either (a) or the "half-disk" in (b).

or a half-disk (as illustrated in Figure 8). Section 4.3 gives a more detailed explanation of our two manifoldness algorithms.

4 Implementation Details

In the previous section, we presented a general overview of our various methodologies for accomplishing our goal of locating the source of topological error in the 3D model creation pipeline. The following delves into these approaches in a more detailed and implementation-focused manner, with the intent of providing a more clear understanding of our overall algorithm, as well as a rationalization for the specific methods that we chose to accomplish our goal. Specifically, we describe our algorithms for analyzing raw range image data (Path 1 in Section 3) and for filling holes during model reconstruction (Path 2 in Section 3). We also illustrate two different algorithms that we implemented in order to verify the manifoldness of a mesh at any given time; these validation tests were extremely helpful during our investigations since our genus calculation assumes a manifold input mesh. They also helped provide validation to our own hole-filler.

4.1 Algorithm: Range Image Analysis

As described in Section 3.1, for our investigation of range image data, we want to analyze regions of overlap between range images and identify and remove any data points that are "too far" away from their closest neighboring range surface (since we hypothesize that these points may likely contribute to topological noise). Therefore, our algorithm may be broken into the following high-level steps:

1. Find all regions of overlap between each pair of range images
2. For each point *P* in an overlap region, find the closest point on the closest neighboring range surface to *P*

3. If the distance to this closest point is greater than a given threshold, flag *P* as a potential erroneous data point

A more detailed description of each of these steps is offered below.

Step 1: Finding the overlap using spatial partitioning

In order to determine the overlapping regions between neighboring range images, we propose a volumetric solution based on partitioning the data-enclosing volume into a regular grid of sub-elements or voxels. These spatial partitions are logically analogous to the voxels used in numerous existing volumetric reconstruction tools ([5], for example) but differ in their intent and use. We use our voxels not for extracting a zero-set isosurface, but rather as a way of grouping points from different range images that have reasonable proximal parity. Therefore, we are able to gain information regarding both whether or not a given region in the volume contains overlapping range images, as well as the distances between overlapping points in those regions that do.

The first step in the spatial partitioning process is to compute a global bounding box for all range images being analyzed for a given model. This is done incrementally by initializing the box to the x, y, and z bounds of the first range image and then comparing these bounds to those of each additional range image and extending the dimensions of the bounding box when necessary. Once the bounding box, or *enclosing volume*, is computed, we partition it off into a regular grid of rectangular voxels. The voxel grid size was determined by experience; in practice, a voxel size of 0.0025 (all voxels are cubes) was found to be reasonable. Once the grid is created, we simply “fill” each voxel with

any vertices or faces that fall within that partition's bounds in the volume. We process faces as well as vertices in this manner because in the distance calculation stage, we compare a vertex to the faces of its overlapping range surfaces. The voxel index corresponding to a given vertex or face is calculated in the following manner:

```
<VOXx,VOXy,VOXz> = ((<VERTx,VERTy,VERTz> - <MINx,MINy,MINz>) / bbDim)
```

Where `<VOXx,VOXy,VOXz>` is the 3D coordinate or index of the desired voxel, `<VERTx,VERTy,VERTz>` is the 3D vertex (or center of the 3D face) being indexed, `<MINx,MINy,MINz>` are the lower bounds of the global bounding box, and `bbDim` is the size of a voxel in the grid.

After all the vertices and faces have been partitioned for a given set of range images, we identify overlapping regions by simply locating any voxel that contains vertices and faces from at least two different range images. By grouping the range data this way, we not only gain a volumetric notion of overlap, but we also achieve a means of accelerating the point-to-surface distance calculation process described in the next section by limiting the comparisons to points and faces that fall within the same partition.

Step 2: Finding closest point-to-surface distance

Our aim in this step is to find, for each vertex in a given overlapping region, the shortest distance to the closest overlapping range surface. We restrict our search to the overlapping regions by only processing voxels that are known to contain vertices or faces of at least two different range surfaces (i.e. "overlap voxels"). Therefore, for each overlap voxel, we calculate the distance from each point P in range surface R_X to the closest point on each face F in range surface R_Y, where $X < Y$ and P and F are both in the same overlap

voxel. From this we can first find the shortest distance from P to each other R_Y, and then keep the shortest of these distances as the overall shortest distance from P to any of its overlapping range surfaces. Thus, the crux of this step is being able to calculate the distance from P to the closest point on F. This is a well-known problem within the realm of 3D graphics, and we follow the traditional solution, as outlined below.

In order to find the shortest distance between a point and a polygon, we first treat each polygon as a plane. This presents two main cases for consideration: the case in which the closest point on the face's plane actually falls within the bounds of the face itself, and that in which it does not. Our method for handling these cases can be described as follows:

for each vertex P:

for each face F:

1. Calculate the shortest distance from the vertex P to the plane defined by face F.
2. Determine whether this closest point in the plane falls within F using a basic point-in-polygon test, such as one of the many described in [15].

   2a. If it does, simply set the shortest distance from P to F, referred to as D_{PF}, equal to the distance found in Step 1.

   2b. If it does not, we must find the shortest distance from P to each of F's edges, and then save the shortest of these as D_{PF}.

3. If D_{PF}, as determined in Step 2, is shorter than the current shortest distance from P to any face, referred to as D_P, set $D_P = D_{PF}$. Loop back to Step 1 with next face F.

To calculate the point-to-plane distance as described in Step 1 above, we use the *Hessian Normal Form* for representing planes in space [36]. In this form, a 3D plane is represented by the equation: $\mathbf{n} \bullet \mathbf{x} = -p$, where $\mathbf{n}$ is the plane's normal vector, $\mathbf{x}$ is any point on the plane, and p is the *plane constant*. One benefit of the Hessian Normal Form is that once found, this plane constant can be used to easily calculate the distance from any point $\mathbf{x_0}$ to the plane using the equation: $D = \mathbf{n} \bullet \mathbf{x_0} + p$. This formula returns a signed distance from $\mathbf{x_0}$ to the plane where the sign of the result corresponds to which side of the plane $\mathbf{x_0}$ lies; however, in our application, the plane's orientation with respect to $\mathbf{x_0}$ is inconsequential and therefore we generally use the absolute value of D in our analysis.

Step 3: Flagging erroneous data points

At this point in our algorithm, we have calculated the shortest distance D_P to the closest overlapping range surface for each vertex P. Therefore, the real work has been done and all that remains is to simply loop through all the vertices and compare each D_P to the predefined acceptable distance threshold D_T. Any vertex for which $D_P > D_T$ is flagged as erroneous and may subsequently be highlighted in the user's display as such. By giving the user visual feedback regarding which data points have been flagged, we allow them the ability to dynamically change the value of D_T and re-analyze the data set until satisfied with the result. Once satisfied, the user may choose to write out the modified

range data with all flagged vertices removed (specifically, we write out modified range data as PLY files [40]). In our analysis, we use this filtered range data as input into the reconstruction stage of the model creation pipeline in order to determine whether removing these flagged vertices improves the genus of the generated model (for results see Section 5.1).

We chose to set the value of D_T for a model based on the average distance between neighboring vertices in a given range image from that model. Specifically, we randomly selected a range image from the set to be processed and calculated the average distance between all neighboring vertices in that range image. We then manually set D_T to be approximately twice this distance. We feel that it is reasonable to assume that the distance between any pair of neighboring valid points in a set of range images would not likely be greater than twice the average neighboring distance for that model. For future work, we recognize that our application could be modified to automatically calculate and suggest to the user a reasonable D_T value based on the input range data, making the filtering process both more efficient and user friendly.

4.2 Algorithm: Basic Hole-Filler

In Section 3.2, we gave an overview of our investigation into the effects of VRIP's hole-filler on the genus of a reconstructed 3D model. The basic idea was that we want to implement a simple and straightforward hole-filling mechanism that we can quantitatively compare to VRIP's hole-filler in order to gain some insight into the topological consequences of VRIP as a reconstruction tool. The only stipulation we

considered in designing our hole-filler was that it should ideally not introduce any topological error into the mesh being modified; this assurance basically acted as a "control sample" in our experiment and allowed us to gain a more accurate assessment of VRIP's side effects. A basic outline of our hole-filling algorithm is listed below, followed by a more detailed description of each step listed:

1. Find existing *hole-spanning* faces (this term is described below) in the mesh
2. Remove any hole-spanning faces
3. Find existing holes in the mesh
4. Fill any holes in the mesh

Step 1: Find hole-spanning faces

Intuitively, the first step in the hole-filling process would be to find the holes in the mesh. However, before we begin this step, it is desirable to ensure that our mesh is manifold-with-boundaries; that is, that every vertex in the mesh is topologically equivalent to either a disk or a half-disk (see Section 4.3.ii). This is desirable for two main reasons: first, because we want our final model to be manifold anyway, and second, because the process of filling in holes is greatly simplified if the initial mesh may be assumed manifold-with-boundaries. In most cases, the holes in the mesh will already conform to this standard; however, we found that many meshes contain cases that do not, such as that in which a face or group of faces span a hole, touching the hole's boundary only at vertices (we refer to this case as containing *hole-spanning* faces). Such a case violates the manifold-with-boundaries pre-condition because the vertices at which the spanning region intersects a hole boundary are topologically equivalent to neither a disk nor a half-disk (see Figure 9).

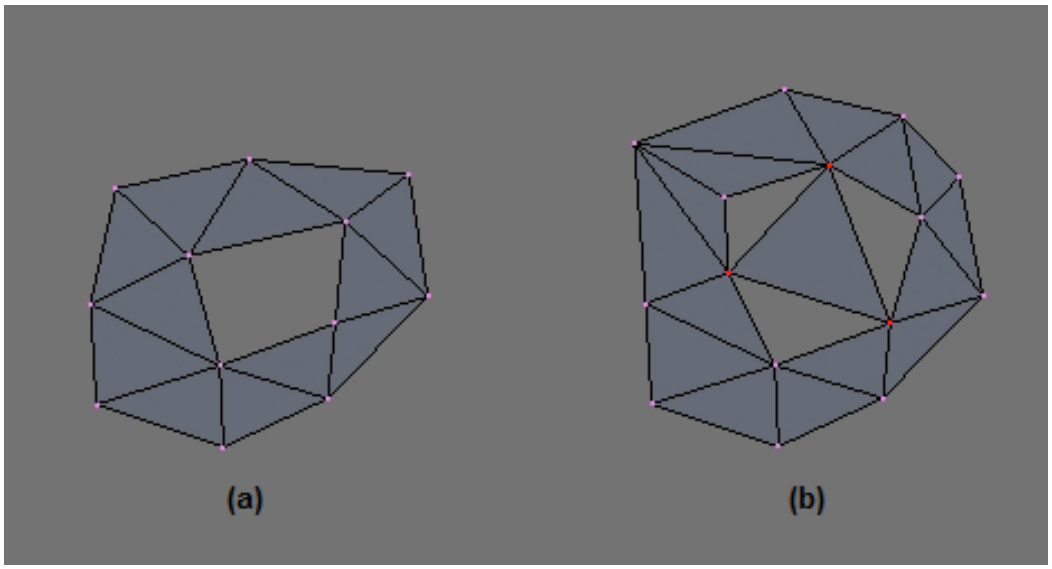

Figure 9: An example of an invalid *hole-spanning* face. The image in part (a) is a valid manifold-with-boundaries mesh. The mesh in part (b), however, is non-manifold since it contains invalid vertices (marked in red) that are neither disks nor half-disks. These vertices represent an invalid *hole-spanning* face.

In order to identify the hole-spanning faces in a mesh, we take the approach of positively identifying all faces that are known to be *non*-hole-spanning faces, implicitly identifying any unmarked faces as hole-spanning faces. The following steps describe our method:

1. Create an empty stack of faces to check and push an arbitrary face to the top.
2. Mark the top face on the stack and add any face sharing an edge with it to the stack.
3. Pop the top face from the stack and repeat Step 2.

We continue this process until the stack is empty; if at this point over half of the total faces in the mesh have been marked, we assume our initial face was not a hole-spanning face and that our current result is therefore valid. Note, that this approach is essentially equivalent to running Dijkstra's shortest path algorithm on the dual of the input mesh. If less than 50% of the faces have been marked, we repeat the process from Step 1, trying a different initial face. Theoretically, this process could be improved by doing some pre-

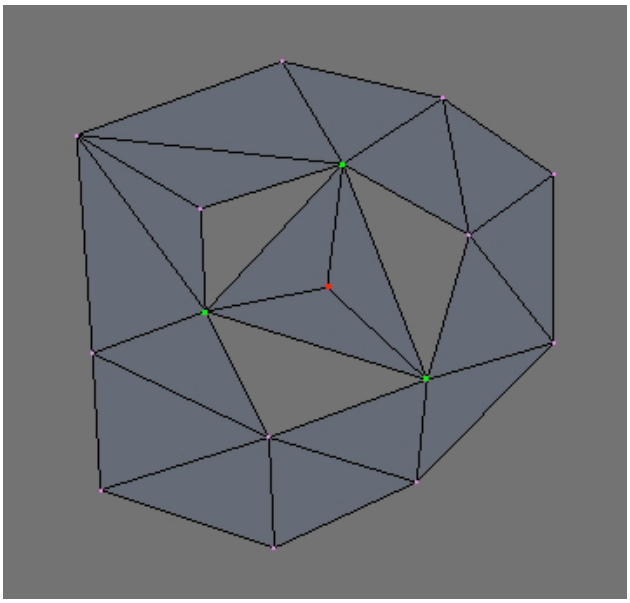

Figure 10: A mesh containing hole-spanning faces. Of the four vertices representing the hole-spanning faces shown here, only one must be removed (shown in red). The other three (shown in green) must remain, since they are also part of valid faces in the mesh.

processing in order to guarantee that the initial seed face is valid; however, in practice, the vast majority of faces in a mesh are valid seed possibilities and therefore our method of arbitrary selection is sufficient.

Step 2: Remove hole-spanning faces

At this point, the only unmarked faces in the mesh may be safely assumed to be hole-spanning faces since we know that all the remaining unmarked faces cannot be edge-adjacent to a marked, or valid, face. Therefore, with this information, removing the hole-spanning faces is reasonably straightforward; we simply iterate over our mesh face data structure and erase all unmarked faces. We also delete any edge that was part of a hole-spanning face, since we know that, by definition, hole-spanning faces cannot share an edge with a valid face. However, we may not simply erase any vertex that was part of a hole-spanning face, since these vertices may or may not be shared by a valid face (see

Figure 10). Therefore, in order to safely update our vertex data structure, we must first erase all hole-spanning faces and edges, and then iterate over all the vertices and remove only those that now have an empty *face list* (a simple mapping that we maintain for each vertex conveying which faces are currently touching the vertex).

Step 3: Find holes in mesh

By finding and removing all the hole-spanning faces from the mesh, we now have a manifold-with-boundaries; that is, that all holes in the mesh are topologically analogous to that depicted in Figure 9 (a). This greatly simplifies the process of identifying holes because once a boundary edge is identified, we need only recursively trace along neighboring boundary edges until we arrive back at the starting edge to find a hole. One problem with this strategy, however, is that it has the possibility of incorrectly identifying holes containing hole-spanning faces. This is because in these cases, as the algorithm recursively follows boundary edges (i.e. those with only one face), it will encounter a "branch" wherever a vertex of a hole-spanning face touches the true hole boundary. Thus, if the algorithm happens to follow an edge of the hole-spanning face instead of the true boundary, the resulting hole boundary will be incorrect (see Figures 9(b) and 10). Thus, by removing the hole-spanning faces, we can guarantee that there is only one possible boundary edge path surrounding a given hole, and may therefore assume that the space bounded by this path is a completely open hole in the mesh.

Based on the above guarantee, our algorithm for finding holes is fairly straightforward and may be outlined as follows:

1. Iterate over all edges and populate a queue of all *boundary edges* (edges with exactly one adjacent face).
2. Select the first boundary edge as the "current" one and remove it from the queue (this identifies the beginning of a new hole).
3. Find a neighboring edge to the current edge in the queue, make this new edge the current one, and then remove it from the queue.
4. Recursively repeat Step 3 until no neighbors exist in the queue and the current edge is a neighbor to the starting edge.
5. Create a new *hole* object, consisting of all the edges removed since the last new hole was identified.
6. Repeat process from Step 2.

Step 4: Fill holes in mesh

Upon completion of Step 3, we have a set of all the holes in the mesh; the only remaining step is to fill each hole by inserting a surface such that all boundary edges for a given hole are adjacent to the inserted surface. Specifically, we fill each hole using a *triangle fan* located at the center of the hole; the detailed steps of our triangle fan algorithm are as follows:

for each hole H:

1. Calculate the geometric center of *H* by adding the corresponding Cartesian coordinates of each boundary vertex and dividing by the total number of boundary vertices.
2. Add a new vertex at this center point

3. Add a new face (in our case, a triangle) for each boundary edge in *H* such that each new face consists of two adjacent boundary vertices and the center vertex, as depicted in Figure 11.

Although this technique makes no attempt to reconstruct the original geometry of the physical model, it does guarantee that the hole-filled regions on the reconstructed 3D model will be manifold and reduce the introduction of extraneous topological handles. Additionally, while we concede that it is still conceivable that our hole-filling approach could produce self-intersecting mesh regions in certain cases, this is not our concern in this project since we are instead focusing on the problem of erroneous topology.

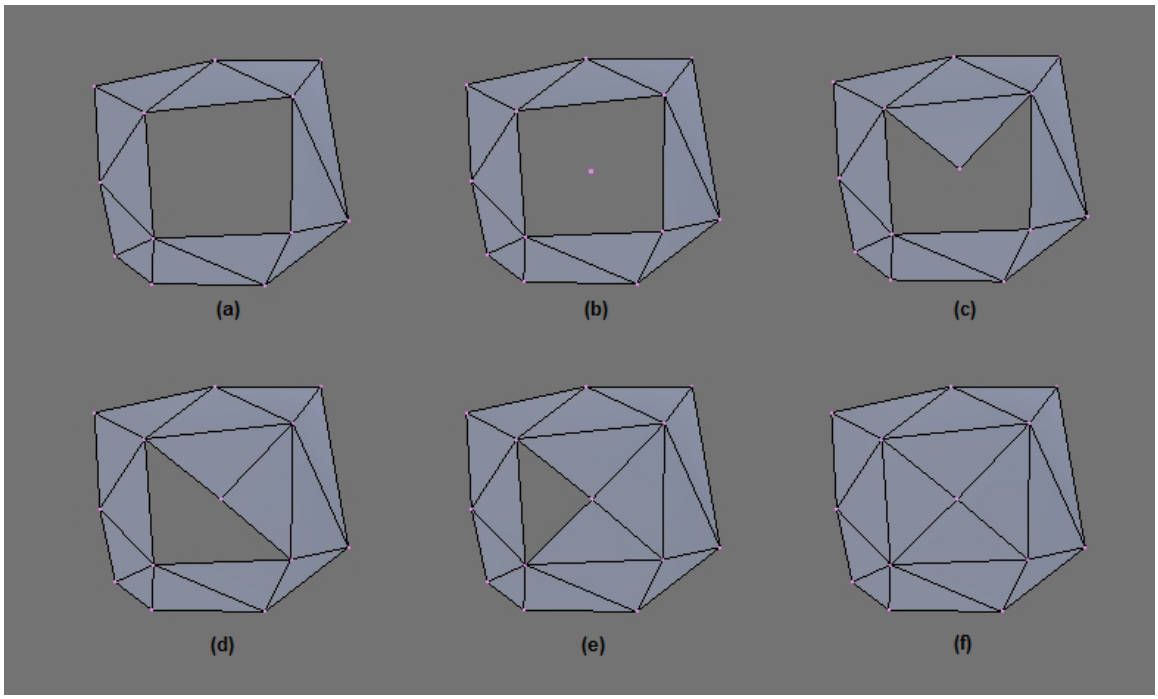

Figure 11: A step-by-step depiction of our hole-filling strategy. Part (a) shows a mesh hole before any hole-filling has taken place. Part (b) shows the addition of the new vertex at the center of the hole, and parts (c) – (f) show the iterative "filling" of the hole using a triangle fan.

4.3 Algorithm: Manifoldness Tests

In order to validate our hole-filling mechanism, we implemented two algorithms to test the manifoldness of a mesh; in doing so, we are able to show that if given a manifold-with-boundaries mesh as input, our hole-filling tool will always produce a fully manifold mesh as output. In addition to offering validation of our own work, these manifoldness tests also allow us to evaluate the manifoldness of models reconstructed using other third party applications, such as VRIP, and therefore give us some insight into the general quality of the models created by a given tool. The details of both parts of our manifoldness test are described below.

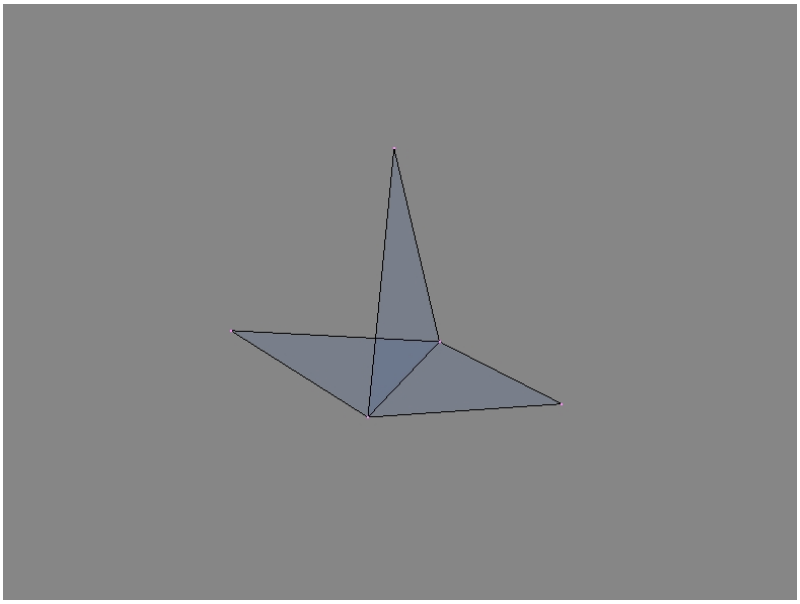

Figure 12: A “shark fin”. This topological defect occurs when three different mesh faces share a common edge.

4.3.i 2-face-per-edge test

Our initial method for testing a mesh’s manifoldness is derived from the stipulation that every edge in a manifold mesh must be adjacent to exactly two faces (edges with exactly one adjacent face are permissible for manifold-with-boundary meshes). The most

common violation of this requirement is manifested in the defect known as a “shark fin”; this topological error occurs when three different faces in a mesh share a single edge, as illustrated by Figure 12. Thus, we can detect a variety of manifoldness violations such as holes and shark fins by simply checking the number of faces that are attached to each edge in a given mesh. Our implementation of this test is fairly trivial; most of the work is done during the initial loading of a mesh. During this operation, we simply maintain a list of faces for each edge object and for each face *F* in the input file, we determine all the edges bordering *F* and add *F* to the face list for each border edge. After this initial cataloging is complete, we simply iterate over all the edges in our mesh and increment a counter each time we find an edge with a face count not equal to 2 (in practice, we display to the user the number of edges that match a given face count).

4.3.ii Vertex disk and half-disk tests

Although it is true that each edge in a manifold mesh must have either one or two adjacent faces, this requirement is not alone sufficient for defining manifoldness; that is, it is possible for a mesh to meet this condition and still be non-manifold (see Figures 7 and 10). Therefore, we also implemented a second algorithm providing a more complete evaluation of a mesh’s manifoldness. This algorithm is based on the concept presented in [2] that each vertex (or rather the set of faces adjacent to each vertex) in a manifold or manifold-with-boundaries mesh must be topologically equivalent to a disk or a half-disk, respectively (Figure 8 illustrates the concepts of a vertex disk and half-disk). This test catches all violations of manifoldness (including those depicted in Figures 7, 10 and 12) and therefore offers a complete means of mesh validation regarding manifoldness (we

will henceforth refer to the disk and half-disk tests collectively as simply “the vertex disk test” since their respective implementations are highly integrated).

The high-level steps of our vertex disk algorithm are described as follows:

for each vertex V:

1. Add all faces touching *V* to a *facesRemaining* list.
2. Randomly select a starting face *F* from *facesRemaining*
3. Find a face *F’* such that *F’* is neighboring *F* and within the *facesRemaining* list.
4. Set *F = F’* and remove the previous *F* from the list.
5. Repeat steps 3 and 4 until either no neighboring face exists in the list, or the original face is found (the original face is identified by maintaining an index for each *F* into the global list of all faces in the mesh).
    - If no neighbor exists in *facesRemaining*, set *F* to be any boundary face (i.e. face containing a boundary edge) in the list and resume at step 3; we call this a *second pass*, and is necessary for identifying possible half-disks.
        - If the second pass also fails (that is, if either no neighbor is found in the list or the original face is found but the list is not empty), *V* is definitively considered an invalid vertex.
        - If the second pass succeeds (that is, if the original face is found and the *facesRemaining* list is empty), *V* is definitively considered a valid *half-disk* vertex.

- If the original face is found, check if the *facesRemaining* list is empty.
    - If the list is empty, *V* is definitively considered a valid *disk* vertex.
    - Else, *V* is definitively considered an invalid, non-disk vertex.

If, for any reason, a vertex is identified as *invalid* in the above algorithm (there are two main possibilities for such a classification), the mesh may be definitively declared non-manifold, since neither a manifold, nor a manifold-with-boundaries mesh may contain an invalid vertex.

5 Results

In this section, we validate our work by presenting experimental data obtained during the testing of our project using various input range data sets. We present results from our analysis of both the raw input range data as well as the hole-filling stage of model reconstruction. The range image data used in our experimental tests was obtained from Stanford's 3D Scanning Repository website [42] and includes the following models: a Buddha statuette, a sculpture of a Chinese dragon, and an armadillo action figure (see Figure 13). Each of these models was scanned using a Cyberware 3030 MS laser scanner device, and consists of approximately 60-70 range scans/images.

(a) (b) (c)

Figure 13: Various models used in our experimental trials. The models shown in parts (a), (b), and (c) are the 3D reconstructions of range data scanned from a Buddha statuette, a sculpture of a Chinese dragon, and an armadillo action figurine, respectively.

5.1 Range Image Data Analysis Results

For our analysis into the topological effects of possibly erroneous or noisy raw range data (as generated by the laser scanning hardware), we focused on our algorithm for filtering data points from the original range image files (see Section 4.1). Our investigation was structured in such a way that the only variable in the pipeline was the input range data. Specifically, we started with the original range data for a given model, and then ran this data through our range data analysis algorithm from Section 4.1 to produce a "filtered" subset of the original data. We then ran both data sets (i.e. control and variable) through the rest of the pipeline in an identical manner, producing two reconstructed models between which the only difference was the range data set used as input. By comparing the genus of the resulting models, we can therefore demonstrate what effect, if any, removing outlying range data points has on the topology of the final model. To obtain more data, we performed the above steps twice for each original range data set; once using VRIP's hole-filling mechanism, and again using our own. The results of this investigation are displayed in Figure 14.

	Range Data	Voxel Dim.	# of Vertices	# of Edges	# of Faces	Genus
Dragon						
VRIP Hole-Filling	Original	0.00025	1769202	5307870	3538580	**45**
	Filtered (0.0017)	0.00025	1776208	5329056	3552704	**73**
Our Own Hole-Filling	Original	0.00025	1702583	5107848	3405232	**17**
	Filtered (0.0017)	0.00025	1700769	5102445	3401630	**24**
Buddha						
VRIP Hole-Filling	Original	0.00023	1564942	4695222	3130148	**67**
	Filtered (0.001)	0.00023	1587575	4763319	3175546	**100**
Our Own Hole-Filling	Original	0.00023	1436039	4308279	2872186	**28**
	Filtered (0.001)	0.00023	1415820	4247586	2831724	**22**

Figure 14: Range Image Analysis Test Results

The modified models described here were filtered using a data point distance threshold of approximately two times the average nearest neighbor distance for the range set; specifically, a threshold of 0.0017 was used for the dragon scans, and a threshold of 0.001 was used for the Buddha. The genus results were somewhat surprising in that the effect of range data filtering was noticeably different depending on the hole-filling mechanism used. For both models, when the VRIP hole-filler was used on both the filtered and non-filtered data, the model resulting from the filtered data had a substantially *higher* (i.e. worse) genus than that from the original data. Specifically, we saw the model's genus increase by 28 in the case of the dragon (a 62% increase) and by 37 in the case of the Buddha (a 55% increase). When we used our own hole-filling technique, the filtered model's genus changed by 7 (or 41%) in the case of the dragon, and of 6 (or 21%) for the Buddha, thus, removing outlier data points from the range images using our own hole-filler did not effect the final genus to the same extent as it did in the VRIP case.

These results reveal a great deal regarding the source of topological error in these models. First of all, they show that outlier points in range images are *not* a significant contributor to topological noise in reconstructed models, as we initially hypothesized. Although this in itself does not show us where the problem *is*, it does at least show us where it is *not*, which is still beneficial to our investigation. Additionally, our results actually support our supplementary hypothesis that VRIP's hole-filler contributes to topological error; our rationale for this conclusion is as follows. The process of filtering range images effectively either creates or enlarges holes or gaps in the original range images. This

therefore directly leads to an increased amount of hole-filling during reconstruction. If we assume (for the sake of argument) that VRIP's hole-filler does *not* create topological handles, then this increased activity would have no adverse effect on the genus of the resulting model; however, if we assume that it *does*, in fact, contribute to the error in the model, then we would expect an increase in hole-filler utilization to cause a corresponding increase in topological handles, and therefore, an increased genus in the resulting model. Thus, since our experimental results validate the latter, we may surmise that VRIP's hole-filling technique is indeed a significant source of topological error in reconstructed 3D models.

Although the difference in genus between the filtered and non-filtered models was much smaller when our own hole-filler was used (compared to the cases using VRIP), these cases are noteworthy for a different reason; namely, the fact that the genus *increased* slightly in the case of the dragon, but *decreased* slightly in that of the Buddha. One possible explanation for the Buddha's decreased genus is that the filtering process actually removed data points that would have otherwise caused topological handles apart from the hole-filling process (the fact that the models created using our hole-filler still have topological error shows that there is at least one *other* source, apart from VRIP's hole-filler). Thus, by re-creating these parts of the model using our hole-filler, we were able to eliminate the error in these areas. That said, the most likely explanation for the dragon's increased genus is simply that whatever else (apart from VRIP's hole-filler) is also responsible for creating topological error in these models was exacerbated by the process of filtering the range images. For example, if we assume for the moment that

slight misalignments in the range image alignment stage do in fact lead to topological error in some cases, it makes sense that removing parts of the data (especially data around the boundary regions) would worsen the existing misalignment and therefore increase the amount of error generated as a result.

5.2 Basic Hole-Filler Results

Our experiment for testing the topological effects of VRIP's hole-filling mechanism (representative of the reconstruction stage in the 3D model creation pipeline) was designed in the following manner. For each model being tested, we obtained the set of range images corresponding to the model and reconstructed two different versions of each; once using VRIP's internal hole-filler and a second time without it. We then applied our project's hole-filling algorithm (described in Section 4.2) to the version of each model that had not been hole-filled by VRIP. Finally, we tested the manifoldness of both versions of each model and calculated their respective genera. The results of this process are displayed in Figure 15.

	Voxel Dimension	# of Vertices	# of Edges	# of Faces	Genus
Dragon					
VRIP	0.00025	1,769,202	5,307,870	3,538,580	**45**
Our Project	0.00025	1,702,583	5,107,848	3,405,232	**17**
Armadillo					
VRIP	0.00023	1,477,794	4,433,382	2,955,588	**1**
Our Project	0.00023	1,477,186	4,431,558	2,954,372	**1**
Buddha					
VRIP	0.00023	1,564,942	4,695,222	3,130,148	**67**
Our Project	0.00023	1,436,039	4,308,279	2,872,186	**28**

Figure 15: Hole-filling Test Results

As the table shows, our basic hole-filling algorithm was able to effectively decrease the genus of the final reconstruction by over 60% for both the dragon and Buddha models. Interestingly, the armadillo model had negligible topological noise in both cases; we attribute this phenomenon to the fact that the armadillo figurine itself has fairly convex topology in comparison to the other two models; that is, there are very few regions of the model obscured from the scanner. This helps validate the logical hypothesis that topological noise is most common in the areas of a model where range data is missing; that is, in areas that are difficult to sample using a laser scanner. Our results also validate our initial hypothesis that the reconstruction phase of model creation is responsible for at least a substantial portion of the topological noise present in many complex models created from range data; more specifically, our data shows that VRIP's hole-filling mechanism is responsible for introducing a significant amount of genus-increasing topological error.

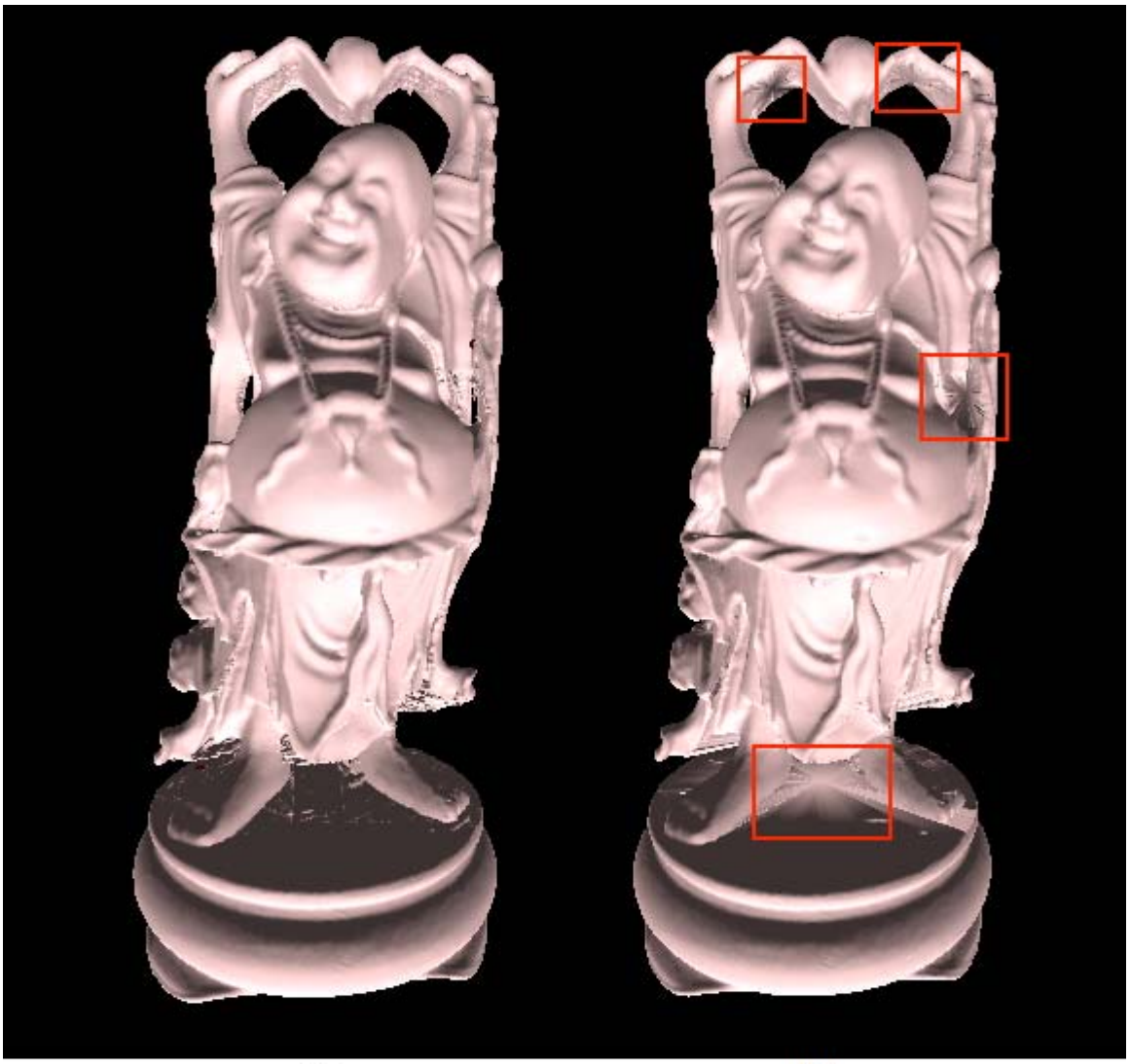

Figure 16: Hole-filled Buddha models. Part (a) was hole-filled by VRIP, and part (b) by our own basic hole-filling algorithm. The red boxes point out areas where the visual results of our hole-filler can be most easily observed.

Figure 16 compares the appearance of the two hole-filled Buddha models. The visual results of our hole-filler can be observed in various places in part (b); the most obvious areas are enclosed by red boxes. As the figure shows, our hole-filler creates noticeably incorrect “starburst”-shaped patterns in the regions where larger holes were filled. This is a direct result of our simplistic approach in hole-filling; the starburst pattern is simply the visual manifestation of the “triangle fan” we create to fill each hole. Since the triangle

fans are created by creating multiple (often long and narrow) triangles to radiate out to the boundary of each hole from a newly created vertex at the center, it makes sense that the resulting region in the 3D model would look similar to that observed in Figure 16 part (b). Figure 17 shows the same comparison for the dragon model (the armadillo model contained no clear hole-filling visualizations). Note that these visual side-effects would not be acceptable for most computer graphics applications, and we do not assert that we have produced geometrically accurate models. However, our boundary filling method was designed with the intention of assisting our investigation into the source of extraneous topology and therefore a final surface reconstruction algorithm would need to be modified to correct these side-effects.

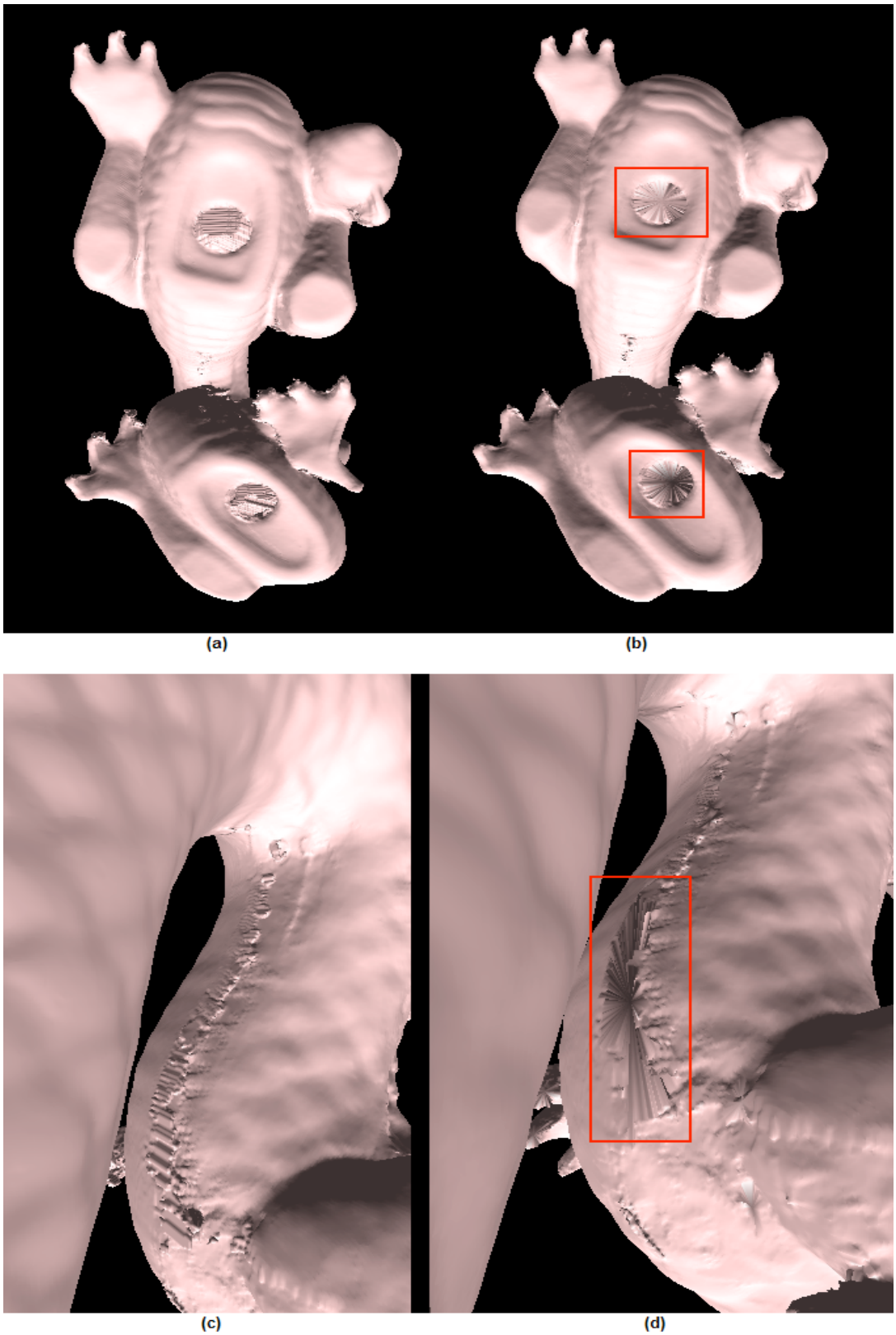

Figure 17: Hole-filled dragon models. Parts (a) and (c) were hole-filled by VRIP, and parts (b) and (d) by our own basic hole-filling algorithm.

6 Conclusion

The ultimate goals of this project were two-fold; first, to identify the source of erroneous topological handles in reconstructed 3D models, and second, to develop a method for alleviating some or all of the erroneous topology. Although we were not able to identify the source of **all** the error in our experimental data, we feel that we have nonetheless met our goals for this project; our analysis of the reconstruction phase demonstrates that VRIP's hole-filler is the cause of over 60% of the topological handles in the experimental data. In addition, our experiments with filtering range image data show that when using VRIP's hole-filler, the filtered range data results in a genus increase of approximately 50-150% over the non-filtered model; however, when using our own hole-filler, the genus remained relatively similar. We offer the following explanation for these results: since the filtered range data has "gaps" in the data (which correlate to new or enlarged "holes" in the mesh defined by that data), the filtered models will require more "hole-filling" than their non-filtered counterparts. Therefore, if the VRIP hole-filler does indeed cause topological handles, it makes sense that increasing the amount of processing done by this mechanism would only increase the amount of error in the resulting model; the fact that the same phenomenon does *not* occur when using our own hole-filler only reinforces this theory.

We base our assertion that VRIP's hole-filler is a major source of topological error on our experimental results as well as our prior knowledge and experience regarding the approach taken by VRIP's hole-filler. In general, VRIP first marks the entire volume as "unseen", then uses a technique called "space carving" to mark voxels containing the

surface as “near the surface” and any space between the scanner and the surface as “empty”. Holes are then identified as boundaries between “empty” and “unseen” regions in the volume, which are then filled by modifying the enclosing voxels. However, in many cases the space carving process relies on the use of “backdrop” range scans; that is, range data that exists “behind” the model and outside of the voxel grid for the sole purpose of improving the space carving results. Without these backdrop scans, certain portions of the volume are never “carved” or marked as empty, which in turn produces undesirable hole-filling results (see parts (a) and (c) of Figure 18). As Figure 18 shows, in some cases VRIP’s space carving/hole-filling process is inexact at best and may add more geometry to the surface depending on the configuration of the empty-unseen boundary regions. The “fill surfaces” generated by VRIP to seal these boundary regions will often have an undefined shape, extent, and topology, unlike the relatively planar “caps” that our hole-filler creates to fill holes. This ambiguity regarding the extent of the empty-unseen boundary regions is very likely the root cause of the extraneous topology we observed as a result of VRIP’s hole-filler.

In addition to showing that VRIP’s hole-filler is a major source of topological error, we also demonstrated the equally important result that outlier points in the range image data are *not* a significant source of topological noise in the corresponding reconstructed models. Finally, in regard to future research efforts, our work also shows that while VRIP’s hole-filler plays a role in creating topological noise, it is not the only culprit; since the models reconstructed without using VRIP’s hole-filler still have higher-than-

normal genus, we may deduce that there is at least one additional unidentified source of error.

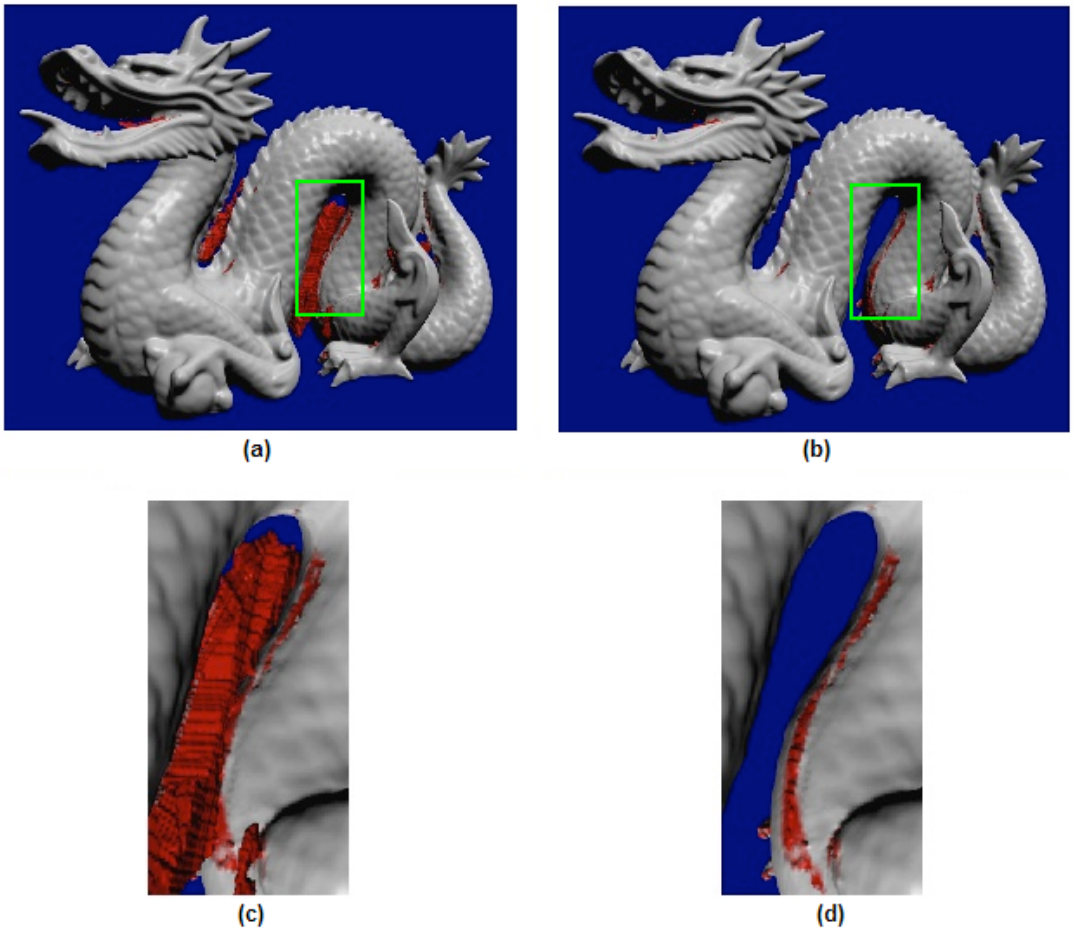

Figure 18: Effects of backdrop scans on VRIP's hole-filler. Parts (a) and (c) were hole-filled by VRIP without the use of backdrop scans for space carving, whereas parts (b) and (d) show the results of the same process using backdrop scans. In all cases, the red portions of the surface correspond to geometry added by VRIP's hole-filling mechanism. (Figure taken from [5])

7 Future Work

The work done for this project can be extended in two main ways. The first option for future work is to explore the range image alignment stage in the 3D model generation pipeline with respect to its possible role in creating topological error. In this project, we have focused investigations into the data acquisition and model reconstruction stages but decided, simply for the purpose of limiting the project scope, to not include the alignment phase. However, we acknowledge that the process of range image alignment is nonetheless a likely source of topological error in reconstructed models. One reason for this is simply because it is one of the most difficult parts of model creation, and still leaves a great deal of room for improvement. Most common alignment techniques used today are based on some variation of the Iterative Closest Points algorithm (ICP) [29], which uses rigid-body transforms to align adjacent range images over a series of incremental steps. However, though this technique is common, recent work has shown that it might not be the best or most logical way to align range image data. According to the work of Brown and Rusinkiewicz, warps in laser scanned range data are often *non-rigid* in nature; they point out that even small calibration errors in the laser scanner device can result in a low-frequency, non-rigid warp in the acquired data, which cannot accurately be fitted by a rigid-body alignment algorithm like ICP [3]. These authors present a new alignment method using *thin-plate splines* to perform non-rigid alignment of range images. Thin-plate splines are a class of non-rigid spline mapping functions that possess a number of desirable qualities for range image alignment. Therefore, one avenue of research is to align a set of range images using two different methods (such as traditional ICP and the non-rigid technique of Brown and Rusinkiewicz) and then

compare the genus results of the models reconstructed from each of these data sets. This would be very helpful since although our work demonstrates that VRIP's hole-filling mechanism is responsible for a large portion of the topological noise in many 3D models, it does not account for all the erroneous topology present in these models; thus, a close analysis of the alignment stage could provide key insight into the source of the remaining error. In addition, it is unclear exactly what effect range data filtering would have on alignment, so a combined investigation of both filtering and alignment may yield interesting and helpful results.

A second avenue for future work related to this project is to improve the hole-filling tool that we developed. Although we succeeded in creating a simple, robust algorithm for filling holes in a mesh and guaranteeing the manifoldness of the resulting mesh, we made admittedly little effort to ensure that our algorithm fills mesh holes in an aesthetically pleasing and accurate manner. That is, although our hole-filled meshes are correct topologically-speaking, they are not necessarily correct visually. This is because we have no built-in facility for recreating missing detail while filling in holes. Therefore, our hole-filling mechanism could be optimized to generate a reasonable approximation of the holes' missing geometry, while still guaranteeing manifoldness. Note that specifically, in Figure 16, our simple hole-filler has closed one of the handles inherent in the Buddha model (in the underarm region) which is incorrect. A hole-filling algorithm which more accurately follows the geometry of the input model should correct this problem.

8 References

[1] Bischoff, S. and Kobbelt, L. P. 2002. Isosurface Reconstruction with Topology Control. In *Proceedings of the 10th Pacific Conference on Computer Graphics and Applications* (October 09 - 11, 2002). IEEE Computer Society, Washington, DC, 246.

[2] Bloomenthal, J., editor, 1997. *Introduction to Implicit Surfaces*. Morgan Kaufmann, San Francisco, California.

[3] Brown, B. J. and Rusinkiewicz, S. 2004. Non-Rigid Range-Scan Alignment Using Thin-Plate Splines. In *Proceedings of the 3D Data Processing, Visualization, and Transmission, 2nd international Symposium on (3dpvt'04) - Volume 00* (September 06 - 09, 2004). IEEE Computer Society, Washington, DC, 759-765.

[4] Chen, C. and Stamos, I. 2005. Semi-automatic range to range registration: A feature-based method. In *The 5th International Conference on 3-D Digital Imaging and Modeling*, 254-261, Ottawa, June 2005.

[5] Curless, B. and Levoy, M. 1996. A Volumetric Method for Building Complex Models from Range Images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and interactive Techniques* SIGGRAPH '96. ACM Press, New York, NY, 303-312.

[6] Curless, B. 1999. From Range Scans to 3D Models. *SIGGRAPH Comput. Graph.* 33, 4 (Nov. 1999), 38-41.

[7] Dale, A., Fischl, B., and Sereno, M. 1999. Cortical Surface-Based Analysis I: Segmentation and Surface Reconstruction. *NeuroImage* 9:179–194.

[8] Davis, J., Marschner, S., Garr, M., and Levoy, M. 2002. Filling Holes in Complex Surfaces Using Volumetric Diffusion. In *Proceedings of the First International Symposium on 3D Data Processing, Visualization, and Transmission*, June 2002.

[9] Dolenc, A. 1993. Software tools for rapid prototyping technologies in manufacturing. *Acta Polytechnica Scandinavica: Mathematics and Computer Science Series 62*, 1.

[10] Fischl, B., Liu, A., and Dale, A., 2001. Automated Manifold Surgery: Constructing Geometrically Accurate and Topologically Correct Models of the Human Cerebral Cortex. *IEEE Trans. Med. Imaging* 20 (1), 70–80.

[11] Francis, and Weeks. 1999. Conway's ZIP proof. AMM*: The American Mathematical Monthly 106*.

[12] Gu, X., Gortler, S. J., and Hoppe, H. 2002. Geometry Images. *ACM Trans. Graph.* 21, 3 (Jul. 2002), 355-361.

[13] Guskov, I., and Wood, Z. 2001. Topological Noise Removal. In *Graphics Interface 2001*, 19–26.

[14] Guskov, I., Khodakovsky, A., Schr¨oder, P., and Sweldens, W. 2002. Hybrid meshes: multiresolution using regular and irregular refinement. In *Proceedings of the Eighteenth Annual Symposium on Computational Geometry (SCG-02)* (New York, June 5–7 2002), ACM Press, pp. 264–272.

[15] Haines, E. 1994. Point in Polygon Strategies. In *Graphics Gems IV*, P. S. Heckbert, Ed. Academic Press Graphics Gems Series. Academic Press Professional, San Diego, CA

[16] Han, X., and Xu, C., 2002. Topology Correction in Brain Cortex Segmentation Using a Multiscale, Graph-based Algorithm. *IEEE Trans. Med. Imaging* 21 (2), 109–121.

[17] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuetzle, W. 1992. Surface reconstruction from unorganized points. In *SIGGRAPH '92 Conference Proceedings* (July 1992), E. E. Catmull, Ed., vol. 26, pp. 71–78.

[18] Jaume, S., Macq, B. M., and Warfield, S. K. 2002. Labeling the brain surface using a deformable multiresolution mesh. In *MICCAI (1)* (2002), pp. 451–458.

[19] Ju, T. 2004. Robust Repair of Polygonal Models. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 888-895.

[20] Khodakovsky, A., Schr¨oder, P., and Sweldens, W. 2000. Progressive geometry compression. In *SIGGRAPH 00 Conference Proceedings* (2000), K. Akeley, Ed., Annual Conference Series, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, pp. 271–278.

[21] Lewiner, T., Lopes, H., Wilson, A., and Tavares, G. 2003. Efficient implementation of marching cubes cases with topological guarantee. *Journal of Graphics Tools*, 8:1–15.

[22] Lorensen, W. E. and Cline, H. E. 1987. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and interactive Techniques* M. C. Stone, Ed. SIGGRAPH '87. ACM Press, New York, NY, 163-169.

[23] Maintz, J., and Viergever, M. 1998. A survey of medical image registration. *Medical Image Analysis 2*, 1, 1–36.

[24] Montani, C., Scateni, R., and Scopigno, R. 1994. A modified lookup table for implicit disambiguation of marching cubes. *The Visual Computer*, 10(6):353–355.

[25] Nielson, G. M. and Hamann, B. 1991. The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes. *Proceedings of Visualization ’91*, pages 29–38.

[26] Ning, P. and Bloomenthal, J. 1993. An evaluation of implicit surface tilers. *IEEE Computer Graphics and Applications*, 13(6):33–41.

[27] Pulli, K., Duchamp, T., Mcdonald, J., and Stuetzle, W. 1997. Robust meshes from multiple range maps, (Aug. 05 1997).

[28] Reeb, G., 1946. On the Singular Points of a Completely Integrable Pfaff Form or of a Numerical Function. *Acad. Sci., Paris, C. R.*, 222, pp. 847–849.

[29] Rusinkiewicz, S., and Levoy, M. 2001. Efficient Variants of the ICP Algorithm. In *Proceedings of the International Conference on 3D Digital Imaging and Modeling (3DIM)*, May 2001, 145–152.

[30] Sander, P. V., Snyder, J., Gortler, S. J., and Hoppe, H. 2001. Texture mapping progressive meshes. In *SIGGRAPH 01 Conference Proceedings* (2001), E. Fiume, Ed., Annual Conference Series, ACM Press / ACM SIGGRAPH, pp. 409–416.

[31] Shattuck, D.W., and Leahy, R.M., 2001. Automated Graph-Based Analysis and Correction of Cortical Volume Topology. *IEEE Trans. Med. Imaging* 20 (11), 1167–1177.

[32] Szymczak, A. and Vanderhyde, J. 2003. Extraction of Topologically Simple Isosurfaces from Volume Datasets. In *Proceedings of the 14th IEEE Visualization 2003 (Vis'03)* (October 22 - 24, 2003). IEEE Visualization. IEEE Computer Society, Washington, DC, 10.

[33] Turk, G. and Levoy, M. 1994. Zippered Polygon Meshes from Range Images. In *Proceedings of the 21st Annual Conference on Computer Graphics and interactive Techniques* SIGGRAPH '94. ACM Press, New York, NY, 311-318.

[34] Van Gelder, A. and Wilhelms, J. 1994. Topological Considerations in Isosurface Generation. *ACM Transactions on Graphics*, 13(4):337–375.

[35] Wang, J. and Oliveira, M.M. 2003. A Hole Filling Strategy for Reconstruction of Smooth Surfaces in Range Images. In *Proceedings of the XVI Brazilian Symposium on Computer Graphics and Image Processing*. October 12-15, 2003.

[36] Weisstein, Eric W. "Hessian Normal Form." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/HessianNormalForm.html

[37] Wood, Z., Hoppe, H., Desbrun, M., and Schröder, P. 2002. Isosurface Topology Simplification. In *ACM Transactions on Graphics*, 2002.

[38] Wood, Z., Hoppe, H., Desbrun, M., and Schröder, P. 2004. Removing Excess Topology from Isosurfaces. *ACM Trans. Graph.* 23, 2 (Apr. 2004), 190-208.

[39] Wood, Z., Hoppe, H., Desbrun, M., and Schröder, P. 2004. An Out-of-Core Algorithm for Isosurface Topology Simplification. Available at http://www.multires.caltech.edu/pubs/topo_filt.pdf.

[40] PLY - Polygon File Format, http://local.wasp.uwa.edu.au/~pbourke/dataformats/ply/

[41] Scanalyze: a system for aligning and merging range data, 2006, http://graphics.stanford.edu/software/scanalyze

[42] The Stanford 3D Scanning Repository, 2007, http://graphics.stanford.edu/data/3Dscanrep/