User Guided Reconstruction of Architectural Buildings

by

Jennifer Pon Chan

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2012

Author . . . .
Department of Electrical Engineering and Computer Science
June 2012

Certified by . . . .
Frédo Durand
Associate Professor
Thesis Supervisor

Accepted by . . . .
Dennis M. Freeman
Chairman, Masters of Engineering Thesis Committee

User Guided Reconstruction of Architectural Buildings

by

Jennifer Pon Chan

Submitted to the Department of Electrical Engineering and Computer Science
on June 2012, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

Evaluating the current structural stability of historical buildings is a challenging task. Even the stability and structure of the original architecture is often poorly understood and difficult to calculate. Using a laser point cloud scan as a template from which to base the geometric reconstruction, we can more accurately capture the current state of the cathedral. In order to analyze the structural stability, the building must be discretized into a collection of adjacent geometric primitives. We have created a system that allows a user to do a 3D reconstruction of these historical buildings, particularly cathedrals, by fitting geometric primitives to the point cloud scan. There are three major components to aid the user in reconstructing the architecture: the user interface, data visualization, and the geometric fitting algorithm. The system efficiently renders the point cloud from which the user can then select portions of the point cloud to model. After the user provides a rough estimate of the geometry, the geometric fitting algorithm snaps the selected geometry to the user's point cloud selection. Once the cathedral is constructed and geometries are adjacent, we can evaluate the forces using structural gradients and predict changes to the geometry to form a more stable structure. Our program allows users to output volumetric 3D models with adjacent blocks such that the structural stability may be evaluated.

Thesis Supervisor: Frédo Durand
Title: Associate Professor

Acknowledgments

I would like to thank Fredo Durand for taking me on as a masters student and offering his infinite wisdom and advice. I learned more from this thesis than I might have imagined. His vision, advice, and guidance were a constant driving force behind this thesis. His constructive comments have helped me become a better writer, researcher, and software engineer. In addition, I would have been lost in this world of architecture if it weren't for John Oschendorf. Despite my lack of knowledge of architecture, he would greet me with a smile, and gently introduce to me the complex world of gothic architecture.

This project would not have been possible without Emily Whiting, whose force evaluation program led to this idea in the first place. I would like to thank her for allowing me to contribute to her project, and taking the time to mentor me initially as a UROP. As a direct supervisor, she was crucial in my understanding of some of the more complex algorithms. Even after her leave from MIT, she continued to offer helpful advice. I would also like to thank Valentina Shin, who is extending Emily's work. She provided hours of help in getting programs up and running. She also offered valuable input and help with evaluating the system.

Sylvain Paris offered a wealth of knowledge, resources, and related work. Vladimir Bychkovsky provided helpful discussions about PCA, as well as insightful discussions about life in general. Robert Wang, who gave me my first UROP in the graphics group, continues to inspire me, and I am honored that I was able to contribute to his project. His dedication as a mentor and a teacher cemented my decision to stay in the graphics group. I would also like to thank Jaakko Lehtinen, my 6.837 Computer Graphics professor. His engaging lectures affirmed my passion for computer graphics. Finally, I would like to thank the members of the graphics group for not only their valuable input and random knowledge, but also for the comradarie and friendship they provided.

Last, but not least, I would like to thank my friends and family for their support throughout this process. Thank you for being there through my bizarre working hours and giving encouraging words. The utmost thanks goes to my family. This thesis is the product of their love, generosity, dedication, and teaching. I would not be here today without them.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Evaluating the current structural stability of historical buildings, especially cathedrals, is a challenging task. Often these buildings are centuries old, and even the structure of the original architecture is often poorly understood and difficult to calculate. Although blueprints of these buildings exist, these blueprints are only 2D representations of the building, not fully encapsulating the 3D geometry. In addition, they are not accurate representations of the building's current state, as these old buildings are often deteriorated and have been modified or refurbished.

For example, the Tarazona Cathedral features many lesions and changes due to the deterioration of the cimborio and clerestory arches [13]. Not only has chemical attack significantly degraded the structure, but changes in structure and the mantling and dismantling of various architectural features such as the flying arches created disruptions in the stability. The Mallorca Cathedral also suffers from a variety of deformations, including large cracks that run through the central nave and arches [13] . Masonry columns were constructed later to support failing flying arches and the piers have suffered from curvature warping and lateral displacement. Additions and modifications need to made to these buildings to better guarantee safety and structural stability.

Using a technique called spatial archaeology, Professor Andy Tallon acquired a laser scanned point cloud of the Bourges cathedral, containing over a hundred million points. Point cloud scans such as these provide a more realistic 3D representation of the current state of the cathedral.

To evaluate the forces on structures such as these, Whiting [19] presented a system that calculates the structural gradients and suggests changes to the geometry to form a more stable structure. However, input into Whiting's system has traditionally been simple geometric elements, as opposed to the full structure of the cathedral.

We have two main problems: acquiring an accurate input 3D model and performing the structural evaluation of these models. In Section 1.2.1, we'll discuss the idea of surface mesh reconstruction and how it ultimately does not fully characterize the volumetric properties necessary to evaluate the forces on the structure. Section 1.2.2 covers the disadvantages of procedural modelling and how it may not provide a fully accurate model. Therefore, we aim to create a system that takes as input the point cloud scan, allows the user to do a 3D reconstruction via geometric shape fitting, and outputs the geometries into Whiting's system such that the structural stability may be evaluated.

1.2 Background

1.2.1 Surface Mesh Reconstruction

Rendering the point cloud as a surface mesh is one way to reconstruct the original structure. Aiger, et al [2] uses 4 point co-planer congruent sets and registration to create a surface mesh. Alliez, et al [3] uses a Poisson surface reconstruction to recover the mesh of a 3d object. Delauney triangulation and graph cuts are also common ways of recreating a 3D mesh [7]. Rusinkiewicz and Levoy [15] proposes an optimized multi-resolution mesh renderer, which could be used to speed up the processing of this large dimensional point cloud mesh reconstruction.

More specifically to the reconstruction of architecture, prior work has also used mesh based reconstruction. Wu, et al. will present their work on Schematic Surface

Reconstruction [20] at CVPR 2012, which uses profile curves and swept surfaces to characterize point clouds of architecture as a surface mesh. In 2009, Edouard Grave performed a surface reconstruction based on the point cloud scan of Bourges. After reconstructing the point cloud surface, he cut each of the surfaces into horizontal slices, with each slice separating each geometric primitive. However, classifying the cathedral as a mesh does not lend itself to the evaluation of forces. The most glaring example of this is the surface reconstruction of walls as planes. As planes are flat surfaces, the structural stability cannot be accurately computed because walls also have depth and mass. Although the mesh reconstruction technique may give the most fine grain details, it does not accurately capture the 3D geometric structure and does not provide the volumetric models needed in order to properly calculate the forces. Ultimately, surface mesh reconstruction only creates a facade of the structure and is missing the important volumetric information necessary to evaluate the stability of the structure.

1.2.2 Procedural Modeling

Ramamoorthi and Arvo [12] approached the problem of generating models from 3D scans by modeling them as generative models, i.e. a generalization of a swept surface defined by continuously transforming arbitrary curves. Their algorithm has two phases. The first is recognition, in which an appropriate model is chosen from a library of potential objects. The second is parameter estimation, which adjusts the model to best fit the data by performing curve refinement. As a cathedral can be modeled as a collection of geometric primitives, we similarly will adopt the recognition step of the algorithm, allowing for an initial guess of the root model. However, because our library of geometric primitives will be basic building blocks, curve refinement does not best meet our needs for the parameter estimation part. In addition, because Whiting's force evaluations rely on the primitives being six-sided blocks, curved structures are an incompatible input.

Parish and Mueller [11] presented a system that procedurally rebuilds cities based off of 2D maps by forming a grammar based on L-sytems. However, because they use

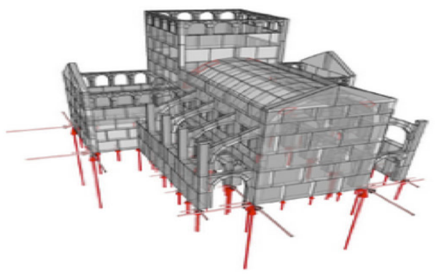
2D maps to rebuild these 3D structures, depth information is lost, making it undesirable for evaluating structural stability. Mueller et al [**?**] then extended this grammar to procedurally model buildings. Similarly, Whiting proposed a procedural modeling system that uses a grammar to build cathedrals [19]. This approach, however, relies heavily on the architect who is familiar with the existing structure and may incorrectly assume that the blueprints are an accurate representation of the current structure. We would like to be able to take current point scans of cathedrals and rebuild the structure according to these scans. Using the detailed point scans instead of blue print models allow us to more finely identify weak points in the current state of the structure.

Nan, et al [10] uses point cloud scans to rebuild large scale urban scenes using SmartBoxes, which snaps axis-aligned rectangular cuboids to urban point cloud scans using RANSAC and interactive user feedback. While this work is closest to ours, fitting the point cloud parameters in an urban architecture versus a cathedral relies on several different assumptions. Nan's technique relies on repeatability of structural elements and design patterns to automate the process of building the entire building. On the other hand, a cathedral tends to have much less repeated elements and many small, fine intricacies that must be accounted for.

1.2.3 Data Fitting

In 2010, Lee implemented registration to fit geometric primitives to the point cloud. However, registration solved for the rotation and the translation of the primitive, but did not include scaling or stretching. In the context of computer vision and robotics, Garcia [6] uses RANSAC to fit primitives to the point cloud. Schnabal et al [16] also uses RANSAC to classify point clouds from primitive shapes. In addition, AutoCAD has an plug-in that allows users to extract shapes from a point cloud [1].

The iterative closest point algorithm is also widely used to register and align the outputs of 3D scanners. Schneider, et al. used ICP to align morphable head models to scans [17]. Rusinkiewiz and Levoy experimented with different algorithmic variations on ICP [14]. Beginning with two meshes and an initial guess of their relative rigid-


(a) Whiting's procedural model

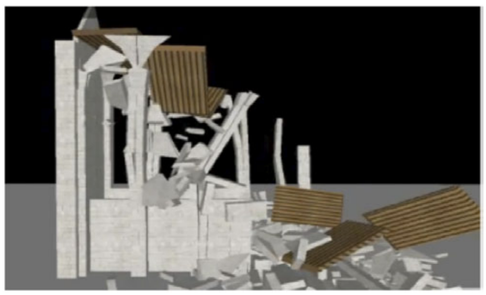

(b) Simulation of Building Falling

Figure 1-1: Whiting's Force Evaluation Work

body transform, it iteratively refines the transform by matching pairs of points and minimizing the error.

1.2.4 Stability Evaluation

Once the cathedral is classified as a collection of geometric primitives, we must be able to evaluate its structural stability in some way. In addition, we would like to suggest improvements to the structure on how to make it more stable.

Force evaluation has often been conducted using finite element methods and a continuous damage model [13]. Finite element analysis often focuses on the visualization of weak points in the structure. However, it does not directly suggest ways or alternatives for the designer to create a more stable structure.

In Siggraph Asia 2009, Whiting et al. proposed a system for a Procedural Model of Structurally-Sound Masonry Buildings [19]. Her system calculates the forces and tensions from one geometric primitive onto another by calculating the adjacencies, or interfaces between each of these blocks. Using these force models, she can use an iterative procedure and calculate the gradients with finite differencing to minimize the tension on each of the blocks in order to suggest a more stable structure. Because of its ability to handle masonry structures, we will use Whiting's program to evaluate the structural stability of the fitted geometric primitives. This system will provide the structural input into Whiting's program.

1.3 Thesis Overview

Using these laser scan points, our goal is to model the cathedral as a set of volumetric geometric primitives. Once the cathedral is modeled as a set of adjacent geometric primitives we can evaluate its structural integrity using force models. Modelling the laser scan as geometric primitives consists of three major components: the user interface, the data visualization, and geometric fitting algorithm. These three components help us to achieve our two main objectives– create a 3D volumetric model based off of the point cloud scans and ensure the blocks are adjacent in order to export them to the force evaluation system. The details of the work are provided as follows.

Chapter 2 overviews the system in terms of the engineering design and a high level description of the user workflow when modeling the cathedral.

Chapter 3 handles the visualization of the point cloud. The laser point scan of Bourges consists of over 2 GB of data, roughly over 100 million points. The visualization component presents the methods for quickly rendering the laser point scan and maximizing the ease in which a user can visually extract key features. Various optimizations were utilized such that the user could more easily navigate through the point cloud.

Chapter 4 reviews the user interface of the system. It was important to have a flexible, robust user interface since the user is the primary agent in reconstructing the cathedral. Selection of points, geometry, and faces had to be implemented and optimized for ease and speed. Manual and mouse manipulation of the geometry gives the user fine grain control of the initial geometry fitting.

Chapter 5 provides a summary of the geometric primitives that were used to model elements of the cathedral.

Chapter 6 summarizes the details the algorithms used to snap geometric primitives to the point cloud. The primary algorithm is ICP, or the iterative closest point method. Given points in a set, the iterative closest point method will find the closest point to a geometric primitive and find the least squares transformation to minimize the error. Iterative closest point allows for any transformation; however, in some cases, shear may not be desirable. Thus, the chapter also highlights polar decomposition as a way of removing shear. In cases where an object is not axis-aligned, the user must realign the shape by choosing corresponding points on both the geometry and the point cloud.

Chapter 7 describes how shapes are made adjacent. In order for the forces to be properly analyzed, each adjacent geometry must have adjacent faces. ICP does not necessarily handle this, so we provide a user interface and algorithm for snapping adjacent geometries together.

In Chapter 8, I present some 3D reconstructions of portions of the cathedral, as well as the force evaluation of these reconstructions as it is run through Whiting's system.

Finally, in Chapter 9 we provide a conclusion and the future possibilities of this work.

Chapter 2

System Overview

To reconstruct point cloud scans into 3D volumetric geometric models, we require three basic components: visualization, user interface, and geometric fitting algorithms. The geometry must fit to the point cloud, as well as snap to other geometries to ensure adjacency. Once the model is created, it is then imported into Whiting's system so that the structural stability of the model can be evaluated.

2.1 Visualization

The viewer has to render three basic components, the point cloud of the cathedral, the selection, and the geometric model. Figure 2-1 shows the hierarchy of the object models.

Because the point cloud is so large and dense, a grid box acceleration structure had to be implemented in order to optimize the viewing.

2.2 User Interface

Figure 2-2 illustrates the process the user should take when reconstructing the structure.

The user interface has to handle four main tasks: selection, visualization, geometry fitting, and geometry handling. Different tools are associated with each of these tasks

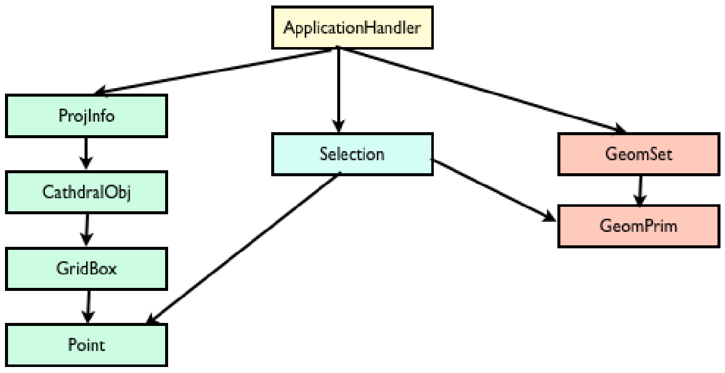


Figure 2-1: Object Model

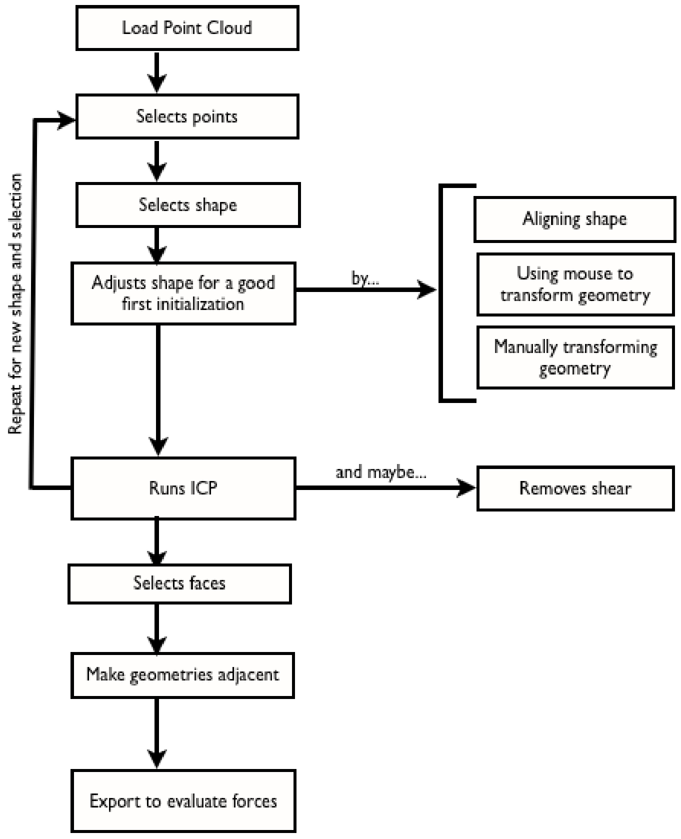

Load Point Cloud
Selects points
Selects shape
Adjusts shape for a good first initialization
by...
Aligning shape
Using mouse to transform geometry
Manually transforming geometry
Repeat for new shape and selection
Runs ICP
and maybe...
Removes shear
Selects faces
Make geometries adjacent
Export to evaluate forces

Figure 2-2: User Pipeline

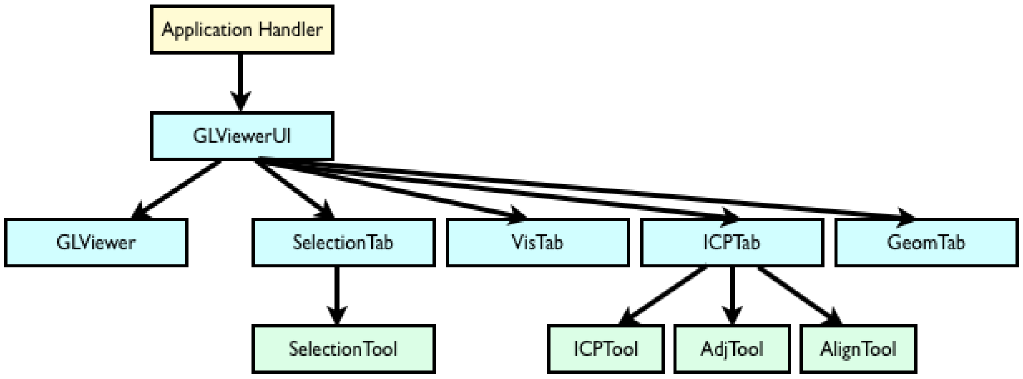


Figure 2-3: Model View Controller diagram

and are responsible for implementing the algorithms for each task. The user must be able to select points, shapes, and faces, as well as manipulate the geometry placed in the scene. The UI for the visualization allows users to more easily navigate through the large point cloud, and we must also provide UI for allowing the user to specify parameters for the geometric fitting algorithms.

Figure 2-3 shows the layout of model view controller. The Application Handler is the high level controller that funnels the communication between the model, the different user interface components, and the OpenGL viewer.

2.3 Geometric Shape Fitting

Because surface mesh reconstruction fails to encapsulate the volumetric data needed to evaluate the forces and procedural modelling does not capture the current state of the structure, this system uses geometric shape fitting to create the 3D model. The geometric shape fitting primarily relies upon the iterative closest point method to snap geometric primitives to point cloud selections. It solves for the best fit using constrained least squares. To handle edge cases, we also allow the user to also mark correspondences before doing the fitting. Polar decomposition was also used to allow the user to remove shear if needed.

2.4 Force Evaluation

As described earlier, we will use Whiting's system to evaluate the structural stability of our model. Since this system will be providing input into Whiting's system, our output must adhere to the following conventions:

- The model must be a valid obj file
- Each geometric primitive must either be a six sided block or only contain six sided blocks
- The model must be centered at the origin
- Geometry must be adjacent
- The bottom of the model must be parallel to the floor

Given a compatible obj file, we can then find the interfaces and calculate the tensions acting on each block. Then, using structural gradients, Whiting's program will suggest changes to the geometry, eventually iterating to a more stable structure.

Chapter 3

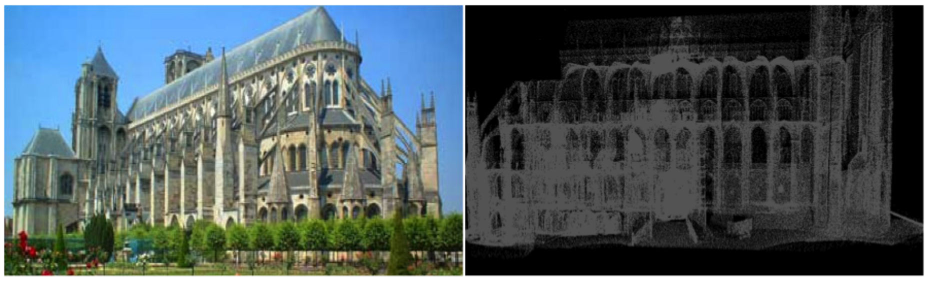
Visualization

3.1 Introduction

The visualization of large point clouds is challenging, both in terms of speed and memory. To handle the large amount of data, this system partitioned the cathedral into axis-aligned grid boxes. We optimized the rendering by using these grid boxes to selectively render areas of the cathedral and by sampling the data at different resolutions.

3.2 Data Structure of the Cathedral

Because the cathedral point cloud is greater than 2 GB of data, the visualization was optimized to handle the large amount of points. As a pre-processing step, the full



(a) Bourges Cathedral in France (b) Point Cloud Render of Bourges

Figure 3-1: Bourges Cathedral

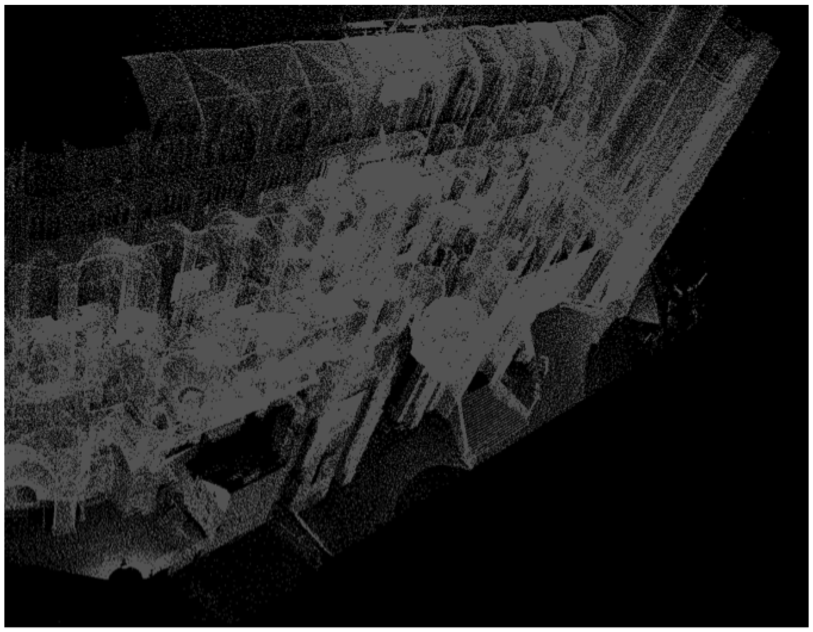


Figure 3-2: Full Point Cloud Render

model was broken into individual, smaller files so that the user could view portions of the cathedral without having to load the huge file. After parsing the 2 GB file and finding the minima and maxima along each axis, the cathedral was broken into 10x10x4 grid boxes based on these point cloud extrema. Saving each grid box as a separate file ensured that sections could be loaded fast and independently.

Because the model is so large, the user in most cases will not want to view the whole cathedral at once. Loading the whole point cloud model without sampling any of the points takes about ten minutes to load. Therefore, we dynamically load sampled portions of the cathedral. Using the visualization tool, the user can specify the span of gridboxes. Any unloaded grid boxes are loaded on demand.

3.3 Visualization Options

The current visualization interface supports two different modes. The user may choose to highlight a current span of gridboxes. Alternatively, the user can choose to only view a span of grid boxes. This tool is useful in that when viewing a point cloud, it can be harder to distinguish planes and shapes. Comparatively, discrete geometry

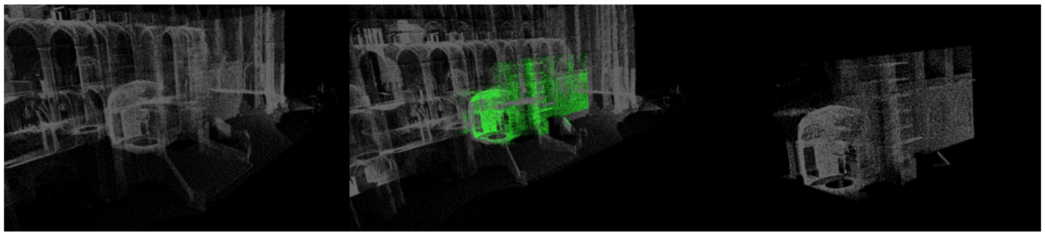

(a) Full view (b) Grid box highlighted (c) Viewing single grid box

Figure 3-3: Visualization Options

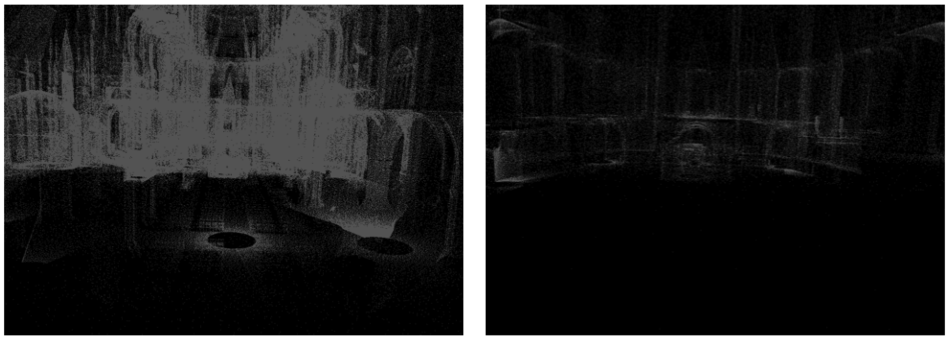

(a) Full Render (b) Fast Render when mouse is moving

Figure 3-4: Fast Rendering

creates occlusions that allow the user to more easily distinguish architectural elements. Because the point cloud has holes, it is much harder for the user to see the boundaries. By allowing the user to visualize a small subsection of the cathedral, it becomes easier to navigate the scene and fit geometry.

To view the cathedral, we provided similar functionality as other 3D modelling programs. Using the mouse, the user can manipulate the camera to manuever through the scene. In addition, other shortcuts, such as Ctrl+Alt+C centers the camera around the current point selection.

3.4 Multi-Resolution Rendering

The viewing environment also supports multi-resolution rendering. Because the cathedral contains a large number of points, when the user manipulates the mouse we only draw a percentage of the points on the screen to optimize the speed of the application. When the camera stops, the full number of points should be displayed.

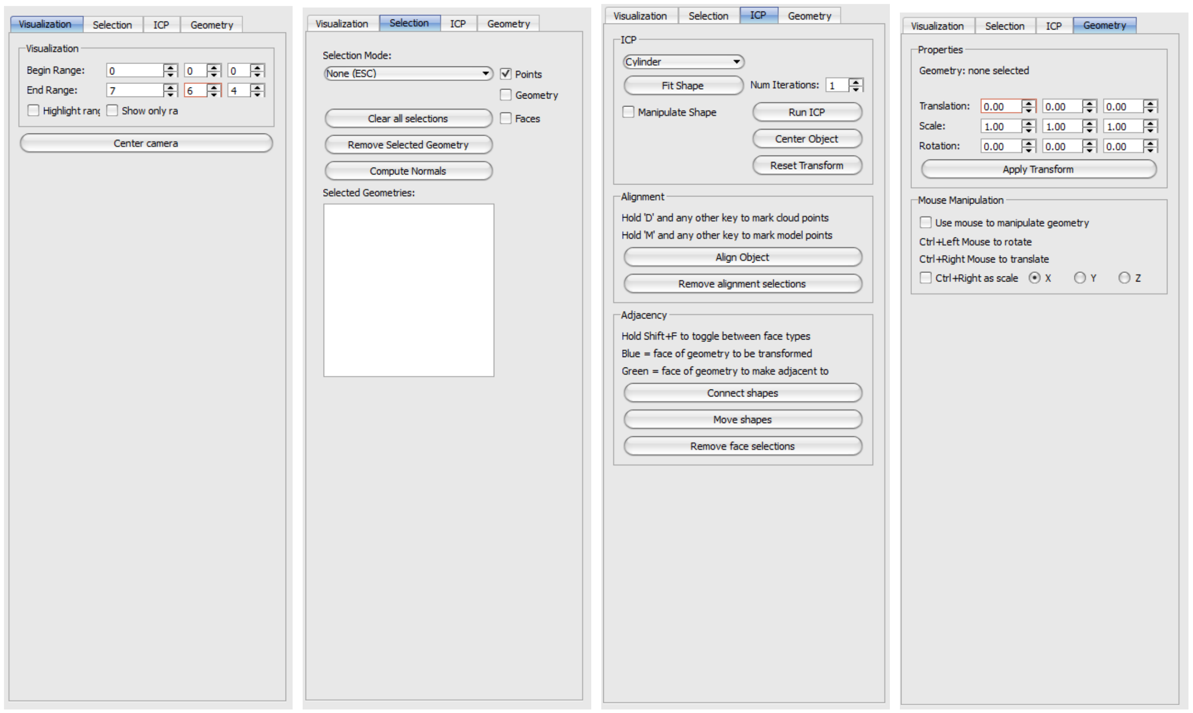
Chapter 4

User Interface

The user interface consists of four main components: visualization, selection, ICP, and geometry. The visualization UI handles the dynamic loading of grid boxes. The selection UI handles selecting points, geometries, and faces. The ICP UI handles the data fitting components, namely the ICP algorithm, the alignment algorithm, and the adjacency algorithm. Finally, the geometry UI gives the user controls to transform selected geometry.

4.1 Selection

The user must choose an appropriate set of points to match with the geometric primitive before ICP can be applied. The user interface supports the selection of points, geometries, and faces, along with tools to subtract or add points from a current selection. It also supports taking the union, intersection, and difference based upon two screen-space bounding rectangles. We select objects by projecting the point data from the 3D world space to the 2D camera view. If these objects lie within the bounds of the axis-aligned rectangle drawn to the screen, then they are selected.



Begin Range: 0 0 0
End Range: 7 6 4
Selection Mode: None (ESC)
[✓ Points] Geometry Faces
Cylinder
Num Iterations: 1
Hold 'D' and any other key to mark cloud points
Hold 'M' and any other key to mark model points
Hold Shift+F to toggle between face types
Blue = face of geometry to be transformed
Green = face of geometry to make adjacent to
Geometry: none selected
Translation: 0.00 0.00 0.00
Scale: 1.00 1.00 1.00
Rotation: 0.00 0.00 0.00
Ctrl+Left Mouse to rotate
Ctrl+Right Mouse to translate

(a) Visualization Tab (b) Selection Tab (c) ICP Tab (d) Geom Tab

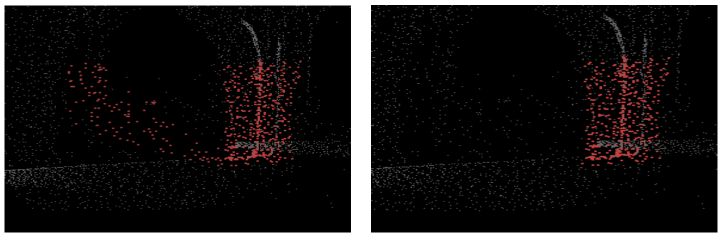
Figure 4-1: UI Tabs

4.1.1 Point Selection

Point selection is a crucial feature of this tool, as the user must be able to select points that they want to fit to a geometric object. The user selects points by clicking and dragging a 2D rectangle onto the screen. For every grid box in the cathedral, we project each of the points to the screen. If the projected point is within the bounds of the rectangle, it is then selected. However, we also limit the selection to a certain z-depth, as we do not want to select points that are too far away. We also support subtraction, union, intersection, and difference.

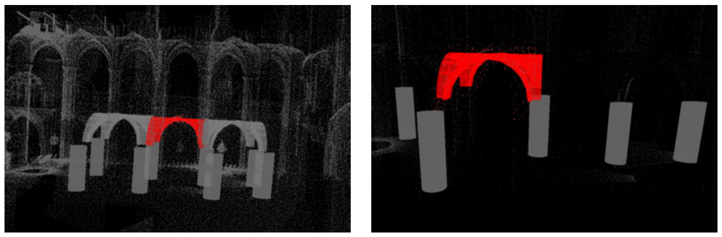
4.1.2 Geometry Selection

The user must also be able to select geometries so that they can manipulate the transform of geometric objects. Alternatively, they may want to delete the geometry, export the geometry, or copy and paste the geometry. In a similar manner as point selection, the user clicks and drags a 2D rectangle onto the screen. For every shape,



(a) Initial selection of points (b) After removal of points

Figure 4-2: Adding and Removing Points



(a) Geometry Selection Example 1 (b) Geometry Selection Example 2

Figure 4-3: Geometry Selection

we project their vertices onto the screen. If any of the projected coordinates of the vertices fall within the bounds of the selection rectangle, then the geometry is selected. When a geometric primitive is selected, it is highlighted red, and shows up in the GUI under the selected geometries list, where the user can also see the ids of the selected geometry. Similar to point selection, subtraction, union, intersection, and difference is supported.

4.2 Geometry Manipulation

The application gives the user different ways to transform the geometry and apply a best guess transformation to the shape. When the shape is given a close initial guess, ICP has a much better chance of converging. In most cases, the initial scaling does a pretty good job at giving a good first guess for ICP. However, in certain edge cases, such as non-axis aligned shapes, the user might first have to manipulate and transform the object to aid ICP.

4.2.1 UI Manipulation

A GUI allows the user to specify the rotation, translation, and scale, and apply that transform to the selected geometry. From the tx, ty, tz values specified for translation and sx, sy, and sz values specified for scale, we form the translation matrix T and the scaling matrix S.

The rotation matrix $R = R_z(\gamma)R_x(\beta)R_y(\alpha)$, whose Euler angles are α, β, and γ, where

$$R_x(\theta) = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) & 0 \\ 0 & sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}.$$

$$R_y(\theta) = \begin{vmatrix} cos(\theta) & 0 & sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\theta) & 0 & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}.$$

$$R_z(\theta) = \begin{vmatrix} cos(\theta) & -sin(\theta) & 0 & 0 \\ sin(\theta) & cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}.$$

The matrices are then combined to M and applied to the geometry. If the geometry has a transform G, then the geometry's new transform G' is

$$M = TSR$$
$$G' = MG$$

After the transformation is applied to the geometry, all of the values are reset. Other methods were also explored, such as storing the global translation, rotation, and scale for each geometry. Maya, the modeling program, does something similar, keeping track of the geometry's general translation, rotation, and scale. However, because we run ICP to figure out a best fit transform for a geometric object, we cannot simply keep track of a translation, rotation, and scale for every geometric object since ICP also introduces a shear component. We tried to use polar decomposition, described in Section 6.2 to extract the translation, rotation, and scale for the geometry. However, because polar decomposition does not yield unique and reliable results, this approach turned out to be impractical.

4.2.2 Direct Manipulation

On top of the manual GUI manipulation, we can use the mouse to rotate, translate, and scale geometry. Most of this extends libGLViewer's manipulated frame interface, which by default, uses Ctrl+Left Mouse to Rotate and Ctrl+Right Mouse to translate. For interactive scaling, when the user clicks a checkbox, it allows them to use Ctrl+Right Mouse to scale.

4.3 Undo/Redo

Because this application needs to be fairly robust and involves a lot of user interaction, we support undo and redo. Undo and redo is a crucial feature in case the user applies an unwanted transform to a geometry.

Undo and redo is implemented each as a simple stack. Each transformation is saved as an Action, which gets pushed to the undo stack. Before each new transformation is applied to a geometric primitive, we save the old transformation of the geometry. Pressing Ctrl+Z pops the action off the undo stack, pushes it onto the redo stack, and re-applies the old transformation to the geometry. Pressing Ctrl+R pops the action off of the redo stack and applies the old transformation to the geometry.

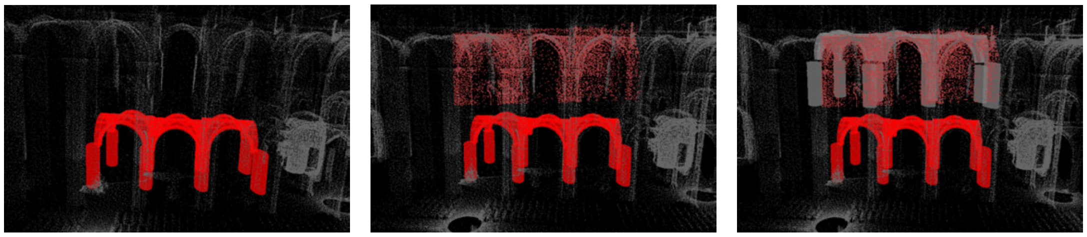

(a) Geometry to be copied (b) New selection (c) Geometry pasted

Figure 4-4: Copy and Pasting Geometry

4.4 Copy/Paste

Users also are able to copy and paste geometry. In particular, this is useful for repeating architectural features, such as columns and arches.

The user can select one or more geometries via the geometry selection tool and then can copy the selection. They must then select a new point cloud selection to inform the program where the copied geometry should be pasted. Ctrl+P or Edit->Paste pastes the copied geometry at the center of the selected points. If there is no current selection, it copies the object to the same position.

4.5 Project Settings

The project .config file contains all of the project settings for loading the project. The file format is as follows:

```
Name of original point cloud file
Default .geoms file to save model to
Name of file for loading grid boxes
Number of grid boxes
Max value in point cloud
Min value in point cloud
```

4.6 Import/Export

4.6.1 Loading and Saving the Model

The user can save the model by pressing Ctrl-S or from the menu File->Save or File->Save as. The program keeps track of the current file the user is saving to, which is either the default file specified by the project configuration file, or whatever was the last file the user specified with the save as option. Instead of saving every single object's vertices and faces in the scene, the program saves the name of the file associated with the object and the current transformation of the object. This is saved as a .geoms file under the project's file directory. The file format is as follows:

```
GeomID
GeomFile
Matrix Transformation of geometry
```

Since the user can save a model, it follows that the user must be able to load a model so that they can model different sections of the cathedral at a time. File->Load Model allows the user to open a .geoms file.

4.6.2 Export

The user can either export a selection of geometry or the full model. It gets exported into an obj file that is compatible with Whiting's program. As noted before, this obj file must only contain blocks that have six sides and quadrilateral faces. The user designates a block with the syntax

```
g blockType_shapeNumber_blockNumber
```

which separates every six faces.

Chapter 5

Geometry Primitives and Cathedral Architecture

To evaluate the forces of the structure, each of the components needs to be a volumetric geometric primitive. The system contains a library of geometric primitives that is able to classify the expanse of the structures in the cathedral.

5.1 Primitive Library Selection

We accumulate a library of primitives, a selection of geometric primitives that can accurately classify a cathedral. Once the user selects some points, they can then choose which geometric primitive best classifies their point cloud selection. The geometry is then appropriately scaled and centered at the centroid of the point cloud. The user can then use ICP or manipulate the shape with their mouse to achieve the desired fitting.

Coming up with a grammar of library primitives was a challenging task, as cathedrals have very different structures, and each structure may have detailed ornate structures. Fortunately, because we are interested in its forces and structural stability, we can discretize a lot of these structures into very basic geometric building blocks.

Because of our initialization process, each geometric primitive must be unit sized,

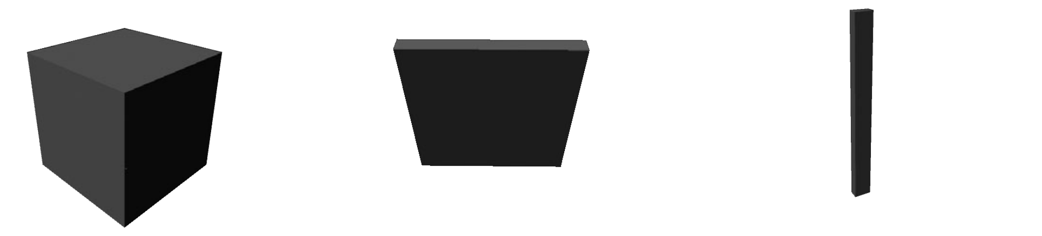

(a) Unit scaled cube (b) A cube can also be transformed to form a wall or plane (c) A cube can also form a pillar or column

Figure 5-1: Basic Cube

centered at the origin, and be a valid obj file. A script converts any obj file to be compatible with the system, normalizing and centering the shape. To be used within the program, each obj file needs to be written to ShapeDictionary.txt, which contains the names and references to the obj files.

5.2 Basic Geometries

Our primitive library features some basic building blocks. Of course, there is a basic cube which can be scaled and stretched. The cube covers a lot of potential architectural features, such as planes, walls, and floors, as seen in 5-1. In addition, at its core, a cathedral is built from a collection of simple stones and bricks, so if the user so decides to, they could arduously model the whole cathedral as a collection of cubes.

Although there are many different kinds of columns with different ornamentation, most can be modeled as a simple cylinder. However, because Whiting's program only allows for six sided blocks, the cylinder gets converted to a block in a post-processing step when it gets exported.

Domes and vaults are also common features of cathedrals. The shape dictionary also contains a half sphere, apse/half dome, and a quarter dome. Again, these objects are more for aesthetic purposes, as these are not six sided shapes.

(a) Unit scaled cylinder (b) Non-cube block

Figure 5-2: Other Basic Geometries

5.3 Special Cases

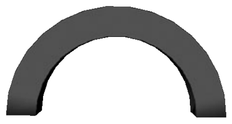
5.3.1 Arches

Arches actually prove to be one of the most difficult features of the cathedral to characterize, especially since there are many different arches, and these different characteristics impact the structural stability of the arch itself. As opposed to the columns, where ornamentation does not have a real impact on its structural stability, the structural stability of the arch is highly dependent on its design.

Simple rounded arches prove to be less complex to model because of its rounded spherical nature. However, pointed arches prove to be much more complex. In addition, pointed arches are not always symmetrical, with each side of the arch having a different center of rotation. Future work, as an extension of Whiting's procedural modeling work, would allow the user to procedurally model their own arch, meaning they would have to specify the center of rotation for each side of the arch. In lieu of this, the introduction of the half arch as geometric primitive allows ICP to scale and stretch such that it could be appropriately fitted and transformed according to the point cloud.

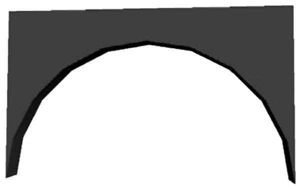
5.3.2 Statues

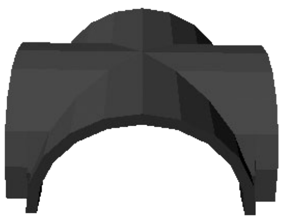
There is also be geometry that we cannot model with simple geometric primitives, such as statues. However, often these statues are on parapets or on ceilings; they add weight to the cathedral structure and still need to be included. We choose to

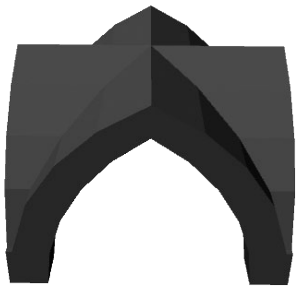

(a) Curved Arch

(b) Pointed Arch

(c) Half of a pointed arch


(d) Spandrel


(e) Groinvault


(f) Gothic Groinvault

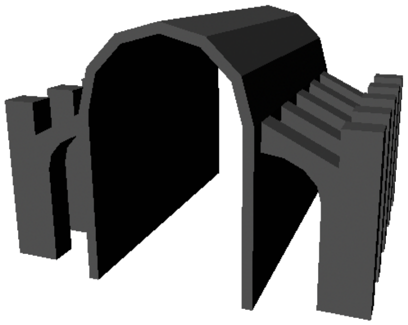

(g) Flying Buttress

Figure 5-3: Different Arch Types

model these statues as a single block of roughly the same size, for the simple sake of computation. As future work, we could also potentially recreate the statue using gradient reconstruction; again this would only be for aesthetic purposes due to the constraints of the force modelling program. However, discretizing these structures as a single block is enough to calculate the impact of the statue on its underlying base.

5.3.3 Wall Thickness

Because walls are not simply planes, we must also provide a user interface such that a user could hypothesize the thickness of a wall or a ceiling, as it might not have been captured by the point cloud scan. The GUI interface for transforming geometric primitives allows the user to change the thickness (scale), position, or rotation. Because the laser scans might not reveal every single component of the cathedral, this re-emphasizes why mesh reconstruction is not the desired solution for this problem. For example, there often are hidden structures behind walls, providing extra structural stability to the building. Being able to enter in wall thickness per geometric primitive (as opposed to a general mesh) helps compensate for this invisible geometry.

Chapter 6

Geometric Primitive to Point Snapping

After the user selects the points from the cloud and has approximately placed the corresponding geometric primitive in the scene, the geometric primitive then needs to snap to the data. Various techniques of fitting geometric primitives to a point cloud were discussed in Section 1.2.3. We use the iterative closest point method, or ICP, to fit the geometric primitive to the point cloud data.

6.1 Iterative Closest Point Method

We implement the primitive to point matching with the iterative closest point algorithm. The Iterative Closest Point Algorithm is 4 steps:

1. Associate points from the test set to the model set
2. Estimate the transformation from the geometric model to the point cloud using least squares
3. Transform the primitive using the estimated parameters
4. Iterate to reduce error

6.1.1 Initialization

ICP heavily relies on having a good initial guess; therefore, we make several initial assumptions and give the user controls in order to improve the performance of the algorithm. If the geometric primitive is initially positioned in a similar orientation as the point set, less iterations are required to converge. Therefore, by allowing the user to manipulate the primitive and specify the initial correspondences, ICP will converge much faster.

A rough initialization places the geometric primitive at the centroid of the point cloud selection, where the centroid is the center of mass defined as:

$$centroid = \sum p * 1/n \text{ where } n \text{ is the number of points}$$

Similar to Nan et al [10], we assume the structure is axis-aligned, find the extrema of the point cloud selection, and scale the geometry according to the point cloud extrema. In most cases, this gives a pretty good guess for ICP, but there are drawbacks when the shape is not axis-aligned. This sometimes causes greater problems for users, as they must correct the effects of the incorrect auto-scaling. We address these alignment issues in Section 6.3. A later observation found that placing the object at the centroid was not robust in handling outliers. The more outliers the point cloud contained, the more offset the center was. Instead, we calculate the median, discard the points relative to the median, and then center the shape at that point.

6.1.2 Nearest Point to Primitive

After the initial fitting, the next step is to find the corresponding snapping points from the cloud to the model. Although there are many different methods for finding the closest point, as described by Szymon Rusinkiewicz [14], we approach this problem by matching each point to the nearest triangle on the primitive mesh. For every point in our laser scan, we find the closest intersection from the point to our model mesh. For every triangle in our mesh, we project point p onto the triangle. The closest point p' from p to the triangle may lie either on the face, the edge, or one of the vertices. The closest point of intersection on the model mesh yields p', the matching point on our model to point p from our dataset.

As an alternative, our algorithm could have switched the order of the loops, finding every point p in the point cloud that coordinates with the vertices of our faces. However, because the point cloud may have incomplete sections that may not directly align with a geometric primitive, for each point in the geometry, there may not be an exact match in the point cloud. Therefore, it makes more sense to find the closest point on the geometry from p in our point cloud.

6.1.3 Projected Point on Face

For each point p in the point cloud, we must first find the closest point p' on the selected model geometry. First, for every face, project point p on face f of the geometry. Given a face with vertices v_1, v_2, and v_3, we can define the following terms:

$$\text{Normal of the plane: } n_{plane} = \frac{((v_2-v_1)\times(v_2-v_3))}{||((v_2-v_1)x\times(v_2-v_3))||}$$
$$\text{Distance to the plane: } d_{plane} = (p - v_1) \cdot n_{plane}$$
$$\text{Projected point on plane } p_{planeproj} = p - d_{plane} * n_{plane}$$

We need then to test if this projected point $p_{planeproj}$ is within the bounds of the plane, because not every projected point may lie on the face itself (it may lie outside the face). If the point lies within the bounds of the plane, we then need to compute the distance from our cloud point p to our projected point on the plane. If this distance is less than our stored minimum distance, then our new closest point becomes the projected point on the plane $p_{planeproj}$ and the new minimum distance d_{plane} is the Euclidean distance from our projected point to p.

6.1.4 Point in Plane

To calculate whether a point is in the plane, we must first calculate the centroid of the face. For a face with i vertices $v_1, v_2, \ldots, v_i$, the centroid c is defined as

$$c = \sum v * 1/i$$

We then find the normal unit vector n_{plane} of the plane. For each edge or halfspace, we have to test if the point is inside or outside. First, we have to calculate the normal to the edge, which we can calculate as the cross product of the plane normal n_{plane} and the edge e, where edge $e_i = v_{i+1} - v_i$.

$$n_{edge} = n_{plane} \times (v_{i+1} - v_i)$$

We also need to determine the direction of the normal of the edge and make sure that all of the normals point outward. We determine the angle by taking the dot product of the vector from the point on the face to the centroid and the edge normal. If the angle of the normal is greater than 0, then the normal points inward with an angle of less than 90 degrees. We want to make every normal point outward, so we reverse the direction of the normal.

We then need to find the angle to the plane, which we calculate as

$$\theta_{plane} = (v_i - p) \cdot n_{edge}$$

If the angle to the plane is greater than 0, then the point is outside of the plane. If this is the case then we stop iterating through the rest of the edges, because we know with certainty that the point is not in the plane. However, if the angle to the plane is not greater than 0, then we need to test the other edges to ensure that all of the edges contain the point. If for every edge on the face, the point is inside with respect to each edge, then the point is in the plane.

6.1.5 Edge Projected Point

However, if the projected point does not lie in the boundaries of the face, we then need to figure out if the point lies on one of the edges of the face. For each of the edges e on the face, we project p onto e. We then need to test if this projected point is within the line edges. Given an edge with vertices v1 to v2, the line is parameterized as:

$$P(t) = edge * t + v_1 \text{ where } x = v_1 \text{ when } t = 0$$

We therefore must solve for t_2 where $P(t) = v_2$ and t_{proj} for P(t) = the projected point on the edge. If t_{proj} for the projected point is greater than $t_1 = 0$ and less than t_2, then t_{proj} lies within the edges of the line. We then find the distance from the projected point onto the edge to our point p. If this distance is less than the stored minimum distance, then our new closest point becomes the projected point onto the edge and the new minimum distance is the distance from our projected point to p.

6.1.6 Vertex Projected Point

If the projected point does not lie on the boundaries of the face or an edge, then the closest point from p to the shape is a vertex the face. Therefore, for each of the vertices of the face, we take the distance from our point p to each of the vertices. If the point to point distance is less than our stored minimum distance, then we set the vertex as the closest point.

6.1.7 Summary of nearest point to primitive

Finding the closest point from the point cloud to model is a crucial part of this algorithm. In summary, for each point, we project every point from the point cloud selection onto each of the faces, the edges, and vertex. If the projected point is within the bounds of the faces or the edge and the distance from the projected point to our point p is less than our globally stored minimum, then it becomes the new closest point.

6.1.8 Finding the Best Fit Transformation

Once we have the point to point matching we can find the transformation matrix from model set p' to our dataset p. This gives us an over-constrained system, but we can solve it using a least squares approximation, which minimizes the error. We need to find a transformation matrix T such that

$$min \sum (p - Tp')^2$$

We can solve this minimization using singular value decomposition [9]. Given an initial transformation from p' to p, we can rerun the algorithm, finding a new intersection point from p to our transformed mesh, and find a new transformation, reducing the error on each iteration.

In our equation where T is our transform matrix, P is our dataset matrix, and M is our model set matrix:

$$TM = P$$

$$\text{Solving for } T\text{: } T = PM^{-1}$$

Because it is an over-constrained equation (both P and M are nx4 matrices, where n is the number of selected points) and finding the inverse of M is difficult, we will instead use the SVD of the model matrix M where $M = VDU$ where D is the diagonal matrix. We can then calculate the inverse of the model matrix M:

$$M^{-1} = VD^{-1}U^T$$

We can now plug M^{-1} to solve for our transform matrix $T = PM^{-1}$. This transform matrix T can then be applied to the geometry, where if the geometry currently has the transform G, then the new transform matrix G' will be

$$G' = TG$$

6.1.9 ICP Optimizations

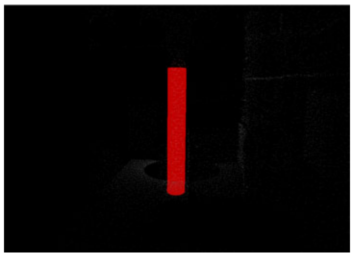
One potential optimization that we explored was weighting the point pairs based on the compatibility of normals given by

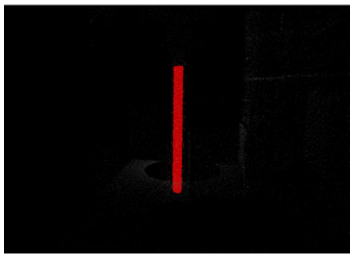
$$Weight = n1 * n2 \text{ [14]}$$

However, because the normals generated from PCA was not always reliable due to the high curvature of the columns, this approach was discarded. In addition, the initial fitting was often sufficient and ICP performed well enough that normal based weighting was unnecessary. See Appendix A for more details.

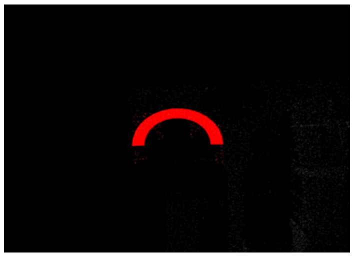
Instead, we extended ICP by weighting the point pairs based on Euclidean distances where the weight was expressed as:

(a) Before ICP- points are selected


(b) Initial fitting of shape


(c) After ICP fitting

(d) Before ICP- points are selected


(e) Initial fitting of shape

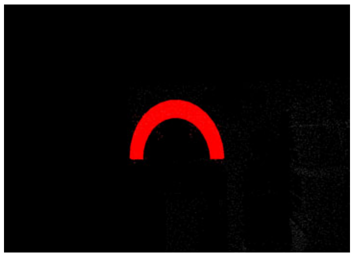

(f) After ICP fitting

Figure 6-1: Results of ICP Algorithm

$$Weight = 1 - Dist(p_1, p_2)/Dist_{max} \text{ [14]}$$

This weighting term helps ICP favor point pairs that have closer Euclidean distances, both helping to remove outliers and converge quicker.

Finally, we also implemented rejection of outlier pairs. The point pairs whose Euclidean distance was greater than .9 of the max distance was then discarded, signifying that these points were outliers. However, if the mean distance was greater than .75 of the max distance, then these points were retained. If this was the case, this means that the mean is closer to the max distance, and there may be no outliers.

6.2 Removing Shear with Polar Decomposition

6.2.1 Motivation

Because we are using ICP instead of a constrained registration, our geometry is subject to many different transformations, including shear. However, with architectural buildings, objects are often axis-aligned and rarely are sheared. Although this may suggest that we should constrain ICP, there are potentially cases, especially in cathedrals, where more unconventional architectural pieces occur, and that shear may prove useful. As an alternative, we create the option for the user to manually remove shear from a geometric object.

In general, removing shear is not a trivial problem, as it requires breaking the transformation matrix of an object into all of its individual transformations (shear, rotation, translation, transformation). Because our geometry has likely already undergone many transformations, this makes deconstructing the transformation a more challenging. Luckily, a quick technique called polar decomposition allows us to deconstruct the matrix into its basic transformations. However, the results of polar decomposition are not unique, as many different combinations of scale, rotations, and translations could result in the final transform. Therefore, polar decomposition does not always yield perfect results, but is still a useful tool for the user to have.

6.2.2 Implementation

Before performing the Polar Decomposition [18], we first extract the translation matrix T from a transformation matrix M. We then perform the polar decomposition, which factors a 4x4 homogeneous matrix M as

$$M = QS$$

Where Q is the best possible rotation and S is the stretch, or scale, matrix. We determine this by averaging the matrix with its inverse transpose until it converges. We set

$$\begin{gathered} Q_0 = M \\ \text{then } Q_i + 1 = (Q_i + Q_i - T) \\ \text{until } Q_i + 1 - Q_i = 0 \end{gathered}$$

As Shoemake [18] points out, this is essentially a Newton algorithm for the square root of I and converges quadratically when Q_i is orthogonal.

We can then factor the stretch matrix as

$$S = UKU^T$$

where U is the rotation matrix and K is diagonal and positive. We determine U and K by taking the SVD of the matrix. The paper suggests taking using a Takagi factorization to find the scale matrix. However, since SVD is quicker, and S is a positive semi-definite matrix, taking the SVD should be sufficient in extracting the scale. We then take K as the our best guess of a scale matrix.

To remove the shear, we then replace the original matrix M with our new matrix M' where

$$M' = TQK$$

where T was our initial translation matrix, Q was the best possible rotation matrix factored by the polar decomposition step, and K is the best guess scale.

For the most part, using polar decomposition is useful in generally removing shear. However, because this decomposition is not unique and there are many different rotations for which U could represent, this techniqe does not always produce desirable results.

6.3 Alignment

6.3.1 Motivation

The initial placement of the geometry to our selection assumes an axis aligned geometry since it takes the extrema of the point cloud selection and scales the unit-sized geometry appropriately. In most cases, this is a reasonable assumption, as pointed out by the work in SIGGRAPH 2010 done by Nan, et al [10]. However, there are many cases in which the geometry is not axis-aligned, such as diagonal walls. Therefore, with the alignment tool, the user can realign geometry as necessary, by selecting points from the model that corresponds to points on the point cloud. Similar to ICP, the least squares solution yields the appropriate transformation. However, in order for this equation to be constrained, the user must select at least 8 corresponding points on the model to yield a reasonable geometric transform.

6.3.2 Single Point Selection

The current selection tool allowed for users to select a rectangular region of points or a geometry object. However, for the alignment tool, users needed the ability to select a single point in the point cloud, or a single point on the geometry. Therefore, the alignment tool was extended to handle single point selection.

For all of the projected points that are within a radius of 10 of the user's click, the distance from these points to the camera is calculated. Assuming the user intended to click the point that was closest to the camera, the point that is closest to the camera and has a projected point within a radius of 10 of the user's click is the chosen point.

For selecting a single point from the geometric model, we perform a similar algorithm as selecting the point in the point cloud. However, instead of projecting all of the points in the point cloud, we project the points of the geometries' vertices in the scene.

6.3.3 User Interface

The first iteration of this tool required the user to mark the data and model points precisely in the same order, and there was no visual indication of which points corresponded to each other. This made the process cumbersome and difficult. Therefore, when ported to the final tool, the user interface of the alignment tool was taken carefully into consideration. A map for the marked data points, a map for the marked model points, and a color map all help the user to keep track of which points correspond to each other.

We can see in Figure 6-2 where our initialization failed because the wall is diagonal and not axis aligned. The user has selected corresponding points in the data and on the model, which are signified by the matching colors. After selecting the points and pressing the Align Shape button, the geometry is now transformed appropriately.

Alternatively, to deal with non-axis aligned cases, the user can also manually rotate the object, whether it is through direct mouse manipulation or by specifying the amount to rotate and re-scale. However, allowing the user to specify correspondences

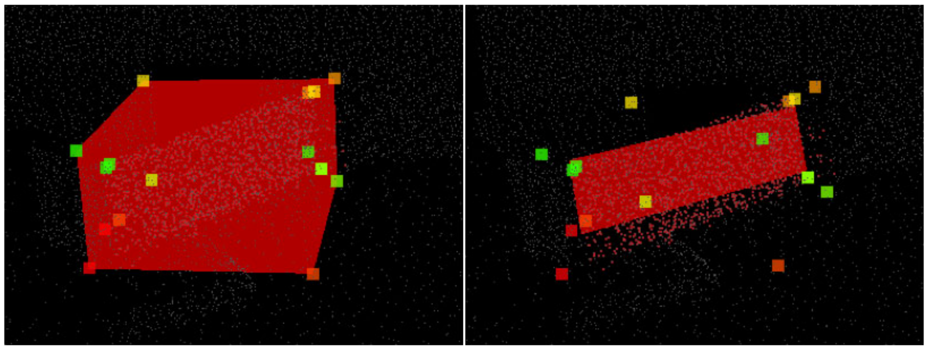

(a) Initial Fitting- Before alignment (b) After alignment

Figure 6-2: Results of Alignment Algorithm

can be faster and more precise, depending on the architectural element and how much the initialization was incorrect.

6.3.4 Algorithm

Similar to ICP, we apply a least squares solution to solving this problem. For each point m selected in the model and each point d selected in the point cloud, we find the best 4x4 transformation matrix T such that

$$d_i = T * m_i$$

where i is the number of user annotated points. Note that the user must specify at least four points for the equation to be solvable.

$$\begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ \vdots \\ d_i \end{bmatrix} = T \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ \vdots \\ m_i \end{bmatrix}$$

Chapter 7

Adjacency

To evaluate the structural stability of a model, Whiting's program requires that each of the geometric blocks have adjacent interfaces. After the geometry has been snapped to the point cloud, there is no guarantee that the geometry is adjacent. We provide an extra tool that enables the user, as a post-processing step, to guarantee that the geometric primitives are adjacent. By allowing the user to select faces, we perform another least squares fitting to snap geometries together.

7.1 Motivation

To evaluate the structural stability of an object, the blocks must be adjacent so that the tension at each of the interfaces may be calculated. Whiting's program requires the input model to already have these adjacencies specified. Unfortunately, ICP only handles the snapping of the geometric primitives to the point cloud, and not the snapping of one geometry to another. Therefore, as a post-processing step, users select two faces and designate that they should be adjacent. Projecting the points from one geometry to another and doing a least squares fitting snaps the geometries together.

7.2 Face Selection

The selection tool only previously handled point and geometry selection, so we extended it to handle selection of faces of a geometric primitive. In summary, for the point clicked, for each face, project each of the vertices to the screen. From the clicked point, draw a ray from the clicked point to (0,0). We need to see how many times this ray intersects the shape boundaries. If there is an even number of intersections, then it is outside of the shape. Conversely, if there are an odd number of intersections, then it is inside of the shape.

For each pair of points bounding the shape, we calculate the line intersection between the line and the ray we created. The un-parameterized equation of the ray going through (px,py) to (0,0) is

$$y = \frac{py}{px} * x$$

The un-parameterized equation of a line going between two points along the line of the shape from (x0,y0) to (x1,y1)

$$y = \frac{y1-y0}{x1-x0} * (x - x0) + y0$$

Combining the two equations to solve for the point of intersection we have:

$$x = \frac{[-(y1-y0)/(x1-x0)*x0+y0]}{[py/px-(y1-y0)/(x1-x0)]}$$

After finding the x-y intersection, we plug it into the parameterized equation:

$$P(t) = p0 + t * (p1 - p0)$$

where p0 = (x0, y0) and p1 = (x1, y1). If $0 \leq t \leq 1$ for the intersection, then we have intersected the line. If the number of intersections are even, then the point lies outside of the face. On the other hand, if the number of intersections is odd, then the point lies inside the face and the face is hit. After calculating all of the faces that were hit, for each of the hit points, we find the distance between the vertex and the camera. The face that has the smallest distance between its hit point and the camera ends up being the selected face.

7.3 User Interface

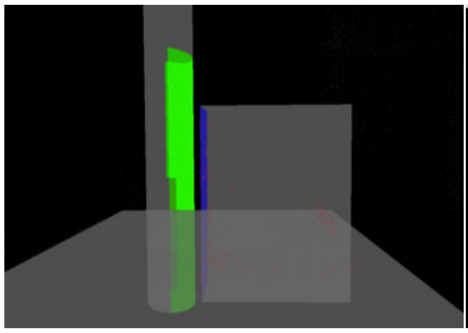
With face selection, there are two modes. Shift+f toggles between the two possible face selections. The blue face indicates the face of the geometry that will be transformed. The green face indicates the face of the geometry that stays static. The user interface also allows the user to add multiple faces or remove multiple faces from the selection.

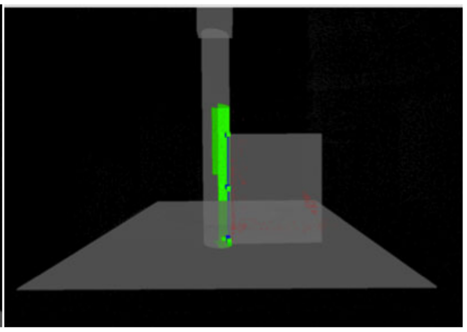
7.4 Algorithm

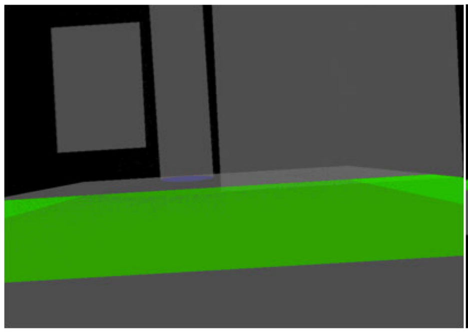
Once we determine the face of the geometry that needs to be made adjacent to the chosen geometry, we project the points of the face from the geometry that is to be transformed onto the surface of the other geometry. We run the closest point algorithm to find the closest point p' for every p on the selected geometry. Once we have the closest point, we solve the least squares problem. For each of the points p on the selected face, the closest p' is the new target point. For every point p not the on the selected face, the target point is p. We then solve this least squares problem using the SVD. Again, we solve for a 4 x 4 transformation matrix T where

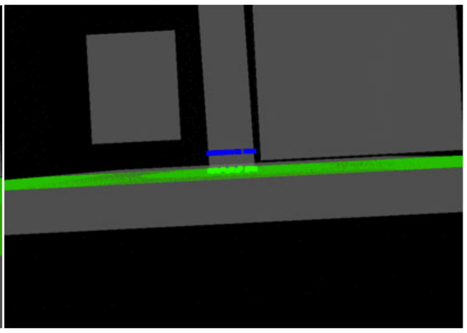
$$\begin{bmatrix} p'_1 \\ p'_2 \\ p'_3 \\ p'_4 \\ \vdots \\ p_{12} \\ p_{13} \\ p_{14} \\ \vdots \\ p_i \end{bmatrix} = T \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ \vdots \\ p_{12} \\ p_{13} \\ p_{14} \\ \vdots \\ p_i \end{bmatrix}$$

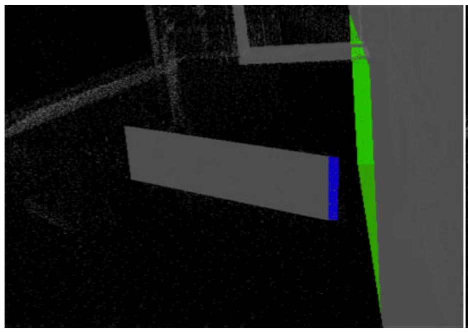
While this method is much simpler than relying on ICP to create adjacent geome-


(a) Shapes not adjacent-faces selected


(b) After running adjacency alg


(c) Shapes not adjacent-faces selected


(d) After running adjacency alg


(e) Shapes not adjacent-faces selected

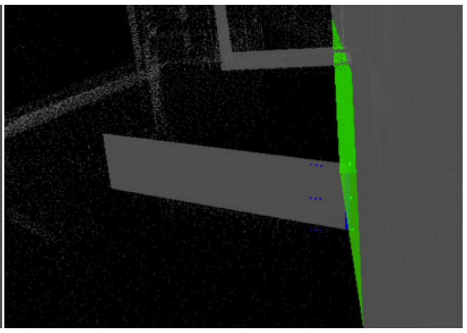

(f) After running adjacency alg

Figure 7-1: Results of Adjacency Algorithm

tries, specifying adjacent faces is still yet another step for the user. In the future, automating this step would be useful, potentially by finding the closest face to a designated face and then automatically deciding that they should be adjacent. However, there are many edge cases in which this approach would not be reasonable. For example, two arches are connected to the same column, we want the faces of each of the arches to be adjacent to the column, but not necessarily to each other.

Chapter 8

Results

The main goal of this system is to provide volumetric input into Whiting's program and it was tested by reconstructing portions of the cathedral; the user geometrically models the structure against the point cloud template. By successfully importing this model into Whiting's system and evaluating the structure's stability, we validate the system's ability to create feasible 3D models that are compatible with Whiting's program.

8.1 Models

The following two examples, Figure 8-1 and Figure 8-2 show two of models that were built. Figure 8-1 took about thirty minutes to construct, including the step of making the shapes adjacent. This model features nine geometric primitives, including arches, spandrels, cylinders, and cubes. Figure 8-2 took about an hour and a half to construct and took great advantage of the copy and paste feature. It features groinvault arches, cylinders, and cubes.

8.2 Discussion

It is hard to evaluate the correctness of the model because the system requires the user to make many assumptions about the internal structure of the building. In many

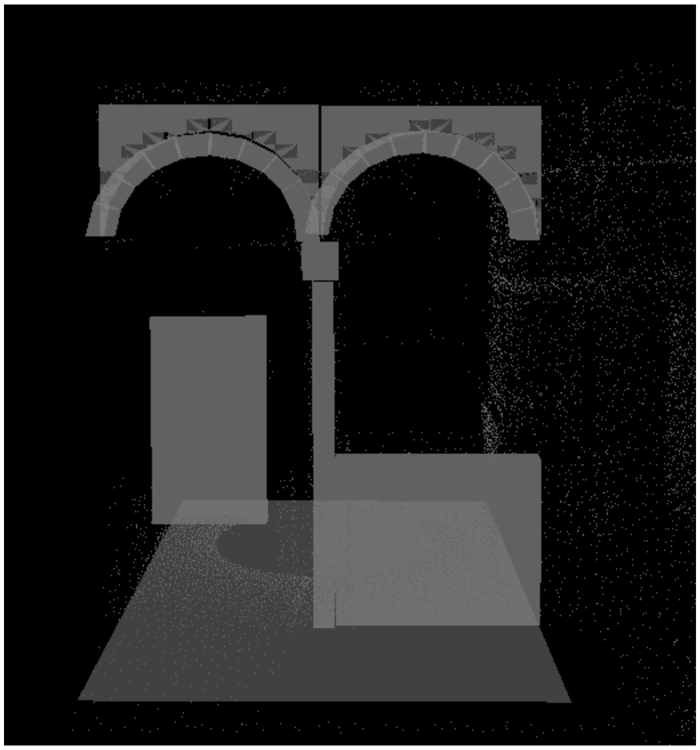
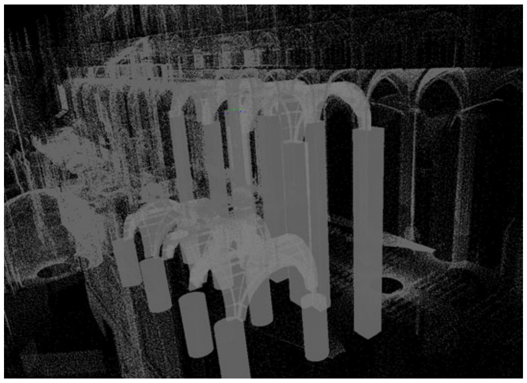

(a) Estimated time to construct: 30 minutes

Figure 8-1: Example Model 1


(a) Estimated time to construct: 1.5 hours

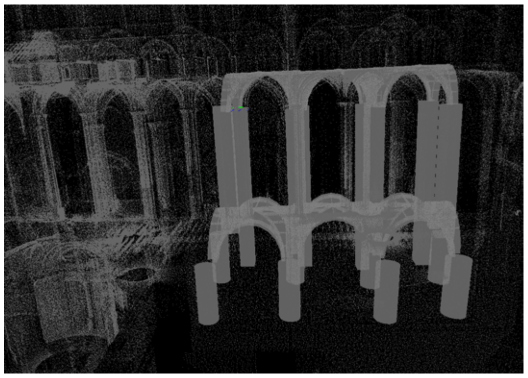

(b) Same model as 8-2a but from a different perspective

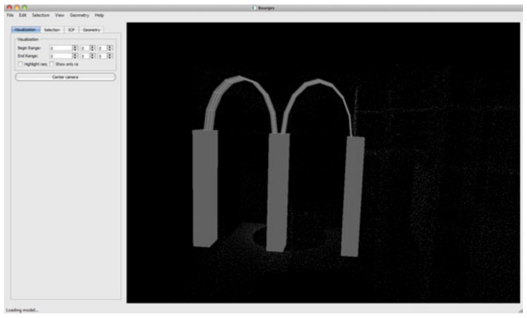
Figure 8-2: Example Model 2

cases, the system forces the user to discretize an ornate feature of the cathedral, such as the decorated column as a simple cylinder. We could hypothetically compare the error results in ICP using the distance between the matched points. However, this still assumes correctness in the user's assumptions about the building. In addition, the point cloud does not capture the depth of walls, so the user must approximate the thickness. Procedurally modelling the cathedral suffered from a similar problem in that it was hard to evaluate the correctness of the structure with regards to the current state of the cathedral. Therefore, we must rely upon the human eye to match the point cloud with the geometric primitives.

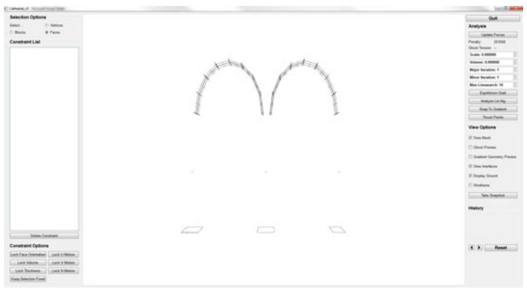
8.3 Force Evaluation

Once the cathedral is fully classified as a system of geometry primitives, we then use force models to evaluate the cathedrals structural stability, as demonstrated by Whiting [19]. Using her system, we evaluate whether or not the force exhibited on existing geometries yields a stable structure or not. Being able to successfully import our geometric models into her system and evaluate the structural forces allows us to validate our system. Figure 8-3a shows two arches being constructed in our point cloud modelling program. Once it is exported into the appropriate format, it is then imported into Whiting's program so that the forces can be evaluated. Interfaces and adjacencies are correctly calculated as seen in Figure 8-3b. Once the interfaces are computed, we evaluate the forces on each of the blocks. Because the arches are thin, the structure is not very stable, having an initial force value of 253,560 as seen in Figure 8-3c. The most stable structure should have a force value of 0.

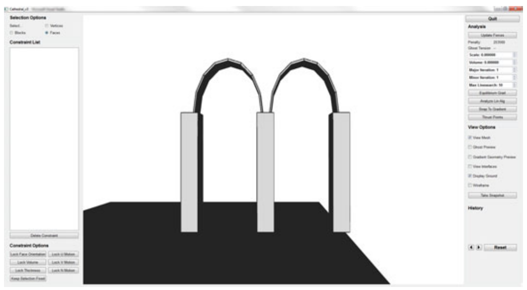
Whiting's system also suggests modifications to the existing structure to make it more stable. She calculates the gradients with finite differencing to minimize the tension on each of the blocks. After running an iteration of the gradient descent, as seen in Figure 8-3d, the structure is made more stable, having a force evaluation of 228,136. The end columns start getting slanted and the arch bases are made wider.



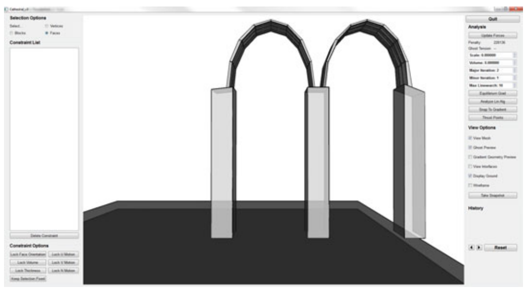
(a) Arch structure constructed in this system



(b) Adjacencies and interfaces interpreted correctly in Whiting's program



(c) Constructed Structure loaded into Whiting's program with forces calculated. Note the current structure is not very steady and has a force evaluation of 253560.



(d) Modifications to the arch after running an iteration of gradient descent. Notice the slanted columns and the thicker arch bases. The structure is more stable, with a force evaluation of 228136.

Figure 8-3: Evaluation of forces

Chapter 9

Conclusion

We present a system that allows users to perform a volumetric geometric reconstruction of a point cloud cathedral and evaluate the structural stability of the 3D model. Combining an extensive user interface with shape fitting algorithms, we can capture a volumetric snapshot of the building's current state and evaluate its structural stability.

9.1 Future Work

We could utilize many more geometric primitive types for more accurate modelling. Because Whiting's program supported force calculation with only six sided blocks, that limited the primitives we could use. Future work includes either generating geometric primitives made out of these simple blocks, or extending Whiting's work to handle curved structures with multiple faces. Other alternatives include creating a procedural modeling program that is specifically designed to create simple architectural structures out of these six-sided building blocks. Alternatively, as a post-processing step, this program, given any obj file, could discretize the mesh into blocks by using a mesh discretizing algorithm.

There are also extensions to what we can visualize with the force models. Once we have computed the force models on the collection of geometric primitives, we can visualize the components of the cathedral in various ways. For instance, weaker areas of the structure can be color coded, and specific data calculations can be displayed based on the users inputs.

Also, this system requires a bit of user involvement for the fitting. The problem is a non-trivial one, but increased automation of this system would greatly help to streamline the reconstruction process.

Allowing ICP to create shear ended up causing complications when creating adjacencies between individual blocks and blocks with the ground, specifically when porting between the two systems. In particular, the constraint that the base blocks must be parallel to the floor caused problems. Implementing a constrained ICP that restricts shear, along with providing the existing ICP implementation would aid the user in the reconstruction process.

9.2 Summary

This work presents an interactive user tool that visualizes point cloud data of a cathedral and provides tools for users to fit volumetric geometric primitives to the point cloud in order to analyze its structural stability. A grid box approach and dynamic loading helps optimize visualization of the dense point cloud. The extensive user interface allows for user selection of points, geometries, and faces, as well as tools for transforming these geometries. With the help of the iterative closest point algorithm and least squares fitting, the user can snap geometries to the point cloud, snap geometries to selected points, and snap adjacent geometries. Finally, once the user is done modeling a section, they can export the model into an obj file and import it into Whiting's system to evaluate the structural stability. By providing a point cloud as a template, this system allows users to create a more accurate 3D volumetric reconstruction of architectural buildings so that its stability can be evaluated and structural modifications may be suggested.

Appendix A

Appendix A

A.1 Normal Calculation

A.1.1 Summary

Knowing the normals of the points has some potentially useful features, especially in the improvement of ICP. We can compute the normals using Principal Component Analysis [4] [8] [5], or PCA.

PCA can be summarized in the following steps:

1. Find the nearest neighbors
2. Calculate the covariance matrix
3. Compute the eigenvalues of the covariance matrix
4. The eigenvector associated with the smallest eigenvalue is the normal

A.1.2 PCA Implementation

In order to perform PCA to find the normal of a point p, we first need to know the nearest neighbors of p. We take advantage of the grid acceleration structure and limit our search of nearest neighbors to the points within p's own grid box. Using the C++

vector built-in sort based on distances to the point, sped things up exponentially over our naive sort implementation.

Once we know the nearest neighbors, we need to calculate the covariance matrix of the set of k nearest neighbors. We first calculate the centroid $p*$ of the nearest neighbors. For each point x in the nearest neighbors set, we calculate the covariance matrix as

$$P = 1/k * \sum(x_k - p^*)^T(x_k - p^*)$$

where k is the number of nearest neighbors and p^* is the local data centroid

$$p^* = 1/k \sum x_j.$$

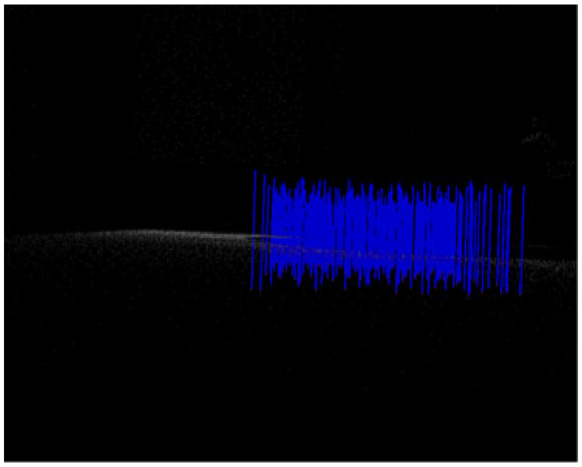
We can then compute its approximate surface normal as the eigenvector associated with the smallest eigenvalue of the symmetric positive semi-definite matrix, or covariance matrix, that we found above. We calculate the eigenvalues by finding the SVD. The eigenvector corresponding to the smallest eigenvalue is the approximate surface normal.

A.1.3 Results

This method of calculating the normals only yielded marginal results. In cases where the point cloud was more planar, this technique worked very well. On the other hand, in cases of high curvature or variable density of points, this technique failed.

For example, in specialized cases, like the decorated column in Figure A-2, the calculation of normals failed, due to the highly ornamental nature of the column and the high curvature of the column. Although the column appears to be merely a simple cylinder, if viewed from above, the user can see that it is much more ornate and contains a high amount of curvature.

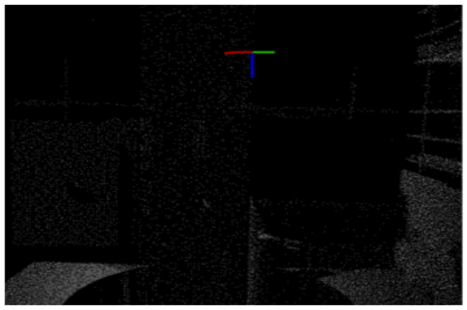
The problem of calculating normals with a high amount of curvature is not a new one and has been tackled by many people. The number of nearest neighbors and the density of the point cloud has a strong influence on the correctness of the normal. Mitra [8] suggests varying the number of nearest neighbors based on the



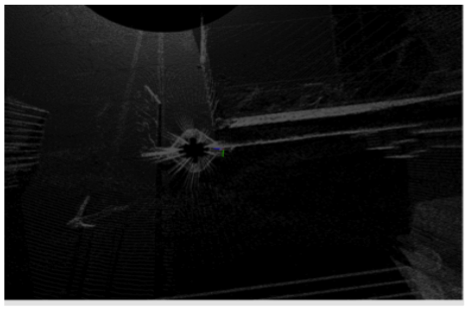
(a) Render of the normals (in blue) on the surface of the plane. In this case, PCA behaves quite well.

Figure A-1: Rendering Normals of Plane

calculated curvature of the point cloud. We experimented with varying the number of nearest neighbors and increasing the density of points, but neither yielded consistent or suficient results. Therefore, because the results of PCA were not consistent, we chose not to include the normals in our calculation of ICP. Because the initial snapping for ICP does a reasonably good job, the inclusion of normals in the optimization of ICP was unnecessary.



(a) Incorrect Normal on Column. The normal is in blue, and the red and green lines are the orthogonal unit eigenvectors. As we can see, the green line should be the correct normal, so PCA fails.



(b) View of column from above. We can see that the column is not simply a cylinder; there is a lot of decoration and ornation, which makes the estimation of normals difficult due to the high curvature.

Figure A-2: Incorrect Normals Example

Bibliography

[1] Autodesk shape fitting. "http://labs.autodesk.com/files/8001_8100/8081/file_8081.pdf".

[2] Dror Aiger, Niloy Mitra, and Daniel Cohen-Or. 4-point congruent sets for robust pairwise surface registration. *ACM Transactions on Graphics*, 2008. http://graphics.stanford.edu/ niloy/research/fpcs/paper_docs/fpcs_sig_08_large.pdf.

[3] Pierre Alliez, Laurent Saboret, and Gal Guennebaud. Surface reconstruction from point sets. CGAL*, Computational Geometry Algorithms Library.* "http://www.cgal.org/Manual/latest/doc_html/cgal_manual/Surface _reconstruction_points_3/Chapter_main.html".

[4] David Belton. Improving and extending the information on principal component analysis for local neighborhoods in 3d point clouds. *ISPRS*, 2008. http://www.isprs.org/proceedings/XXXVII/congress/5_pdf/83.pdf.

[5] Edward Castillo. Point cloud segmentation via constrained nonlinear least squares surface normal estimates. 2009. ftp://ftp.math.ucla.edu/pub/camreport/cam09-104.pdf.

[6] Sergio Garcia. Fitting primitive shapes to point clouds for robotic grasping. http://www.nada.kth.se/utbildning/grukth/exjobb/rapportlistor/2009/ rapporter09/garcia_sergio_09131.pdf.

[7] P. Labatut, J.-P. Pons, and R. Keriven. Efficient multi-view reconstruction of large-scale scenes using interest points, delaunay triangulation and graph cuts. In *IEEE International Conference on Computer Vision*, Rio de Janeiro, Brazil, Oct 2007.

[8] Niloy Mitra and An Nguyen. Estimating surface normals in noisy point cloud data. *Symposium on Computational Geometry*, 2003. http://graphics.stanford.edu/ niloy/research/normal_est/paper_docs/ normal_estimation_socg_03.pdf.

[9] Bryan Morse. Camera calibration. http://morse.cs.byu.edu/750/lectures/lect04 /calibration.slides.printing.6.pdf.

[10] Liangliang Nan, Andrei Sharf, Hao Zhang, Daniel Cohen-Or, and Baoquan Chen. Smartboxes for interactive urban reconstruction. *ACM Trans. Graph.*, 29:93:1–93:10, July 2010.

[11] YIH Parish and Pascal Mueller. Procedural modeling of cities. *ACM Transactions on Graphics*, 2001. http://www.cs.purdue.edu/homes/aliaga/cs197-10/papers/p_cities.pdf.

[12] Ravi Ramamoorthi and James Arvo. Creating generative models from range images. *ACM Transactions on Graphics*, 1999. http://www.cs.berkeley.edu/ ravir/papers/invgen/paper.pdf.

[13] Pere Roca. Studies on the structure of gothic cathedrals. *Historical Constructions*, 2001. "http://www.civil.uminho.pt/masonry/Publications/Historical".

[14] Syzmon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. *Third International Conference on 3D Digital Imaging and Modeling*, 2001. http://www.cs.princeton.edu/ smr/papers/fasticp/.

[15] Syzmon Rusinkiewicz and Mark Levoy. Qsplat: A multiresolution point rendering system for large meshes. *ACM Transactions on Graphics*, 2010. http://graphics.stanford.edu/papers/qsplat/qsplat_paper.pdf.

[16] Ruwen Schnabel. *Efficient Point-Cloud Processing with Primitive Shapes*. Dissertation, Universität Bonn, December 2010.

[17] David C. Schneider and Peter Eisert. Algorithms for automatic and robust registration of 3d head scans. *Journal of Virtual Reality and Broadcasting*, October 2010. "http://iphome.hhi.de/eisert/papers/jvrb2010.pdf".

[18] Ken Shoemake and Tom Duff. Matrix animation and polar decomposition. *Proceedings of the conference on Graphics interface*, 1992. http://research.cs.wisc.edu/graphics/Courses/838-s2002/Papers/polar-decomp.pdf.

[19] Emily Whiting, John Ochsendorf, and Frédo Durand. Procedural modeling of structurally-sound masonry buildings. *ACM Transactions on Graphics*, 28(5):112, 2009.

[20] Changchang Wu, Sameer Agarwal, Brian Curless, and Steven M. Seitz. Schematic surface reconstruction. *CVPR*, 2012. http://www.cs.washington.edu/homes/ccwu/sweep.pdf.