University of Cape Town

Undergraduate Thesis

Creating a Boundary Representation Model from Indoor Laser Scans

Author:

Tim Marsh

Supervisor:

Dr. George Sithole

A thesis submitted in partial fulfillment of the requirements for a B.Sc. Degree in Geomatics

in the

Department of Geomatics

November 5, 2015

vi veri veniversum vivus vicii

Acknowledgments

K THX BYE

Plagiarism Declaration

1. I know that plagiarism means taking and using the ideas, writings, works or inventions of another as if they were one's own. I know that plagiarism not only includes verbatim copying, but also the extensive use of another person's ideas without proper acknowledgment (which includes the proper use of quotation marks). I know that plagiarism covers this sort of use of material found in textual sources and from the Internet.

2. I acknowledge and understand that plagiarism is wrong.

3. I understand that my research must be accurately referenced. I have followed the rules and conventions concerning referencing, citation and the use of quotations as set out in the Departmental Guide.

4. This assignment is my own work, or my group's own unique group assignment. I acknowledge that copying someone else's assignment, or part of it, is wrong, and that submitting identical work to others constitutes a form of plagiarism.

5. I have not allowed, nor will I in the future allow, anyone to copy my work with the intention of passing it off as their own work.

Tim Marsh


\AM@currentdocname .png
.png

Abstract

With laser scanning becoming more prominent as a form of surveying, it becomes increasingly important for us to be able to process the resulting point clouds and make sense of them. This Report takes indoor laser scans and attempts to create boundary representations of these laser scans using machine learning processes and algorithms.

The need for boundary representations of rooms arises from engineering and architectural professions, where CAD models are needed of existing structures to facilitate future work on the buildings. Currently the best method for creating CAD models is by hand is CAD software but this is slow and potentially inaccurate. This Project aim to solve this issue of speed and accuracy.

Contents

List of Figures

Chapter 1

Introduction

1.1 Subject of the Report

Point cloud resulting from laser scans are highly unstructured entities, there is no information in the point cloud other that point locations and often times colour and intensity. None of this says anything about the indoor area scanned. Models need to be created from these point clouds but with not much more information other than the locations of the points it is not easy. This is the where the field of Machine learning comes in.

This Report sets out to use Machine learning ideas and Algorithms to take laser scanned point clouds of indoor environments and create boundary representations of them as accurately as possible.

1.2 Background to The Investigation

3D representations of buildings vary largely in terms of how rigorously the models are structured. There are two extremes to it, the first is highly structured models, made in CAD software.

The other end of the scale is 3D points clouds created by laser scanners that are entirely unstructured.

For taking surveying related measurements having just a point cloud works fine, you can pick out specific points and do measurements across the points. But when 3D models are used for more complex applications such as an as in engineering and architecture, just having a point cloud will not cut it. These applications need structure that a laser scanned point cloud cannot produce.

So it becomes necessary to create structure in these points clouds. This is usually done by turning the point cloud into a 3D model that engineers and architects can then work on and use.

1.3 Objectives

The primary objective of this project is to be able to create boundary representation models from indoor laser scans. This will be achieved by answering a few questions:

- What Is the most accurate and efficient method to estimate normals for every point in a point cloud?
- What is the most efficient method to segment a point cloud?
- What are the best segmentation parameters to use for an indoor environment?
- What is the most effective way to filter out unwanted segments after segmentation?
- What is the best method to fit planes segments?

- How can planar segments of a point cloud be turned into boundary representations of their entire extent?

1.4 Implementation

The programming will be done in C++ as it is a popular, well documented language and supports many large and extensive libraries such as Point Cloud Library (PCL) and Eigen.

Using C++ and PCL an algorithm to complete these tasks with as little user input as possible will be created.

1.5 Scope and limitations

This Report uses a room in the Menzies building at the University of Cape Town as there are many available scans of the room.

The algorithm is not limited by noise in the point cloud but by the complexity of the room that is scanned.

1.6 Outcomes

The outcome of this paper will be to create an automatic system that creates 3D boundary representation models as accurately as possible from indoor laser scans.

Chapter 2

Literature Review

2.1 Introduction

Segmentation and classification of indoor scans is an existing problem, a lot of the processes used in the newly created algorithms are not new. They are simply tweaked in such a way so as to produce the result required for an indoor environment.

This section looks at new and old publications of the processes and algorithms associated with creating the program this thesis set out to make.

As well as other papers and articles that have created similar programs and systems.

2.2 Algorithms

2.2.1 Segmentation

For automatic processing of point clouds the segmentation of the point cloud is one of the most important processes. it is important that the segmentation

of the point cloud is correct and the method is the best for the use.

Principal Components Analysis

Principal Components Analysis (PCA), is a statistical technique that transforms a number of possibly correlated variables into a smaller number of uncorrelated variables called Principal components. the number of principal components is less than or equal to the dimensions of the data set, so in the case of point clouds 3.

The first of the Principal components represents the largest variability in the data set, with the variability decreasing as you go along. essentially resulting in the 3rd Principal component being the normal to the surface (**?**).

Edge-based Segmentation

Edge-based segmentation has two main sections to it; first to detect edges in the point cloud, and secondly to group all points contained within the edges as one segment.

The algorithm used, however, does not use Edge-based techniques further reading on the topic can be found with **?** and **?**.

Region-based Segmentation

Region-based segmentation algorithms look for areas that fit into a certain criteria and group them together as a single region.

Region growing for 3D point clouds is an adapted algorithm that was originally created by **?** for the segmentation of intensity images.

Region growing needs two things; first, seed points based on their curvature values, and secondly criteria in which to extend these points into regions.

Common criteria include, Colour and normal direction.

There are several methods for doing these two things as described by **?**.

? proposed the idea of adding a smoothness constraint which finds smoothly connected areas in the point cloud. The method they propose requires a small number of parameters which provide a trade off between over and under segmentation. It is this refined algorithm that point cloud library uses in as its default region growing.

2.2.2 Surface Extraction

Random sample consensus

Random sample consensus (RANSAC) is an iterative method used to estimate the model coefficients of a set of observed data that contains outliers. RANSAC was developed by **?** in **?** as a way of solving the Location Determination Problem in photogrammetry and computer vision applications.

RANSAC achieves its goal by iteratively selecting a random subset of the original data, this subset of the original data are considered inliers. A model is then fitted to the inliers (A plane or line ect). The remainder of the data set is then compared to the model if a point fits well with the model it is considered a hypothetical inlier. The model is then re-estimated from the hypothetical inliers. The model is then evaluated by the error relative to the inliers of the model.

This process is repeated a fixed number of times, each time either rejecting the model if there are too few inliers, or replacing the last saved model if the error is lower.

Least Squares Plane fitting

Another method of fitting a plane to a set of 3-dimensional points is through least squares. Least squares itself was created in 1805 by a French mathematician Adrien-Marie Legendre.

Least squares is used to fit a plane to a point to get the normal of the point by taking a region around the point and fitting the plane to all those points. The process of fitting a plane to a set of points is described by **?**.

Occlusion in Cluttered Environments

In **? ?** Presented a robust approach for reconstructing the main architectural structure of complex indoor environments given laser scans of the rooms. within the paper they speak about how large vertical objects can be mistaken for a wall when they are in fact a cabinet or locker or something of that nature.

Walls, by definition, cover the full vertical extent of a room. But when working wit laser scans this is not something that you can enforce because there is often clutter in the way so the full extent of the wall is not seen. So to deal with this **?** employed a simple visibility test to determine if one segment was 'in-front' of another. They cast rays from the scan center through the OBB (oriented bounding box) and set out to determine if those rays intersected any other segment. If the rays do intersect another segment the first segment obviously lies in-front of the other.

2.2.3 Model Generation

Use of Primitives in Uncluttered Environments

? from Video and Image Processing Lab in the University of California,

Berkeley, wrote a paper on planar 3D modeling of building interiors from point clouds generated by laser scanners. The scans that they used were cleaned out of noise and were not in cluttered environments.

The method that **?** proposed did not use any specific segmentation algorithm, they instead calculated the normals for each point and classified points based on the direction of their normal. From there they fit planes to represent walls. They also attempt to create stairs by looking at all the points labeled as 'remaining' and attempt to fit a predefined staircase model to the points. Edges are made up from veracities and faces are made from

This idea could be extended to any item where a primitive can be created.

2.3 Optimization

2.3.1 Data storage

Boundary representation model

Boundary representation models, or B-rep for short, can be considered as an extension to the wireframe model. The advantage of a B-rep model over a wireframe model is that a B-rep model has an interior and an exterior, essentially a B-rep stores topological information. B-rep models have 3 main topological items: faces, edges and vertices's. Edges are made up of veracities and faces are made up of edges. Boundary representations models can also save and display curved edges.

Point Cloud Data file format

The PCD file format is a file format was created by Point Cloud Library (PCL) for the storage of point cloud files used with their library. PCD is

not the only format for storage of point clouds but has been created to offer greater flexibility and speed to reading and writing files.

There are two different storage modes of PCD, ASCII form, essentially just plain text with each point on new line separated with a space. And binary form where the data written to the file ia a complete copy of the pcl::PointCloud.points array/vector.

The ASCII style of storage makes the cloud usable in other point cloud applications, and is readable and editable by a user in simple text viewers.

The advantages of PCD over other file formats is:

- The ability to store and process organized point cloud datasets - this is of extreme importance for real time applications, and research areas such as augmented reality, robotics, etc
- binary *mmap/munmap* data types are the fastest possible way of loading and saving data to disk
- storing different data types (all primitives supported: char, short, int, float, double) allows the point cloud data to be flexible and efficient with respect to storage and processing.
- n-D histograms for feature descriptors - very important for 3D perception/computer vision applications.

Another advantage is that by controlling the file format, PCL can adapt it to best suit PCL libraries. (**?**)

K-d tree

In computer science trees are a widely used data structure or abstract data type. In the context of this research paper they are used for creating a search-able data structure to make searching for nearest neighbors faster.

A K-d tree is a space partitioning data structure used for structuring unorganized points. a K-d tree is a structure designed specifically for multi dimensional situations, hence the name k-dimensional tree, where k is the dimension of the search space. The advantage of this data type is that is can handle many different types of queries very efficiently (**?**).

Wavefront obj

Obj (or .OBJ) is a geometry definition file format first developed by Wavefront Technologies, used to store object composed of lines, polygons, free-form curves and surfaces.

Obj files don't not require a header making them easier to change, add to or remove from. Obj circumvents this need for a header by having a key at the beginning of every line, for example; "v" for veracities and "f" for a face.

2.3.2 Open Multi-Processing

OpenMP is an application programming interface (API) for writing multi-threaded applications. Certain functions in Point Cloud Library have support for multi-threading using OpenMP.

OpenMP is used in some of Point Cloud Libraries stock functions and allow multi-threading to speed up processing times. unfortunately it is not supported for the segmentation algorithms because race conditions prevent it from being correctly implemented.

The Normal estimation class in Point Cloud Library has an OpenMP replacement that is 100% compatible with the single-threaded function, no effort is required to run normal estimation on a point cloud up to 8 times faster depending on the number of cores the computer has available.

Chapter 3

Method

Points within a 3d point cloud are simply represented by their x, y, z Cartesian coordinates. This means that there is a possibility that there are two points with the same x, y, z coordinates that have been acquired at different times, this is assuming that the origin is the same. Comparing these points is not as easy as one would assume, they may occupy the same point in space, but cannot simply assume that they are the same.

We may get some extra data from a laser scanner, such as intensity and colour. But this doesn't really give us any information about the area surrounding these points and weather anything has changed.

Situations where points need to be compared for any reason require more information than can be provided by a laser scanner. So the idea of looking at each point individually now falls away. We need to start looking at the bigger picture of what the point cloud is telling us.

Machine learning is a subfield of computer science that evolved from pattern recognition and learning theory in artificial intelligence. it is in the field of machine learning that we start to look at the bigger picture of what our

point clouds are telling us. Machine learning is the study of creating algorithms that can learn from and make predictions about data.

The aim of this Thesis is, given an uncleaned, noisy point cloud of a room to create an .obj file with lines representing all the edges of that room.

3.1 Downsampling point clouds

Downsampling a point cloud is necessary when the resolution set on the scanner is very high and the cloud has many redundant points in it. For instance a 1m x 1m section of floor could potentially have up to 500 000 points, this is a huge amount of data for a small section of floor. After downsampling to 1cm the number of points drops right down to around 10 000. This si still a lot but is much more manageable, and because this project doesn't work with fine detail downsampling the entire point cloud to a manageable size allows the processing time to go right down without the loss of any accuracy.

Downsampling in this project uses a voxelized grid approach. A voxel grid is essentially a regular grid in 3D space (think of a voxel grid as a set of tiny 3D boxes in space). This 3D grid is overlaid on the point cloud data, then within each voxel (each small box) all the points present are approximated with their average point. Figures **??** and **??** show the difference between a dense point cloud and a downsampled point cloud.

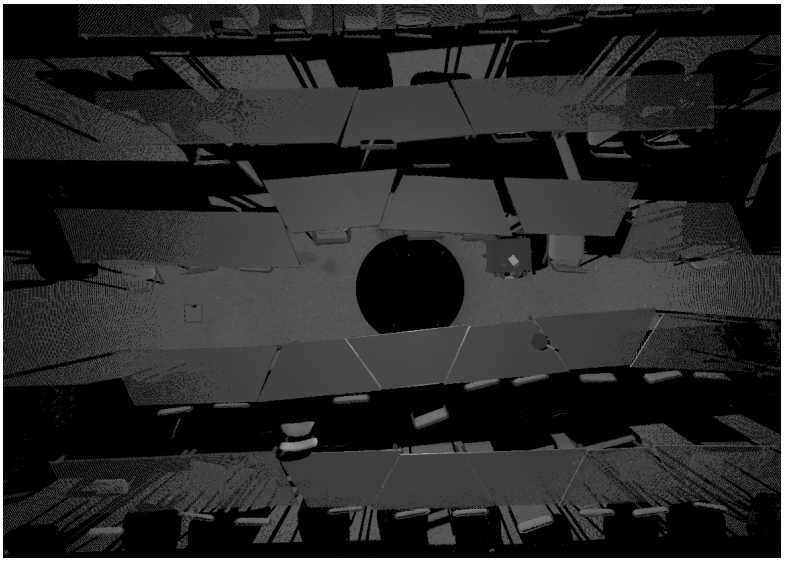


Figure 3.1: A point Cloud Before Voxel Downsampling

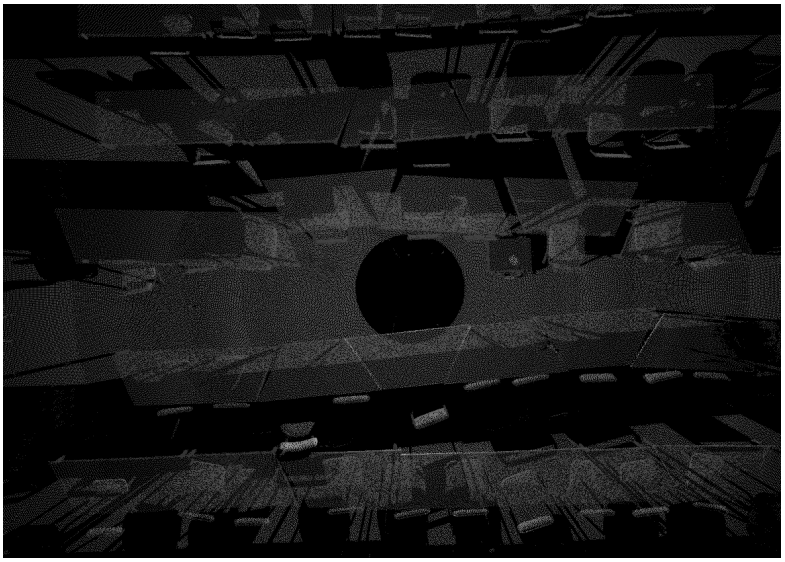


Figure 3.2: A Point Cloud After Voxel Downsampling to a 1cm Resolution

By default in this project all clouds that are read in are downsampled to a 1cm resolution before any processing is done on them.

3.2 Segmentation

3.2.1 Principal Components Analysis

A Principal Components Analysis (PCA) is the default method for estimating normals in Point Cloud Library. it is the fastest method because of PCL's multi-threading option for the function.

A principal components analysis can be thought of as fitting an n-dimensional ellipse to a set of data. Each axis of the ellipse represents a principal component. If an axis is large then the variance along that axis is large, and vice versa. When calculating the normal of a point set, the smallest axis is the axis with with the least variance and therefore represents the normal. This is easy to think of with a 2D disk of points, like in figure **??**. Its easy to see how the 3rd axis for these set of point will be coming out of the page, as it has to be orthogonal to the other two.

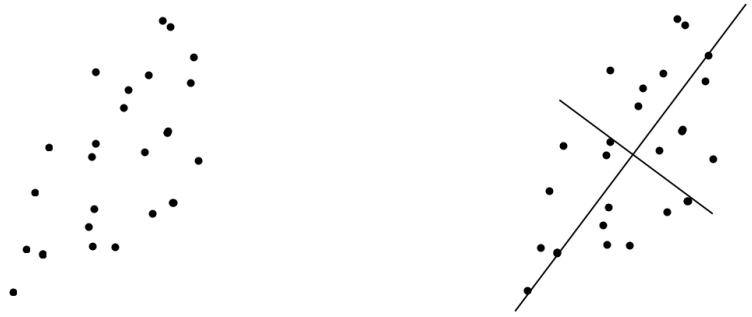


Figure 3.3: A 2D representation of a Principal Components Analysis

To fit this n-dimensional ellipse we compute the covariance matrix of the data set and calculate the eigenvalues and corresponding eigenvectors of this covariance matrix.

Calculating the Covariance Matrix of a given set of 3D points

The covariance matrix, C, is calculated for each point p_i as follows:

$$C = \frac{1}{k} \sum_{i=1}^{k} .(p_i - \bar{p}).(p_i - \bar{p})^T \tag{3.1}$$

Where k is the number of points in the neighborhood of point p_i, and $\bar{p}$ is the 3D centroid of the nearest neighbors.

Calculating Eigenvalues and Eigenvectors

Eigenvalues, λ_j, and eigenvectors, $\vec{v_j}$, are calculated on the covariance matrix, C, for a given point:

$$C\vec{v_j} = \lambda_j \vec{v_j} \tag{3.2}$$

After solving the system and getting results for λ_j and $\vec{v_j}$, the smallest eigenvalue and its corresponding eigenvector is are as the normal $\vec{n_i}$ for that point.

There is no mathematical method for determining the sign of the normal, its orientation as computed in the PCA is ambiguous, and not consistent over the whole point cloud.

The solution to this issue is simple. If the viewpoint is known, in the case of laser scans the scan center, then orientate all normals $\vec{n_i}$ towards the viewpoint. This whole process results in what is seen in figure **??**.

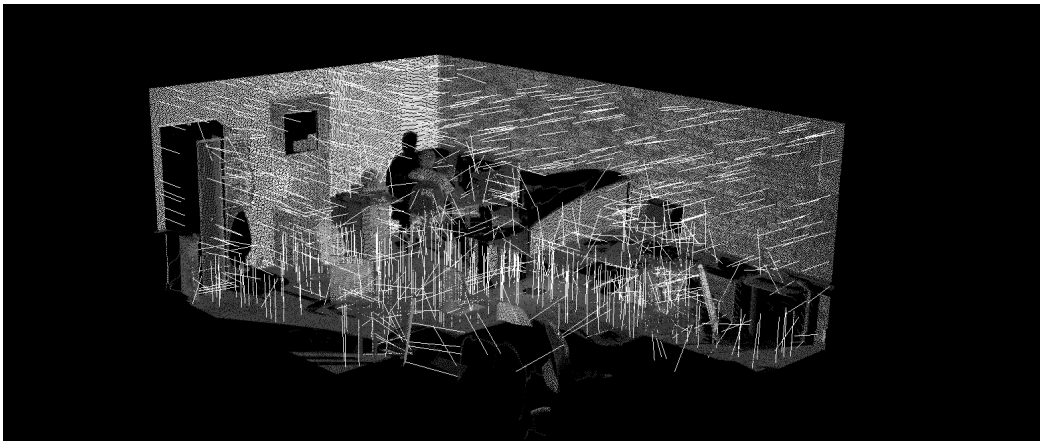


Figure 3.4: Normals calculated over a cut down section of a point cloud (the roof has been removed for ease of viewing)

3.2.2 Region Growing

A Region growing algorithm starts off by selecting seed points. This is done by calculating the curvature of each point then storing all the points in an array, sorted by their curvature value. The reason for this is because the point with the least curvature value is located in the flattest section of the point cloud. Now regions can begin to be grown using the flattest points in the cloud as seed points.

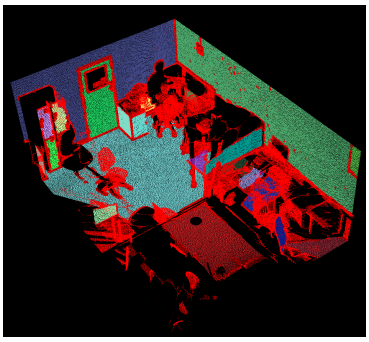
The first point in the sorted array is chosen as the first seed point:

- The nearest neighbors to this point are looked at and are either rejected or accepted into the region based on user defined criteria such as normal deviation and smoothness and curvature constraints.
- Once a point is accepted into the region, the process starts again based on that point.
- Once no more points are found for that particular seed, or the region

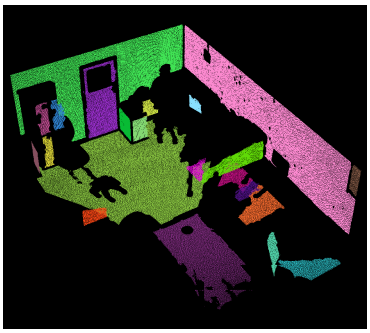
reached a specified maximum, the seed is removed from the set of seeds and the region is added to the global segment list.

This is repeated while the list of available points is not empty.

If after a seed point has been through the process and the number of points does not reach the user defined minimum size, the region removed from the cloud as unclassified. Unclassified points are removed from the cloud completely after the process is terminated.



(a) Result of Region Growing



(b) Unclassified points removed

Figure 3.5

The segments are coloured randomly to make them easier to distinguish for the user.

A pseudo-code algorithm for region growing can be found in appendix A.

3.3 Surface Extraction

3.3.1 Random Sample Consensus

Random Sample consensus (RANSAC) is used to estimate a normal to the segment by fitting a plane to it.

RANSAC achieves this goal by iteratively selecting a random subset of the original data, this subset of the original data is considered inliers. A model is fitted and the hypothesis that this is the correct model is then tested as follows:

- All other data is then tested against this model, if a point fits well it is considered a hypothetical inlier.
- The model is considered a reasonably good fit if enough points are considered inliers.
- The model is then re-estimated from all hypothetical inliers, as opposed to only the initial set.
- Then finally the model is evaluated by calculating the error of all the points relative to the model.

This process is repeated a fixed number of times, each time either rejecting the model if there are too few inliers, or replacing the last saved model if the error is lower.

Choosing number of iterations in RANSAC method

k (the number of iterations) required for a successful estimation within a certain probability can be calculated:

$$k = \frac{\log(1-p)}{\log(1-w^n)} \tag{3.3}$$

Where P(success) $= p$ and P(Selecting an inlier) $= w$ and $n =$ the points that are needed to create the initially estimated model(in 3D case of plane fitting $n = 3$)

w is usually not known but a rough estimate can be found.

Below is a 2D representation of RANSAC with a line.

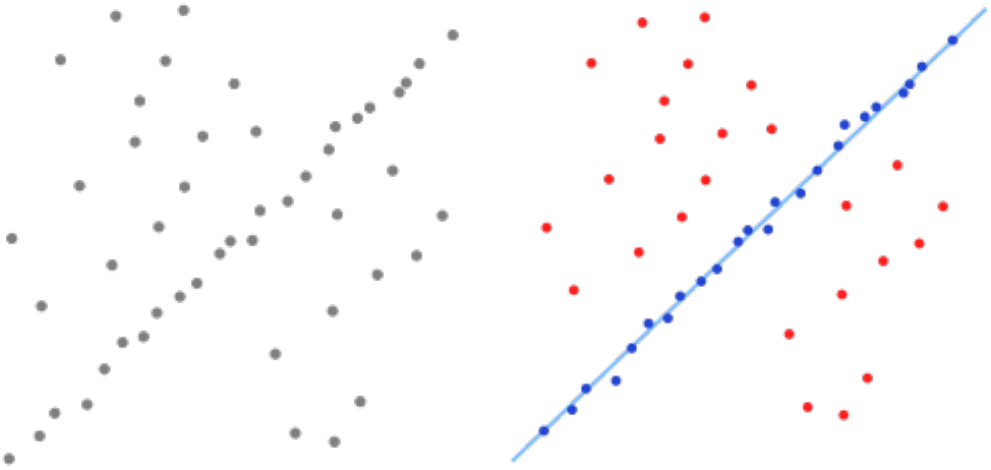


Figure 3.6: A 2D representation of RANSAC showing inliers in blue and outliers in red (source wikipedia.org)

A pseudo-code algorithm for RANSAC can be found in appendix A.

3.3.2 Segment selection

Segment selection is the process of filtering out useful segments and removing segments that add nothing to the final result. The process of deciding which segments to keep and which ones to remove is simple, segments are kept or rejected on the basis of 2 criteria:

Vertical Extent - The difference between the highest and lowest points in the segment

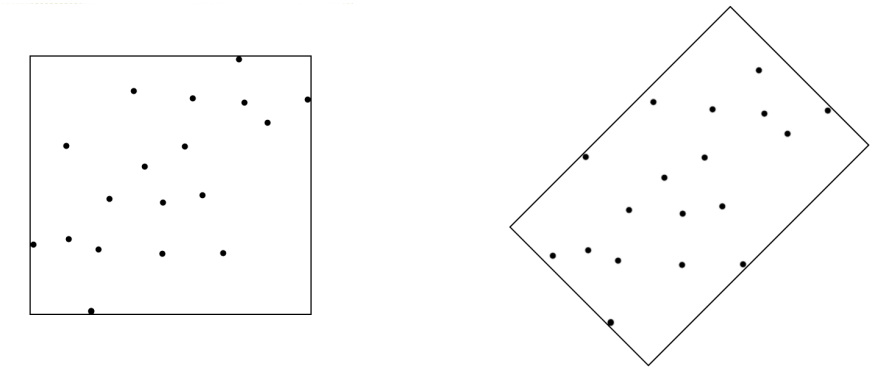
Angle from vertical - The angle between the segment and vertical.

Once The filtering has taken place only big planar segments that represent the extents of the room are left.

3.4 Model Generation

3.4.1 Bounding Box Generation

There are two types of Bounding boxes:



(a) AABB (b) OBB

Figure 3.7: (a) an Axis-Aligned Bounding Box, (b) an Oriented Bounding Box

An Axis-Aligned bounding Box is a 3-dimensional box, containing all the points in a point set, aligned with the x, y, z axes of the 3-dimensional Cartesian space of the point set.

An Oriented Bounding Box is a 3-dimensional box, containing all the points in a point set, aligned with the principal components of the point set.

Creating an Axis Aligned Bounding Box

An axis Aligned Bounding Box is created through a simple maximum and minimum process. Extract the maximum x, y, z points in the set as well as the minimum x, y, z points and use them to create a box around the point set.

Creating an Oriented bounding Box

Creating an object oriented bounding box is a 6 step process.

Step 1 Compute 3D centroid and the covariance matrix of the point set

Step 2 Extract the eigenvectors of the covariance matrix. These eigenvectors represent the orientation of the OBB

Step 3 Calculate the transformation parameters between the eigenvectors and the x, y, z axes.

Step 4 Use these parameters to transform the point set to the x, y, z axes.

Step 5 Fit a normal AABB to the point set.

Step 6 Transform the point set as well as the calculated AABB back to their original positions. The AABB not becomes an oriented bounding box.

3.4.2 Plane with Plane Intersection

The fitting of planes to segments as spoken about in Section **??** is a precursor to finding the intersection line of planar segments. Once a plane has been fitted to each segment, finding the line of intersection is simple 3D geometry.

The intersection of two planes with the equations $a_1x + b_1y + c_1z = d_1$ and $a_2x + b_2y + c_2z = d_2$ will result in a parameterized line, L, with the form $L = L_0 + t\vec{u}$ where $\vec{u}$ is the direction vector of the line and L_0 is a point on the line.

The direction of the line of intersection is the cross product of the two plane normals. Given:

$$\vec{n_1} = \begin{bmatrix} a_1 \\ b_1 \\ c_1 \end{bmatrix} \text{ And } \vec{n_2} = \begin{bmatrix} a_2 \\ b_2 \\ c_2 \end{bmatrix}$$

then $\vec{u} = \vec{n_1} \times \vec{n_2}$.

After calculating $\vec{u}$, to fully define the line, a specific point on it must be found.

That is, we need to find the point $L_0 = (x_L, y_L, z_L)$. There are 4 methods to do this:

1. Through a direct linear equation, this method however requires a user to set either a variable to zero. This is not practical.

2. By creating a third plane and intersecting the 3 planes to to get a point.

3. by creating a line on one of the planes and intersecting the line with the other plane.

4. Construct system of equations using Lagrange multipliers with one objective function and two constraints. This is the method that Point cloud library uses.

Through some derivation using Lagrange multipliers a system of equations is formed:

$$\begin{bmatrix} 2 & 0 & 0 & a_1 & a_2 \\ 0 & 2 & 0 & b_1 & b_2 \\ 0 & 0 & 2 & c_1 & c_2 \\ a_1 & b_1 & c_1 & 0 & 0 \\ a_2 & b_2 & c_2 & 0 & 0 \end{bmatrix} b = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -d_1 \\ -d_2 \end{bmatrix}$$

$$where\, b = \begin{bmatrix} x_L \\ y_L \\ z_L \\ \lambda \\ \mu \end{bmatrix}$$

From this we have the equation of the line, $L = \begin{bmatrix} x_L \\ y_L \\ z_L \end{bmatrix} + t\vec{u}$.

3.4.3 Orthogonal Distance of point to line

The distance from a point to a line is reasonably ambiguous as it could be to any point on the line. The orthogonal distance to a line is not ambiguous. It is the shortest distance from the point to the line.

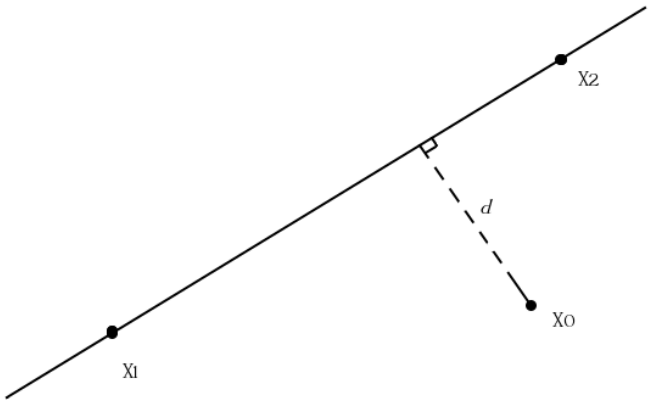


Figure 3.8: Orthogonal distance, d, to a line

? explains it as follows:

A line in 3-dimensions can be represented by 2 points, $\vec{x}_1 = (x_1, y_1, z_1)$ and $\vec{x}_2 = (x_2, y_2, z_2)$ giving:

$$L = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} + \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \end{bmatrix} t \tag{3.4}$$

The squared distance from point X_0 and a point on the line with parameter t is therefore:

$$d^2 = [(x_1 - x_0) + (x_2 - x_1)t]^2 + [(y_1 - y_0) + (y_2 - y_1)t]^2 + [(x_1 - z_0) + (z_2 - z_1)t]^2 \tag{3.5}$$

To minimize the distance, set $d(d^2)/dt = 0$ and solve for t to obtain:

$$t = -\frac{(\vec{x_1} - \vec{x_0}) \cdot (\vec{x_2} - \vec{x_1})}{|(\vec{x_2} - \vec{x_1})|^2} \tag{3.6}$$

The minimum distance can then be found by plugging t back into equation **??** and simplifying to obtain:

$$d^2 = \frac{|(\vec{x_2} - \vec{x_1}) \times (\vec{x_1} - \vec{x_0})|^2}{|(\vec{x_2} - \vec{x_1})|^2} \tag{3.7}$$

and taking the square root of both sides results in:

$$d = \frac{|(\vec{x_0} - \vec{x_1}) \times (\vec{x_0} - \vec{x_2})|}{|(\vec{x_2} - \vec{x_1})|} \tag{3.8}$$

3.4.4 Project points onto line

Projecting a point onto a line in 3-dimensional space is a similar problem to finding the orthogonal distance to a line. With the standard line equation of a line represented by two points:

$$L = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} + \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \end{bmatrix} t \qquad (??)$$

In both, projecting a point and orthogonal distance, cases you need to know a value for t. The formula for this, shown in equation **??**. Once t has been calculated getting the projected point is as simple as substituting its value into equation **??** and seeing the resulting x, y, z point. This point is the orthogonal projection of the point onto the line.

3.4.5 Creating an OBJ file

Due to the fact that obj files are plain text they are easy to write in the same way as a text file, as long as the format is correct. Obj files support many different types, these types are all differentiated by using a keyword at the beginning of each line.

For example writing a 3 planar objects that make a small portion of a cube will result in an obj file that looks as follows:

```
#Vertex List
v 0 0 0
v 1 0 0
v 1 1 0
v 0 1 0
v 1 0 1
v 0 0 1
v 0 1 1

#face list
o object.side.1
f 1 2 3 4
o object.side.2
f 1 2 5 6
o object.side.3
f 1 6 7 4
```

And results in a model that looks like Figure **??** below.

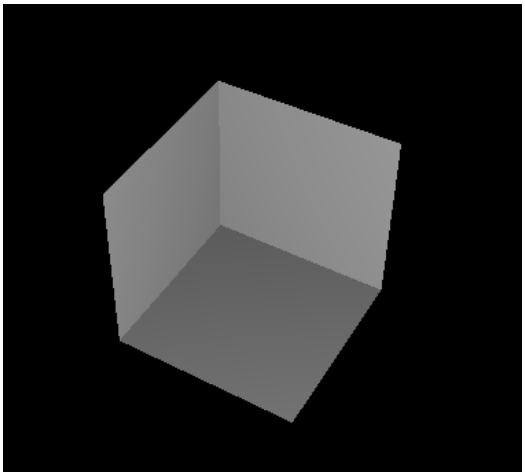


Figure 3.9: 3 sides of a cube

It is also possible to save textures and colours in an obj file by creating a .mtl (material) file. In the material file colours are defined in a similar way to obj files with "ka" representing an ambient colour "kd" a diffuse colour and "ks" a specular colour.

A material file also offers the ability to use texture maps, which are essentially maps of what colour is where for specific objects. Texture maps are stored in a .tga file.

Chapter 4

Results

The aim of this project is to create a boundary representation of an indoor environment from laser scan data. This section builds on the literature review and Method and describes the process undertaken to get a B-rep model.

The point clouds are processes through an command line program that takes in the point cloud name as a parameter, then saves the B-rep model as an obj file with the same name and same file location as the point cloud.

4.1 Segmentation

The segmentation of the point cloud is done with a built in region growing function from Point Cloud Library. A detailed description of the region growing and normal calculation process is given in Section **??**.

4.1.1 Normal Calculation

Before segmentation can take place normals need to be calculated for each point in the cloud. Normals are calculated using a Principal components

analysis, with the neighbourhood around the point, k, set to the nearest 40 points.

40 was chosen because it gives the best results, higher and the gaps at the corners of the room start to become very big making segments smaller, as well as no co-planar features are detected for example a door will become part of the wall. Any smaller and there is too much variation in the normals making the region growing segmentation leave holes in the planar segments such as in Figure **??**:

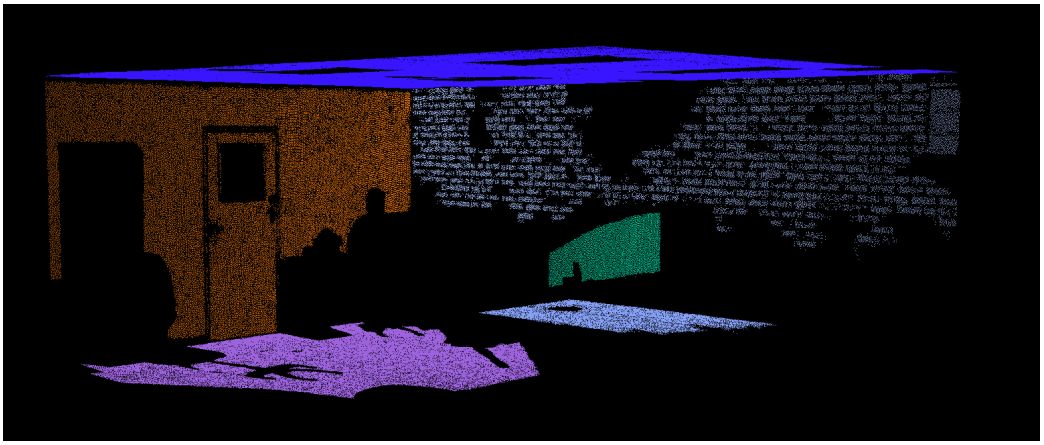


Figure 4.1: $k = 5$

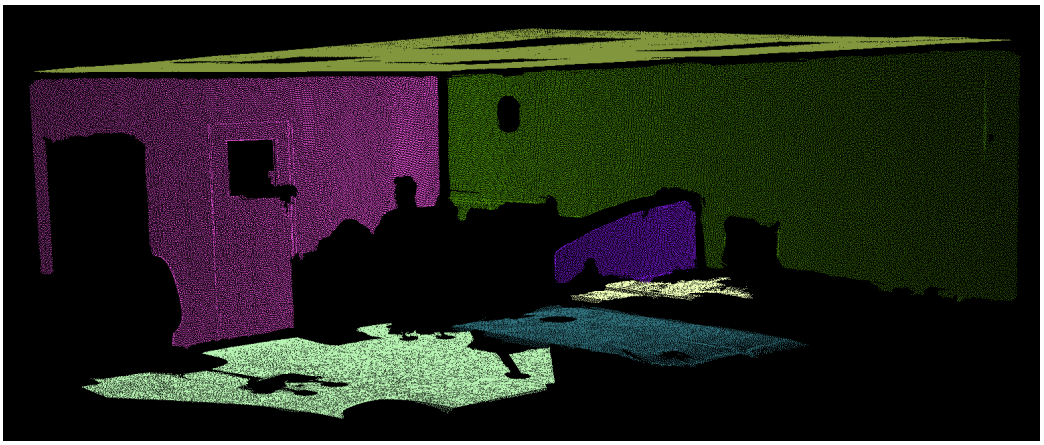


Figure 4.2: $k = 200$

Both images are done with the region growing parameters chosen in section **??**

There is also a speed aspect to the neighbourhood size, smaller results in a much faster computation, and larger much slower. This is not a huge issue as the normal estimation function has multi-threading making it only take up to 10s at most. This is only about 5% of the total run-time.

4.1.2 Region Growing

Once the normals have been calculated the point cloud is segmented using region growing. Region growing takes in 5 parameters; Minimum cluster size, Maximum cluster size,Search Method, Number of neighbours, Smoothness threshold and Curvature threshold.

In the case of this project bigger segments are good, so the Maximum cluster size value was never set allowing clusters be as large as the need to be.

The minimum cluster size was set to 5000 points because the chances of a significant wall section being less than 5000 points is very low, having this large number also removes a lot of the clutter in a room. Removing clutter is an important part of this project as any small segment thats not a part of the extent of the room (i.e. walls, roof or floor) will create problems with the rest of the program.

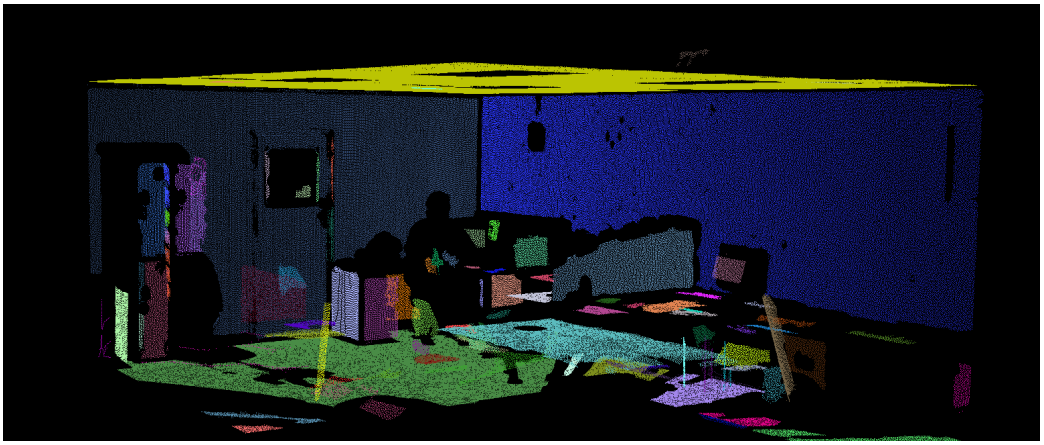


Figure 4.3: Minimum Cluster Size = 100

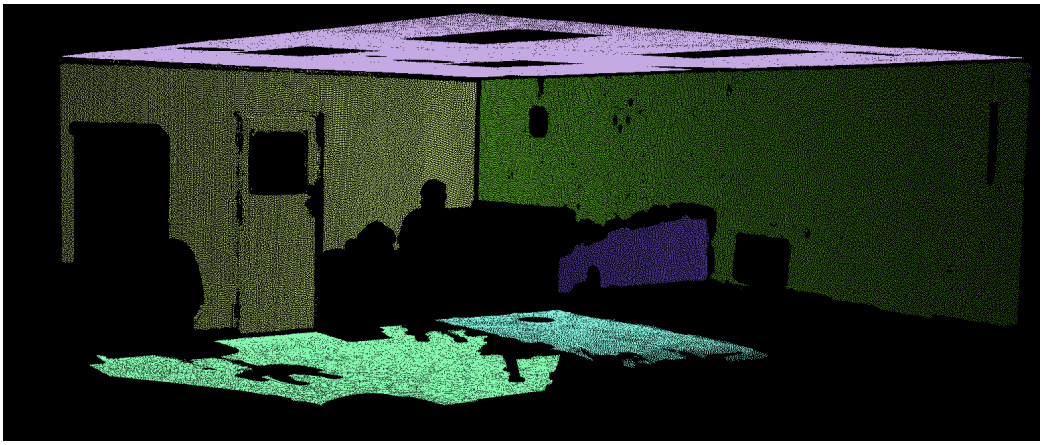


Figure 4.4: Minimum Cluster Size = 5000

The Smoothness and Curvature thresholds are very important in this project as the outcome relies on planar segments so strict thresholds on curvature and smoothness are important to make sure that any segment is a planar segment and does not cover two sides of a room. You can see in Figure **??** that the roof and right hand wall are one segment, resulting in a fitted plane that goes at a 45° angle through the room. As well as the person in the figure who is large enough to pass all the tests but is most defiantly

not a feature that needs to be kept. So because this project relies heavily on planes the Smoothness and Curvature thresholds are strictly set so that only planar segments are kept.

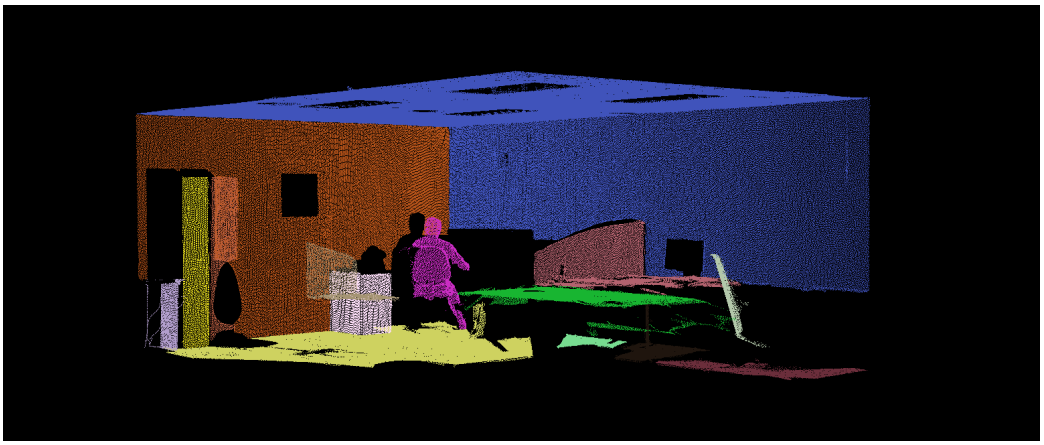


Figure 4.5: Region Growing Run with no Smoothness or Curvature Thresholds set

Number of neighbours and search method in region growing is the same as in the normal computation. The number of neighbours is set to the same value as the normal calculation. the main difference in varying values for number of neighbours is that with a very low value, shown in fig. **??**, the door can be seen as a separate segment to the wall and with a large value the door becomes part of the wall. This becomes important later

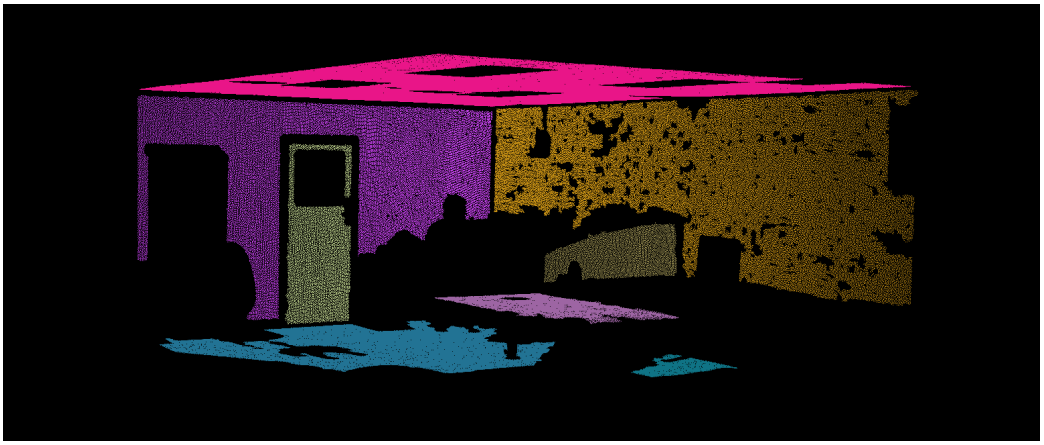


Figure 4.6: Number of Neighbours = 5

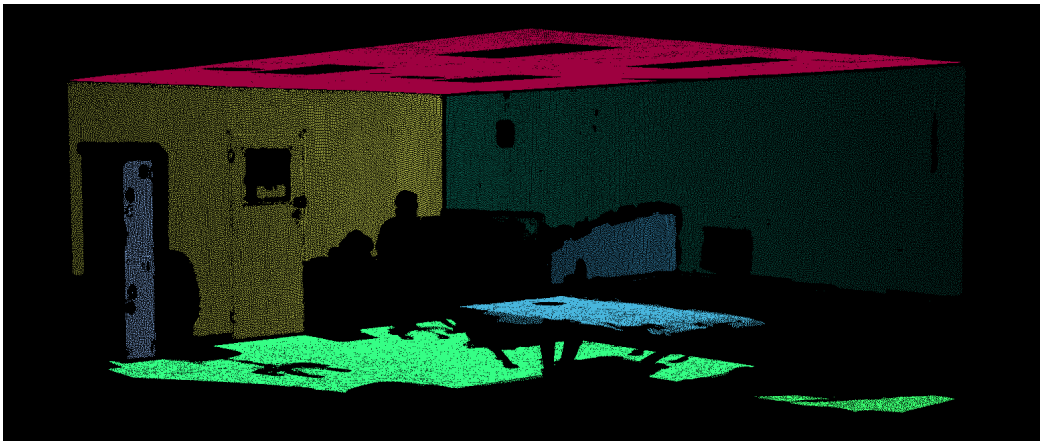


Figure 4.7: Number of Neighbours = 200

The final parameters, as shown in the constants.h file, are as follows:

```
int MinClusterSize = 5000;
int NumberOfNeighbours = 40;
float SmoothnessThreshold = (1.0 * M_PI / 180);
float CurvatureThreshold = 1.0;
```

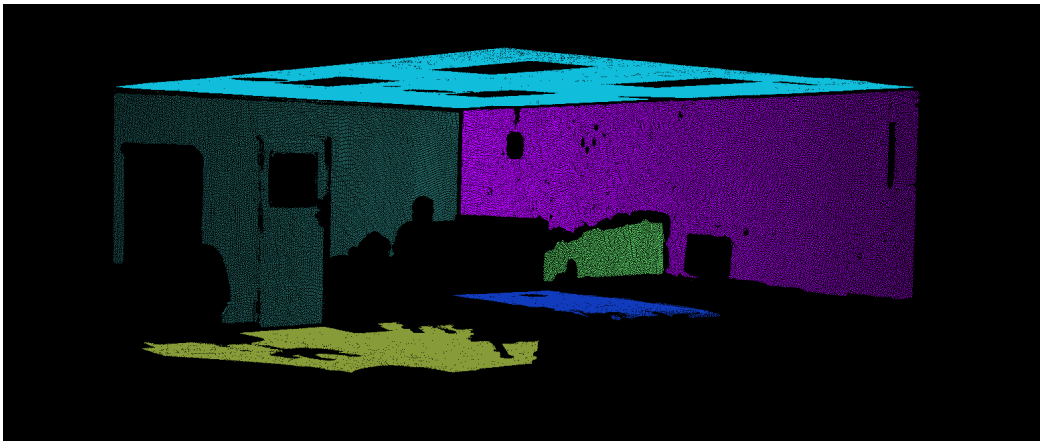


Figure 4.8: Final results of region growing

The segmentation of the point cloud is the most time consuming function in the whole process, taking up on average 80% of total run-time.

Once again the speed of the process depends greatly on the parameters set

4.2 Surface Extraction

It is immediately evident in Figure **??** that there are too many small segments that are not necessary and irreverent to the final output. So it becomes necessary to filter these segments out. The first obvious step is to up the minimum cluster size like in figure **??**. But even with the larger cluster size There are still a number of segments that need to be filtered out.

To begin this, a array of the segments is created after the region growing has taken place.

4.2.1 Segment selection

The array of segments created in the segmentation section of the program is not entirely useful due to the fact that it contains segments all the segments.

Most of the segments are small cluttering objects, like sides of desks of tops of tables. In process of trying to extract the boundaries of a room all this clutter is unnecessary, so we need to filter it out somehow.

filtering out Horizontal and Vertical surfaces

To start this process, the array of segments is split up into two separate arrays, horizontal segments and vertical segments. This is done by iterating through the segments and deciding which category they fall under.

A plane is fitted to the segment using the RANSAC method outlined in section **??**. Now the orientation of the segment can be determined from the coefficients a, b, and c of the planes equation.

$$ax + by + cz = d \qquad \vec{n}_{segment} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{4.1}$$

Once the normal to the segment is determined it is compares to a known vertical vector $\vec{v}$.

$$\vec{v} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{4.2}$$

The angle between the two vectors is then the deviation the normal has from vertical and can be thought of as the angle the segment has from vertical.

The angle is found using the dot product:

$$\vec{n}_{segment} \cdot \vec{v} = |\vec{n}_{segment}||\vec{v}|cos\theta \tag{4.3}$$

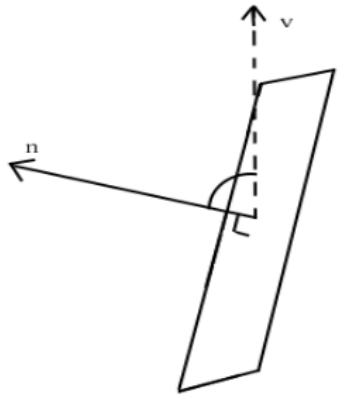


Figure 4.9: The deflection of the plane normal from vertical

Solving for θ gives us the angle between the segment and vertical. From here it is easy to decide if a segment is horizontal or vertical, by seeing if θ is closer to 0°/180°or 90°.

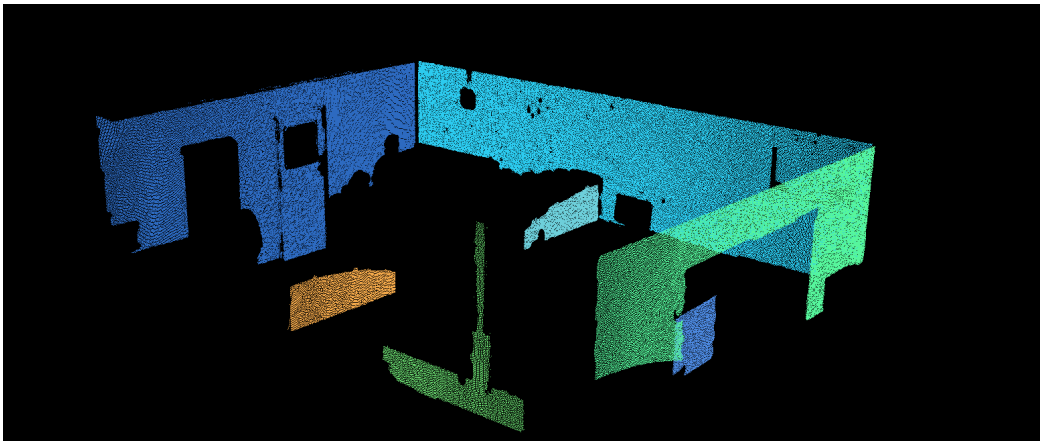


Figure 4.10: Vertical Segments Before Filtering

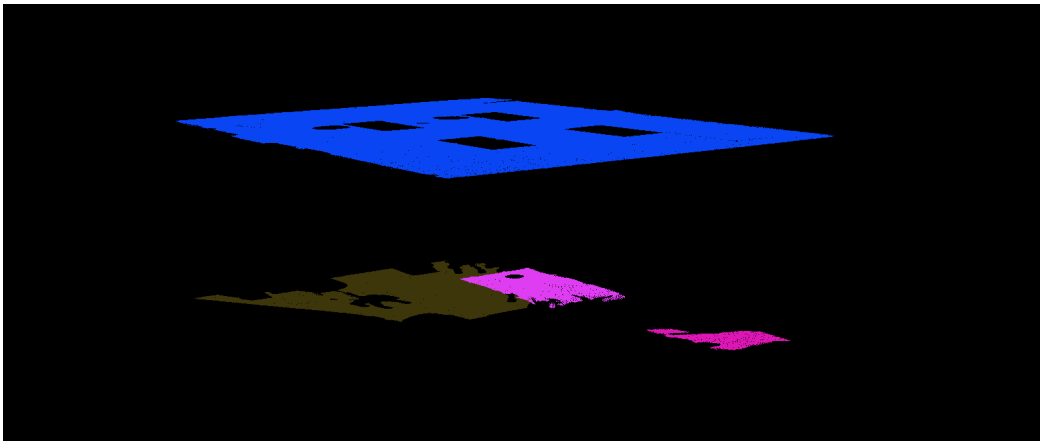


Figure 4.11: Horizontal Segments Before Filtering

Filtering of Vertical Segments

Vertical segments are filtered through two tests:

1. The vertical Extent of the Segment.

2. Another angle check.

The vertical extent of the segment is determined by finding the highest, $max = (x_{max}, y_{max}, z_{max})$, and lowest, $min = (x_{min}, y_{minx}, z_{min})$, points and getting the distance between them:

$$Vertical\ Extent = \sqrt{(x_{max} - x_{min})^2 + (y_{max} - y_{min})^2 + (z_{max} - z_{min})^2} \tag{4.4}$$

If the vertical extent of a segment is less that 1m the segment is removed on the basis that it is to small. And does not represent a wall.

The selection based on Angle is simply an extension of the previous method for splitting the the segments based on orientation, but with a much more strict angle defining vertical, θ must fall within $\pm 10°$of $90°$, i.e. the segment must be within a $10°$of perfectly vertical:

$$80^\circ < \theta < 100^\circ \tag{4.5}$$

Filtering of the Horizontal Segments

When deciding what is the roof and what is the floor in the whole point cloud a reasonable assumption to make is that the highest and lowest segments are the roof and floor respectively.

To find the highest and lowest segments is fairly simple. While looping through all the horizontal segments determine which segment has the highest average z value and the lowest average z value. These two segments are added to the filtered vertical segments in an array called Extent_Clusters.

The Extent_Clusters array contains only the outer most segments, essentially the roof, floor and walls.

4.3 Model Generation

The model generation section of this project is where the planar segments are turned into vertices's and faces so that they can be written to an obj file.

4.3.1 Boundaries of Segments

Step 1 in this process is to get the Oriented Bounding boxes of each of the segments. because the segments are planar the bounding boxes are also planar. This means that when a 3D cube is fitted to the segment there are only 4 unique points representing the cube as opposed to 8.

The 4 duplicate points are therefore removed leaving 4 points that are coplanar to the segment to represent the boundary of the segment as shown in Figure **??**.

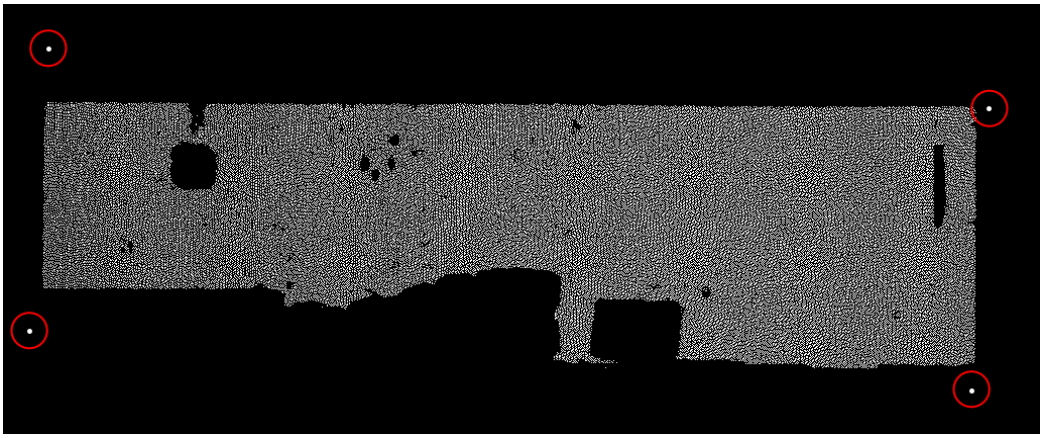


Figure 4.12: The 4 Boundary points of the segment indicated by the red circles

It is these coplanar boundary points that are adjusted to the corners of the room.

4.3.2 Plane with Plane intersection

As we have planes fitted to each segment from earlier in section **??** time can be saved by not fitting them again.

For this section two loops are run inside each other, iterating over the array Extent_Clusters. This makes it possible to intersect all the segments with each other, but also introduces the issue of the program attempting to intersect the same plane with itself of parallel planes with themselves, resulting in lines of intersection way off towards infinity of just errors.

To solve this the same function that checks the angle between the plane normal and vertical (section **??**) is used to check the angle between the two planes. If this angle is less than 15° then that pair of segments is skipped.

If the two segments pass this test, then they are intersected using the

method in section **??** and a line of intersection is returned.

4.3.3 Adjusting the boundary points through Extrusion

There are 4 points essentially defining each segment, it is these 4 points that are extruded so that the segment representing a wall of floor actually represents the full extent of that wall or floor. To do this a series of adjustments are applied to the boundary points so that they work their way outward towards the corners of the room. This is done using the Projection function.

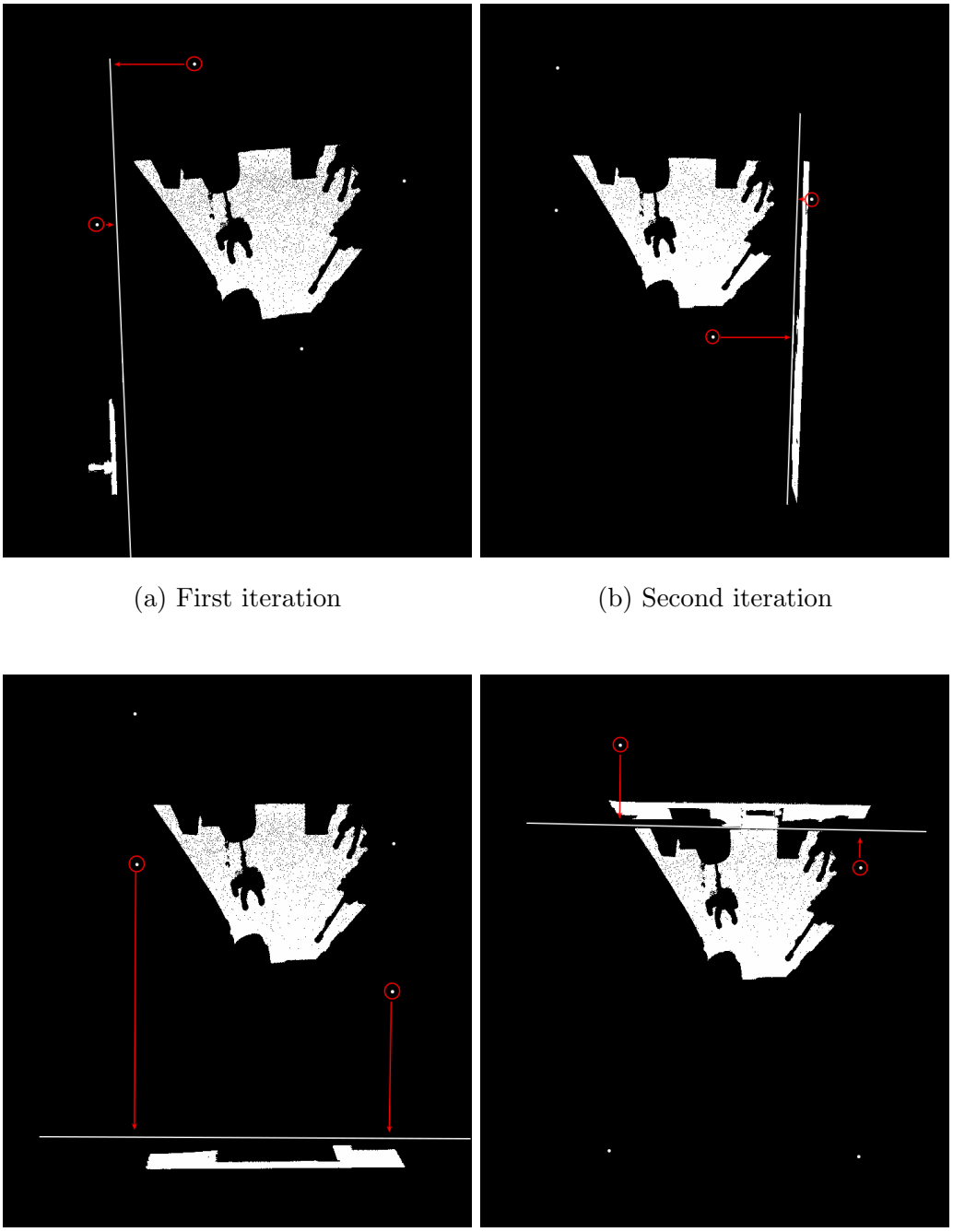
Only one of the segments boundary points are projected because all segments will meet twice, due to the two nested loops running through all segments each. This means that the first loops segment boundaries are adjusted with reference to the second loop, and the second loops segment boundaries are never adjusted.

The Projection function is provided with only 2 objects, the line of intersection that was created earlier, and the boundaries points that were created in **??**. Note that in the line of intersection computation the angle between the two segments is checked as explained in **??**.

The Projection function projects the closest two points, determined using the method in Section **??**, to the line of intersection between the who segments.

This process is graphically shown in Figure **??**, sub-figure (a) shows the first iteration, (b) shows the second, and so on.

The two red points circled in each of the figures are the closest two points to the line of intersection. These points are projected onto the line of intersection, this projection is shown by the arrows.



(a) First iteration

(b) Second iteration

(c) Third iteration

(d) Fourth iteration

Figure 4.13: The iterative extrusion of boundary points to the lines of intersection

Figure ?? shows the complete path taken by each boundary point as it is projected onto each line of intersection. The blue points are starting positions of the boundary points as calculated in ??. These points after projection result in the red points.

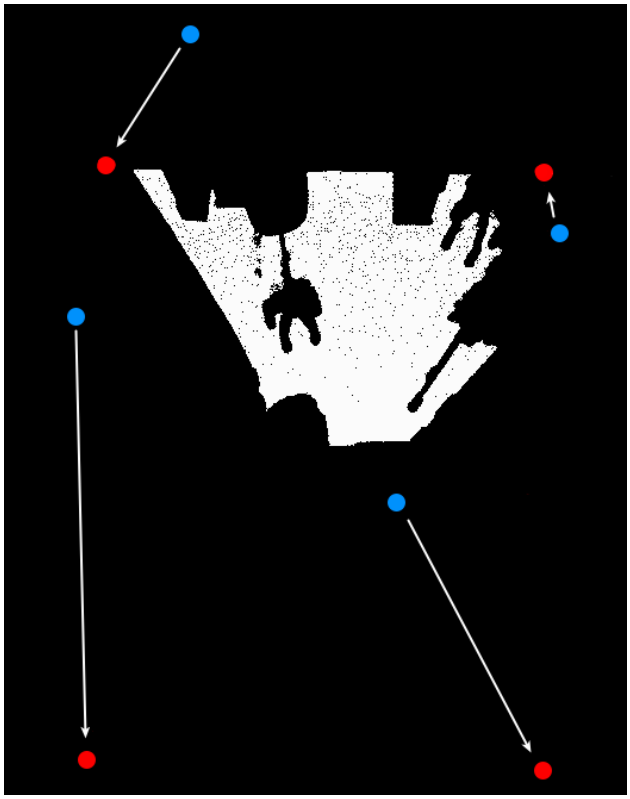


Figure 4.14: The final positions of the boundary points after Extrusion represented by the Red points and original positions represented by the Blue points

4.3.4 Dealing with Duplicate Corner Points

Due to the fact that the corner of a room is created by the intersection of 3 segment planes, there will be 3 boundary points at each corner, one for each segments plane.

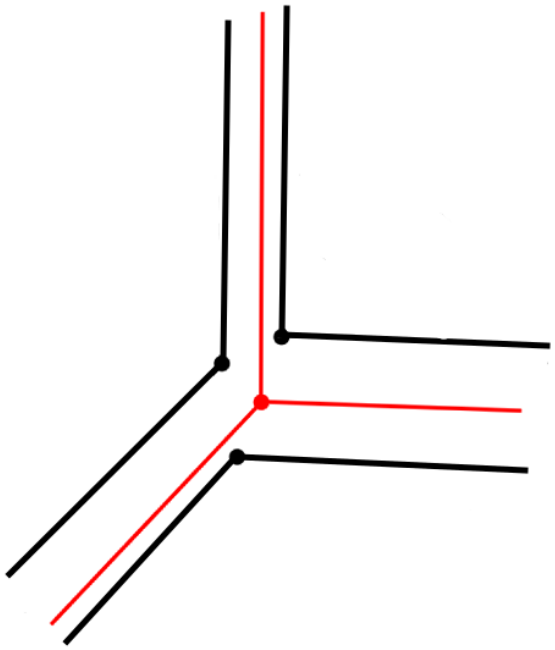


Figure 4.15: Duplicate Cornet points shown in black, averaged corner shown in red.

These points do not always have the same value and usually vary by up to 5cm. To solve this issue of multiple points where there should be one point, a simple clustering algorithm is run to group together points close together.

The clustering algorithm simply take points that are within 5cm of each other and places them in an array defining that corner.

The array is then averaged and that average value set as the value of the 3 corner points. Resulting in 1 point representing a corner.

Figure **??** shows in black what the corner of a room without averaged points will look like. The red point and lines shows what the corner will look like after averaging.

4.3.5 Creating the B-rep Model from the Adjusted Boundary Points

Creating the boundary representation model from the the averaged, Adjusted boundary points is the last part of this project. The model is created by iterating through each set of boundary points and writing each one of them as a separate object to an OBJ file as outlined in Section **??**. Resulting is a simple boundary representation of the original laser scan.

The output of the B-rep model are lines representing the edges of the room as seen in Figure **??**, **??** and **??**.

The results of different point clouds can be found in Appendix B.

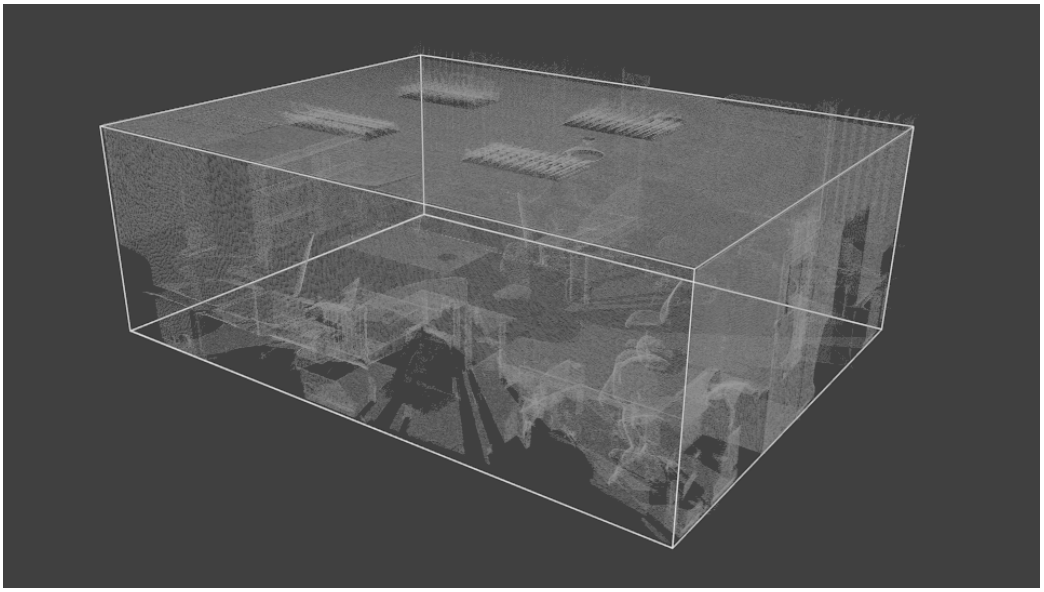


Figure 4.16: Final Boundary representation of the room looking from the front right corner of the room

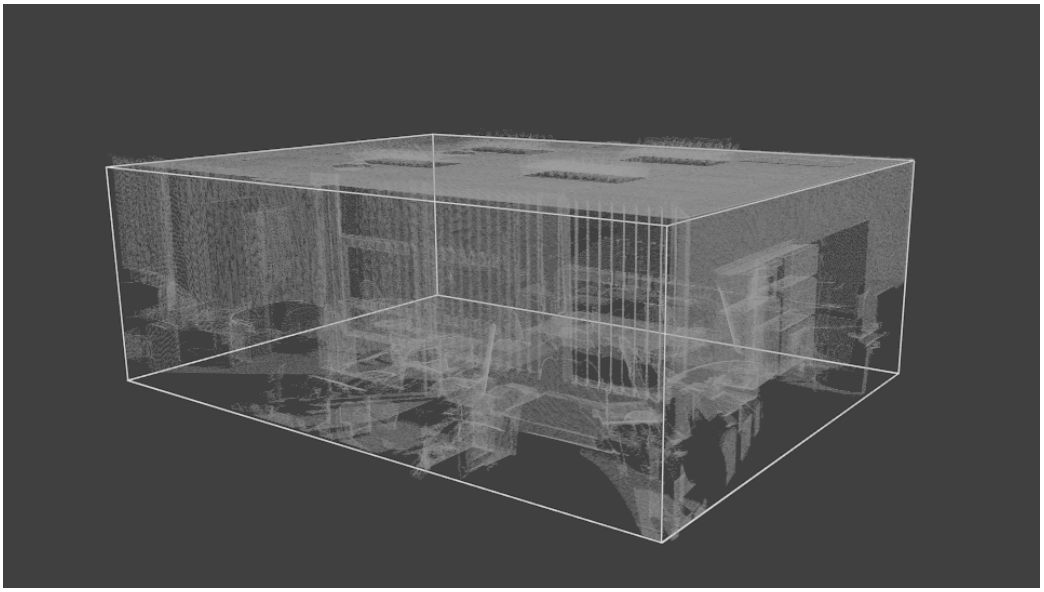


Figure 4.17: Final Boundary representation of the room looking from the back left corner of the room

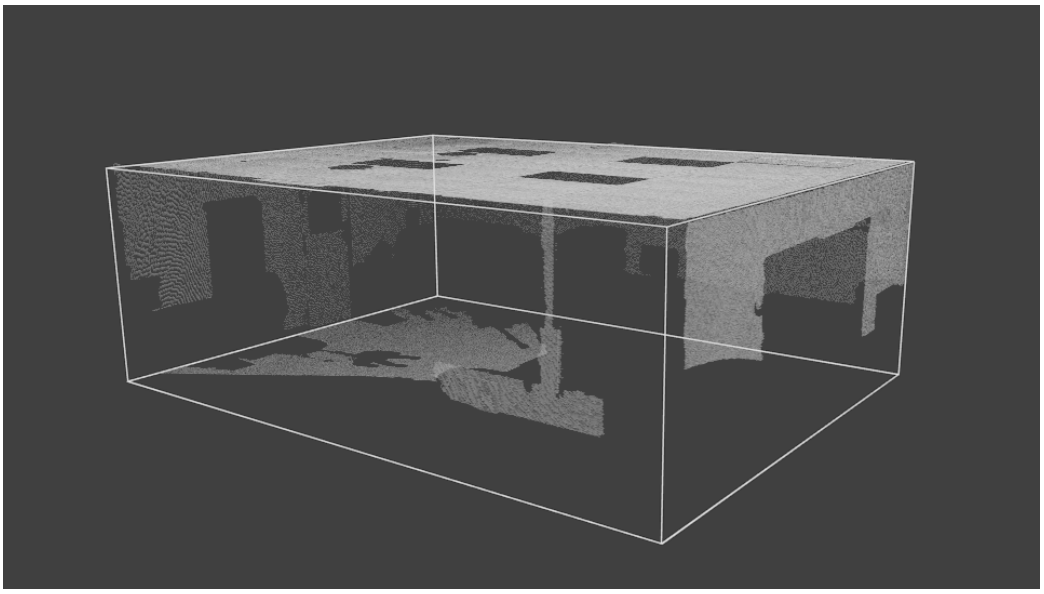


Figure 4.18: Final Boundary representation of the room with segmented cloud

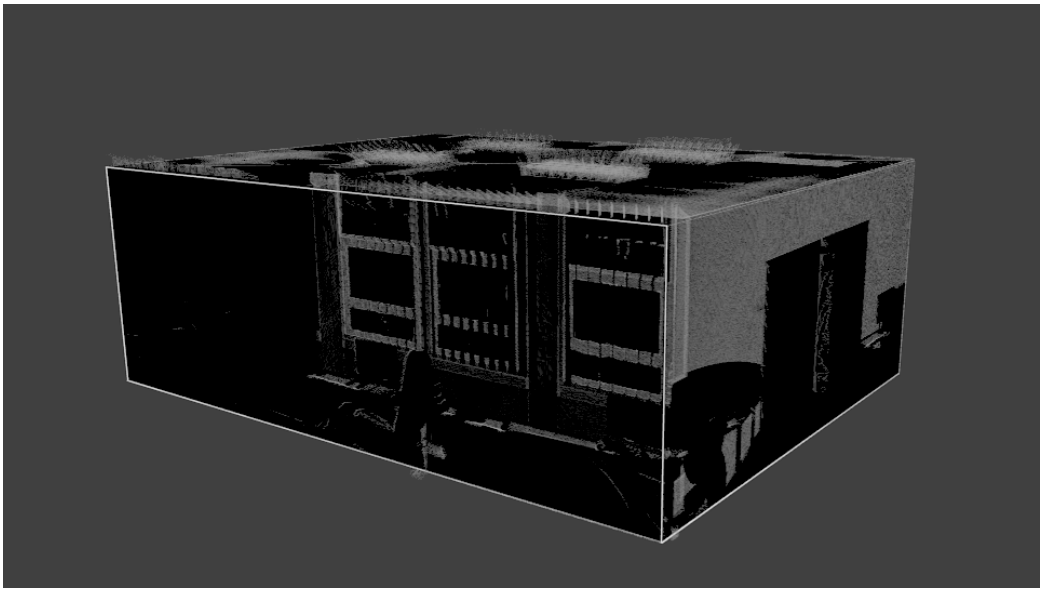


Figure 4.19: things

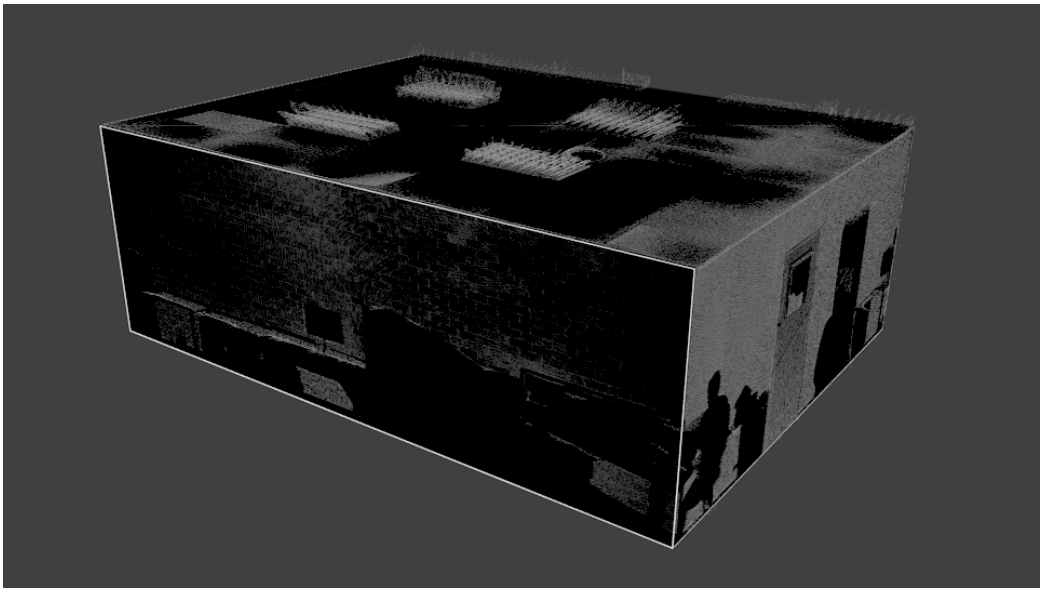


Figure 4.20: things

Chapter 5

Discussion

The process described in this Report offers a way of creating boundary representations of indoor scans without the need for user interference or preprocessing of the cloud. Section **??** shows this report has succeeded in creating a boundary representation of an indoor scan that room does not differ from the real room by a significant amount. even though the scan is not the best and there are obstacles for the process to contend with.

Figure 5.1: Objects and blinds obscuring the left hand wall

5.1 Comparison of model to Real Measurements

In determining if the resulting Boundaries are correct, and a good representation of the room that has been scanned, measurements were taken of the room with a distometer and corresponding measurements were taken on the model. the results of this comparison can be seen in table **??**.

m	Distometer	Model	Difference
1	7.205	7.208	0.003
2	5.426	5.420	0.006
3	7.203	7.206	0.003
4	5.422	5.427	0.005
H 1	2.492	2.495	0.003
H 2	2.495	2.497	0.002

Table 5.1: Measurements with a distometer versus measurements taken on the resulting model

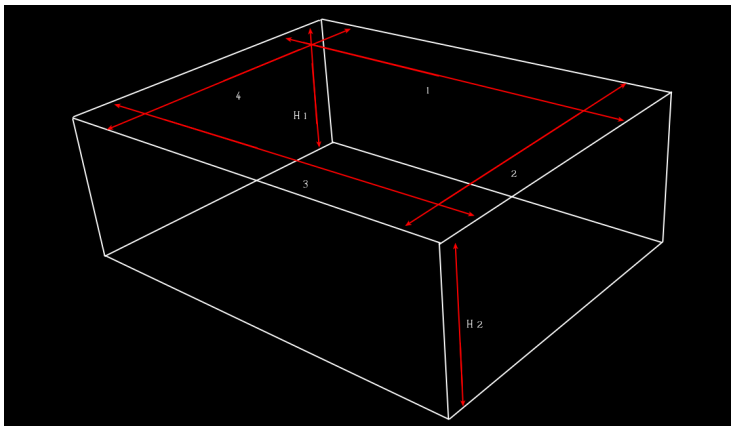


Figure 5.2: Locations of the measurements in Table **??**

It is important to note that the Oriented bounding box of the room has the dimensions 5.795m, 7.455m, 2.900m. The OBB is therefore 37cm bigger along the width of the room, 25cm in the length and 40cm bigger vertically. This is important because it may look like a bounding box has been created but that is not the case.

5.2 Accuracy of resulting model

Figure **??** shows how a specific segment varies from the final model. The error ranges from perfect fit, shown in red, to an error of 10mm in the blue section at the top left.

Figure 5.3: Blue to Red scale of point variance from fitted plane on right hand wall segment

It is easy to see that the only major deviation from a perfect fit is in the top left of the Figure, this is due to a poster being stuck on the wall there. It is reassuring to see that even with the poster on the wall the results have not been effected.

5.3 Limitations of this process

As it stands the main limitations on this process are the complexity of the room that is scanned. Rooms that are reasonably rectangular can be processed successfully. But as soon as small sections start to kink out or there are walls that start to run at different angles the Extrusion method explained in section **??** fails. This can be seen in the images of a secondary point cloud boundary representation process in Appendix B.

5.4 File Size

A benefit of this system is that large point clouds can be greatly reduced in size without loss of to much vital information. Of coarse there is a loss of detail, but as said in the previous section this is a current limitation on the process, but can be overcome in future versions of this system.

The two indoor scans of around started off with a resolution of 1.3mm point spacing at a distance of 10m, file sizes of around 7GB. Through down sampling the point clouds the point spacing was increased uniformly to a 1cm, reducing file size to less than 50MB. Then after the process outlined is this report is run on the down sampled point cloud they are simplified to 24 points and a file size of under 5KB.

Obviously as the detail in the model increases and materials start to be used the file size will increase but never to the extent that a full point cloud will.

Chapter 6

Conclusions and Future work

In conclusion this report has succeeded is creating a process to create boundary representations of indoor rooms. The process described in this report is still in its infancy, and has a few limitations.

6.1 Improving Detail in the System

The progression from the point that this process is at can be extended in a number of directions, firstly the current models created are very simplistic and cannot deal with rooms that have more than 6 sides, essentially only very simple rooms. The need for this process to be able to deal with more complex rooms is the first major improvement that will take place.

From the complexity of the rooms increasing, the detail in the model is the next step in making the process more complete, adding doors and windows into the models. This presents its own new challenges because the program will need to be able to differentiate between wall segments, door segments and general cluttering segments. This makes the segments selection section substantially more important.

6.2 Extending the Process to Multiple Rooms

another extension to this system is the ability to process more complex networks of rooms. Figure **??** shows a complex network or rooms that could potentially be turned into CAD models in the future of this process. This concept can be extended even further into multiple story networks. from here it becomes necessary to create topology in the points before they are processed, so that rooms and floors can be differentiated from each other.

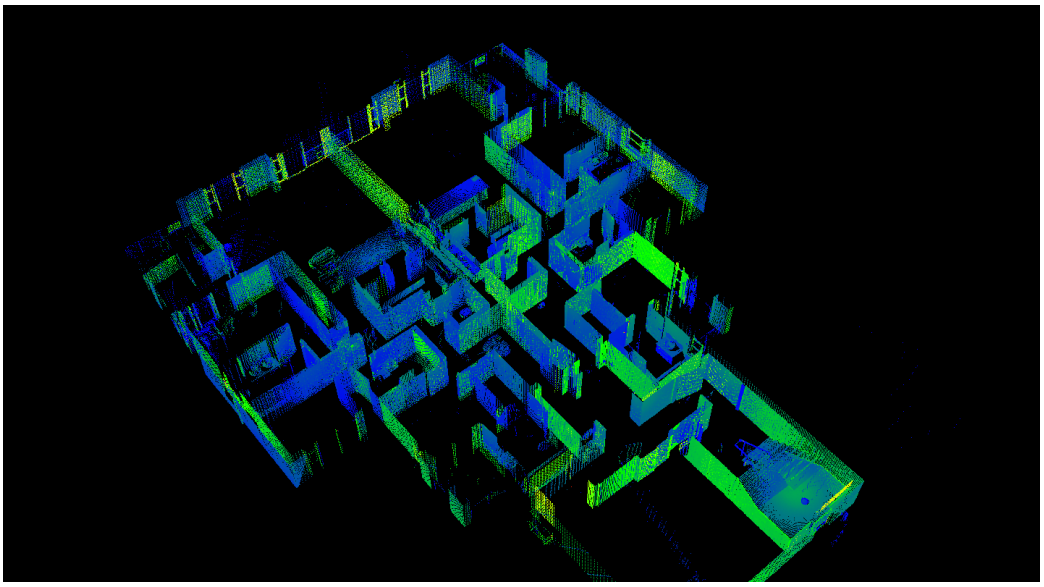


Figure 6.1: A more complex network of rooms within a building

6.3 User Interface

Currently the whole system is run from a command line and the parameters are fixed with no way to change them other than recompiling the whole project. Creating a user interface that allows users to change parameters themselves as well as a visualizer build into the user interface that allows

users change parameters and see the result in the same window. This is the final step in making this system into a full package,is to build additional, and unrelated, functionality into the system. Such as creating mesh models of point clouds or even simple down sampling of clouds.

Appendix A

Pseudocode

A.1 Region Growing

- **While** $\{A\}$ *is not empty* **do**
  - *Current region* $\{R_c\} \leftarrow \emptyset$
  - *Current seeds* $\{S_c\} \leftarrow \emptyset$
  - *Point with minimum curvature in* $\{A\} \rightarrow P_{min}$
  - $\{S_c\} \leftarrow \{S_c\} \cup P_{min}$
  - $\{R_c\} \leftarrow \{R_c\} \cup P_{min}$
  - $\{A\} \leftarrow \{A\} \setminus P_{min}$
  - **for** $i = 0$ *to* **size** ($\{S_c\}$) **do**
    - *Find nearest neighbours of current seed point* $\{B_c\} \leftarrow \Omega(S_c\{i\})$
    - **for** $j = 0$ *to* **size** ($\{B_c\}$) **do**
      - *Current neighbour point* $P_j \leftarrow B_c\{j\}$
      - **If** $\{A\}$ *contains* P_j *and* $cos^{-1}(|(N\{S_c\{i\}\}, N\{S_c\{j\}\})|) < \theta_{th}$ **then**
        - $\{R_c\} \leftarrow \{R_c\} \cup P_j$
        - $\{A\} \leftarrow \{A\} \setminus P_j$
        - **If** $c\{P_j\} < c_{th}$ **then**
          - $\{S_c\} \leftarrow \{S_c\} \cup P_j$
        - **end if**
      - **end if**
    - **end for**
  - **end for**
  - *Add current region to global segment list* $\{R\} \leftarrow \{R\} \cup \{R_c\}$
- **end while**
- **Return** $\{R\}$

A.2 RANSAC

```
Given:
data - a set of observed data points
model - a model that can be fitted to data points
n - the minimum number of data values required to
   fit the model
k - the maximum number of iterations allowed in the
    algorithm
t - a threshold value for determining when a data
   point fits a model
d - the number of close data values required to
   assert that a model fits well to data
Return:
bestfit - model parameters which best fit the data
   (or nil if no good model is found)
iterations = 0
bestfit = nil
besterr = something really large
while iterations < k {
        maybeinliers = n randomly selected values
           from data
        maybemodel = model parameters fitted to
           maybeinliers
        alsoinliers = empty set
        for every point in data not in maybeinliers
            {
```

```
            if point fits maybemodel with an
               error smaller than t
            add point to alsoinliers
        }
        if the number of elements in alsoinliers is
            > d {
            % this implies that we may have
               found a good model
            % now test how good it is
            bettermodel = model parameters
               fitted to all points in
               maybeinliers and alsoinliers
            thiserr = a measure of how well
               model fits these points
            if thiserr < besterr {
                    bestfit = bettermodel
                    besterr = thiserr
            }
        }
        increment iterations
}
return bestfit
```

Appendix B

GTL Full Results

ADD TEXT ABOUT THESE IMAGES

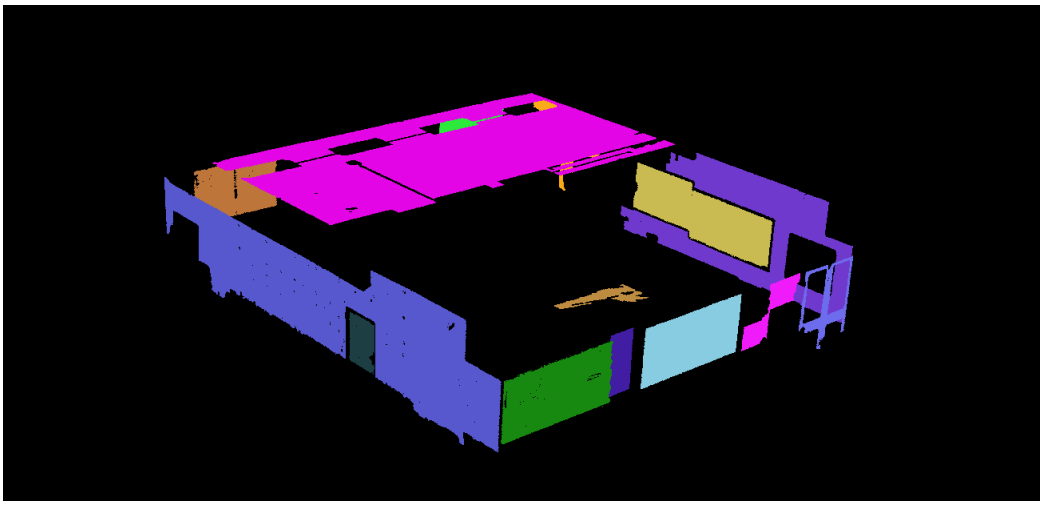


Figure B.1

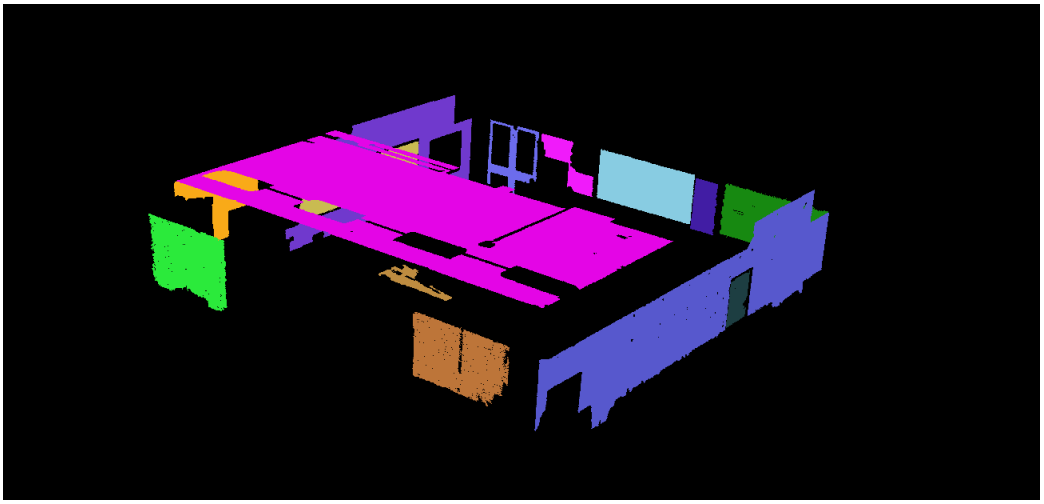


Figure B.2