University of Otago

Department of Computer Science

COSC490 Project Report

Plane Detection in Point Clouds

Author:
Helena Crawford
(7562257)

Supervisor(s):
Assoc. Professor Steven Mills

July 20, 2020

Abstract

Finding geometric planes in dense point cloud models can give a more useful and compact representation of the object being modelled. In this project, accurate and computationally efficient methods of detecting these planes are explored. A parallel implementation of RANSAC with 3.5x speedup over 4 threads has been created, with reasonable classification of noise, but too few planes detected to completely represent the structure of dense models. I aim to further improve the accuracy of this process, and to increase performance with other techniques beyond parallelization.

1 Introduction

Multi-view stereo modelling devices often produce a point cloud as output, where many points in space make up the object being modelled. This project's main aim is to produce accurate planar models from these point clouds using a plane detection algorithm.

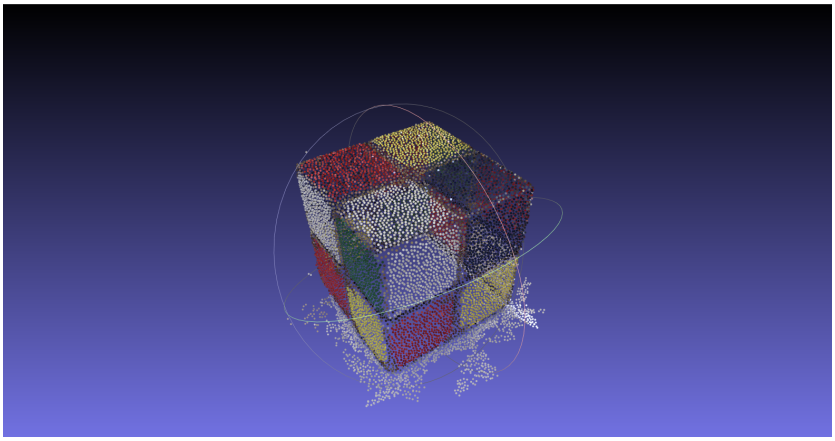

Figure 1: Point cloud of a wooden cube

1.1 Albany St Studio Model

Planar models are particularly good at efficiently representing buildings and architecture over more irregular objects (such as human beings). Conversely,

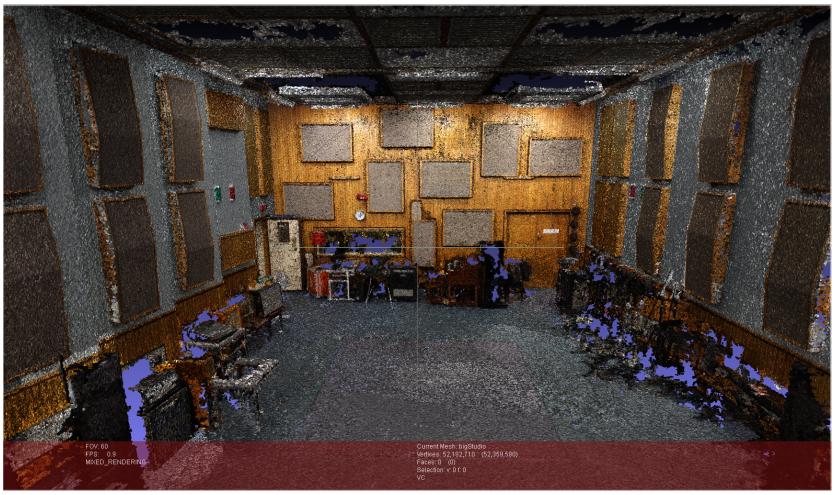

Figure 2: Point cloud of the Albany St recording studio

point clouds are not the ideal medium for representing large subjects in high fidelity. For example, a recently captured model of Albany Street Studio has 52 million points, taking up approximately 1.37 gigabytes of storage. This room at the time of modelling was composed of mainly planar architecture, with poorly captured and insignificant objects (for the architectural context) cluttering the 3D model at floor level. A representative 3D model of just the planar architecture would be significantly easier to work with in terms of storage size and computation time for viewing, as well as when calculating other attributes of the building such as the bouncing of sound waves in acoustics modelling. Many architectural point cloud models also have high levels of clutter and density of points, making this project viable for other subjects of interest in the field.

1.2 Aims

In this project I aim to create a plane detection algorithm efficient enough to process large models like Albany St studio in reasonable time. This algorithm should be robust to noise from object clutter or measurement error, and accurate enough to replace the original point cloud.

2 Background

An overview of the structures that 3D models of different types are composed of, and the different algorithms and variants used to detect them from point clouds in the field is important to understand this project. Much of this interim report will focus on the parallel computing analysis and implementation used to improve performance of the plane detector, so an introduction is also given to the chosen parallelization API.

2.1 3D Model Representations

The original title of this project, Modelling with Lines and Planes, offered a choice between detecting two different structures within objects in three-dimensional space. Lines are one-dimensional structures in a three-dimensional space with two points defining their direction, while planes are two-dimensional structures defined by three points (note that any subspace with one dimension less than its surroundings is known as a Hyperplane). Both structures require algorithms to detect and verify their quality, i.e. the number of points in the cloud that can be assigned to them (the extremes of which define their length within the model), and how well they represent the desired geometric qualities of the scene. I chose planes for this project with an interest in how they can more fully represent a scene on their own, as Hofer et al. note that the best 3D mesh reconstruction of their objects comes from a combination of lines and points [5, 6]

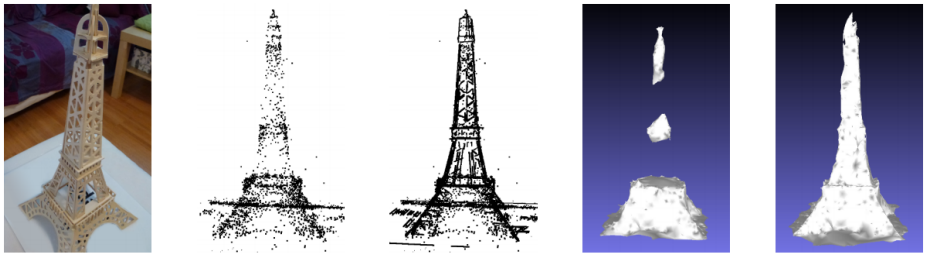

Figure 3: The original object, points, lines detected and 3D meshes generated in Hofer et al. The last 3D mesh is generated from both lines detected and points from the point cloud input.

Figure 4: Model of a toki (a Māori stone adze) created using multi-view stereo methods, and the main planes identified in it:

2.2 RANSAC and Variants

Two prominent algorithms for fitting models to noisy data are Random Sample And Consensus[1], and the Hough Transform. The Hough transform finds instances of shapes by a voting procedure in a parameter space, which becomes unwieldy in dimension when more than three parameters are required. However as planes only require 3 points to define, it is suitable for plane detection and has been used to great success[2]. The PLY file format can encode point clouds with optional properties beyond location, such as colour, normal vector direction, texture co-ordinates and faces for 3D meshes. Limburger et al. use normal vectors of the input points to quickly detect planes in the scene, however Albany St studio and other models used for this project only have point locations and colours with no geometric relationships between individual points. So although this variant of the Hough transform is highly optimized and accurate, their work is not suitable for use in this project.

RANSAC finds planes by randomly generating hypotheses (sets of three points), fitting points to those hypotheses and comparing the plane sizes over a large number of trials, and was chosen for this project over the Hough transform due to its familiarity and simplicity. Many variations on RANSAC exist for different applications such as BaySAC[3] (using previous trial outcomes to propose more accurate hypotheses) and Preemptive RANSAC[7] (generated a fixed number of hypotheses and verifying them in parallel). While these were not directly applicable to my plane detector, the latter informed my analysis of parallelism at the plane and trials levels of RANSAC.

Input: $X = \{\vec{x}_1, \vec{x}_2, ..., \vec{x}_n\}$, a set of 3D points
P the number of planes to find
T the point−plane distance threshold
R the number of RANSAC trials

Algorithm Basic Random Sample And Consensus for Plane Detection

```
1: procedure RANSAC(X, P, T, R)
2:     for p ← 1, P do
3:         bestPlane ← {0, 0}
4:         bestPoints ← {}
5:         for r ← 1, R do
6:             S ← {x_1, x_2, x_3}                    ▷ 3 points at random from X
7:             thisPlane ← fitPlane(|S|)
8:             thisPoints ← {}
9:             for all x_i ∈ X do
10:                if distance(thisPlane, x_i) < T then
11:                    thisPoints ← thisPoints + x_i
12:                end if
13:            end for
14:            if |thisPoints| > |bestPoints| then
15:                bestPlane ← thisPlane
16:                bestPoints ← thisPoints
17:            end if
18:        end for
19:        output bestPoints
20:        X ← X − bestPoints
21:    end for
22: end procedure
```

2.3 OpenMP

OpenMP is a multiprocessing API chosen in this project for its ease of implementation and portability. It creates a virtual shared-memory architecture, where all threads running in parallel have equal access time to a shared block of memory. The number and allocation of threads to processors is determined in run-time by the hardware available, background workload and the nature of the block of code marked as parallelizable. Variables can be assigned private to each thread or shared between them.

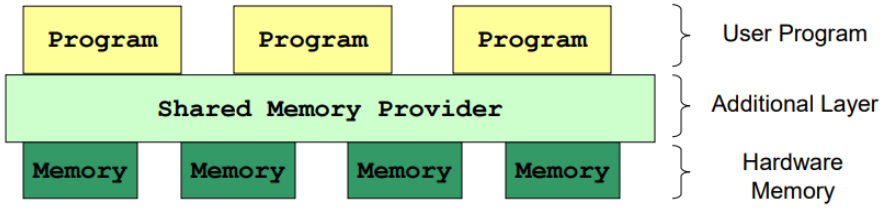

Figure 5: Shared memory diagram, National University of Singapore CS3210 - AY1920S1 - L3

3 RANSAC Inputs and Output

Ideally the number of random model trials to run, R, should be determined from the probability of finding the largest remaining plane in the scene. The distance threshold, T, required for points to be on a plane is calculated from the average of the model's bounding box size over the three dimensions. This simple metric produces reasonable thresholds when multiplied by a scalar factor, which after some experimentation was set to 0.01. This distinguishes planes that are a distance of approximately 1% of the scene's scale apart. The number of planes, P, can be set dynamically by simply continuing to find planes until a user-defined percentage of the scene is explained. The inverse of this percentage can be thought of as the noise level: the desired number of points to ignore and leave unassigned to any plane. In trials this was set at 99%, or 1% of points being classified as measurement noise or clutter.

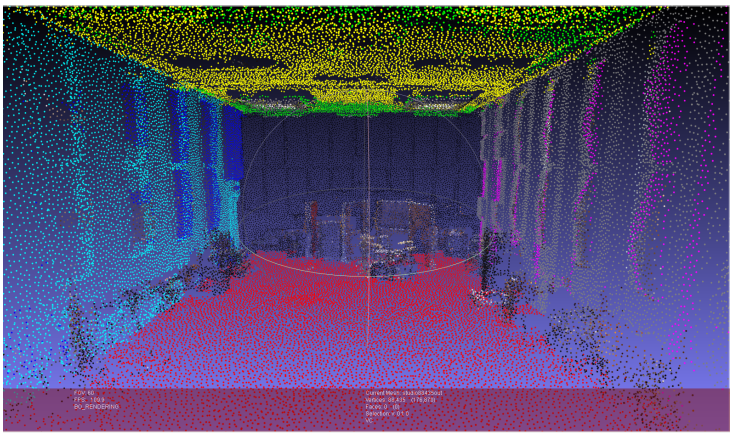

Figure 6: View of the Albany St studio points coloured for 9 planes

So far, the plane detector produces a PLY file as output with inlier points recoloured to represent the found plane they lie on. It also returns the equations of the planes found in the model co-ordinate space. Ideally, the plane detector should return both the equations and outermost point co-ordinates of each plane for rendering a purely planar model. Adding a texture generated from the incoming point cloud colours is also proposed for future work, in order to efficiently represent the colouring of the subject as well as its geometry.

4 Parallelization

Parallel computing is used in this project to easily improve performance of the program by capitalizing on the number of repeated, independent tasks in the RANSAC algorithm and the multi-processor hardware available. Initially, three possible levels of loops in the RANSAC algorithm were proposed to parallelize upon: trials, planes and points. As the former of these loops encase the latter, only one level of parallelism could be implemented without significant restructuring of the RANSAC algorithm.

4.1 Synchronization

Before the parallel speedup of any algorithm can be evaluated, its safety and correctness must be ensured. In a shared-memory architecture, this relies on synchronization of access to shared resources to avoid simultaneous write operations and maintain the intended order of operations. OpenMP provides numerous synchronization constructs that, while easy to use, can create significant overhead cost in memory and execution time. Of the three levels proposed to parallelize on, the distance calculations for each point in the cloud was chosen to minimize synchronization needed while promising direct performance gains for increasing model sizes.

4.2 Planes in Parallel

Finding planes in parallel was proposed via a primary/secondary thread scheme, where each secondary thread runs its own set of RANSAC trials to find a plane, while the original primary thread maintains and updates a list of each point's assignment to the largest plane found thus far. This sharing

```
2: for p ← 1, P do
3:     bestPlane ← {0, 0}
4:     bestPoints ← {}
5: end for
```

of work has a few immediate issues. Firstly, every thread is searching in the same field without the restriction of the point cloud after each plane is found serially, so may come to similar conclusions that give no performance gains. Secondly, synchronization would be heavily required and inefficient in the chosen medium of OpenMP (in fact, this primary/secondary structure would better suit a distributed-memory system where separate processes message each other such as the Message Passing Interface). Lastly, the performance gains depend entirely on the number of planes in the model rather than the size, whereas the object's planar complexity is a far smaller consideration to the aims of this project than its point cloud size.

4.3 Trials in Parallel

```
5: for r ← 1, R do
6:     S ← {x_1, x_2, x_3}                ▷ 3 points at random from X
7:     thisPlane ← fitPlane(|S|)
8:     thisPoints ← {}
9: end for
```

RANSAC trials mostly operate independently until a final plane size is obtained, at which point a trial must compare its size to the largest plane found thus far in the set of trials, and update the largest plane if is exceeded. I predicted synchronizing both the read of the shared largest plane and the write to update the largest plane would be a costly exercise from my previous parallel computing experience. However, this approach still held promise due to the length and complexity of code preceding the operation. The selection of points, creation of new vectors and planes, and calculations required before comparing planes could save significant run-time when done in parallel, possibly enough to justify the overhead and synchronization costs.

Parallel RANSAC trials have been used in Pre-emptive RANSAC [7], where

a fixed number of hypotheses are generated and verified in parallel using a breadth-first scheme. Nistér's work highlights the need for the number of RANSAC trials to be know in advance, as set-length parallel FOR loops are the original and most efficient use of OpenMP [8]. An upper limit on RANSAC trials can be computed from the first plane found, and this number has an indirect correspondence to the number of points in the point cloud via the inlier ratio. This makes parallel RANSAC trials more promising for parallel speedup relative to point cloud size than parallel plane-finding, and, while more complex, this method had been considered for implementation if parallelizing distance calculations returned disappointing results.

4.4 Distance calculations in Parallel

```
 9: for all x_i ∈ X do                ▷ parallel for shared(thisPoints), private(i)
10:     if distance(thisPlane, x_i) < T then
11:         thisPoints ← thisPoints + x_i                          ▷ critical section
12:     end if
13: end for
```

The distance calculations are fairly independent, with no iteration of the FOR loop relying on the results of a previous one. The critical directive used at the end of the loop ensures thread safety by allowing only one thread at a time to add the indexes of points on the plane to a vector shared between all threads. This is the only piece of synchronization needed in this parallel block; in terms of performance, the distance calculation and comparisons are far more computation-intensive over millions of points than the vector add operation and overhead required to complete it serially, as is reflected in the profiling results.

4.5 Results

Profiling the serial execution of the plane detector gives a clear indication of the most computationally intensive methods. RANSAC takes up 93% of the CPU time, and within it the distance calculations from points to planes specifically. In the program this is done using the Eigen::Hyperplane absDistance() method, a reasonably efficient method mainly taking up CPU time with dot products and other Matrix operations. The sheer number of

Method	CPU %
RANSAC	93.44
- Distance comparison	91.59
- - Distance calculation from point to plane	(89.73)
- - Comparison of distance to threshold	(1.86)
Reading from PLY	5.40
Writing to PLY	0.27

Table 1: Profiling results of key methods

repetitions of this method is the main factor in its large CPU time, even with the program executing on a relatively small 88,435 point model in profiling. To find just the largest plane in this model with a reasonable 400 trials, the distance comparison is called 35 million times. Given there are multiple planes left with a decreasing but still significant number of trials run for each, and the number of trials to achieve the success probability also increases with the number of points in the scene, the calls to this function can be expected to increase exponentially with larger models.

As predicted, the performance gains from parallelizing the distance calculation and comparisons are significant. Due to the random nature of the algorithm giving large variations in execution time, the average of 5 run-times (of the RANSAC method only) were taken for the 8026-point and 88435-point models. The hardware and OpenMP environment were able to provide a maximum of 4 threads, with speedup of 2.16, 3.17 and 3.55 time for 2, 3, and 4 threads respectively. This proves the expected trend of parallelization having increased benefits with more points in the model. However, there are a few quirks to these results, such as 4 threads having minimal speedup over 3 even with 88435 points, and the parallel efficiencies (speedup per thread) for 2 and 3 threads exceeding the theoretical limit of 1. This can be mostly explained by the random nature of the program giving varied execution times, with an aim for future work on this project being large-scale testing with bigger models, more threads and a larger sample size of executions. Testing on different hardware with more threads available should also shed light on the performance plateau observed at 4 threads, whether using larger models will bring the parallel speedup closer to four-fold or demonstrate parallel overhead costs closing in on potential gains at the limit of available threads. Aside from further testing, the parallelization of the plane detector can be

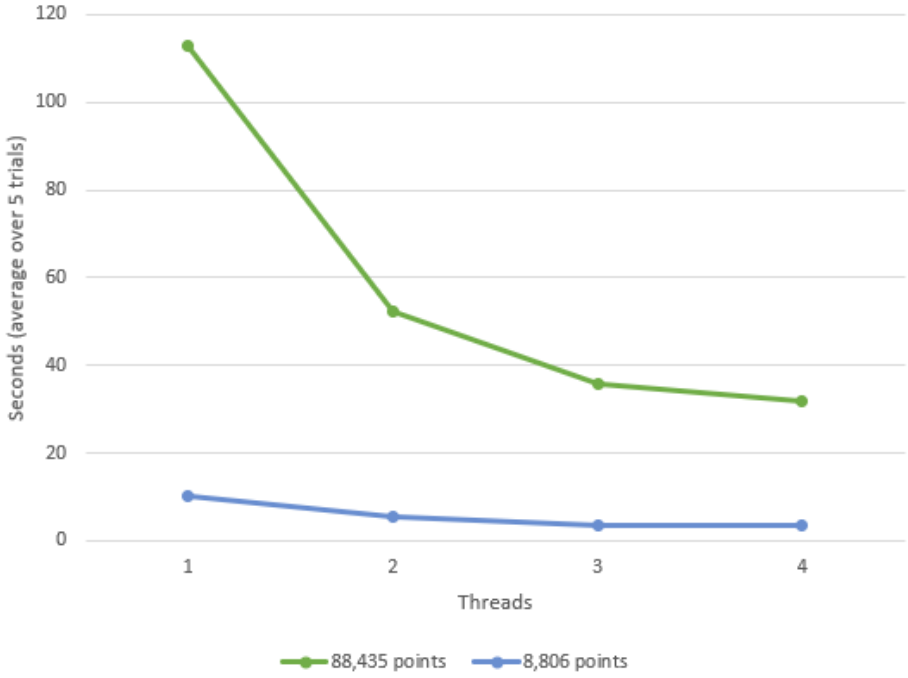

Figure 7: Plotting execution time of RANSAC methods against number of threads used.

deemed complete with such a marked improvement in execution time. The earlier analysis of potential areas of parallel speedup and synchronization costs indicates diminishing returns for further parallelization.

5 Future Work

The future improvements planned for this project move away from parallel computing, as the results above already give a powerful performance boost that should scale with increasing hardware. The following proposals will improve the plane detector's accuracy, serial performance, and its removal of clutter in the scene. Along with these improvements to the algorithm itself, a rendering of the Albany St studio planar model should be produced for the final report, either within the plane detector program using OpenGL or in a separate renderer. A minor improvement to this output would be creating textures for the planes detected taken from the point cloud colours, better

representing the subject visually while maintaining efficiency of computation and storage.

5.1 Point Re-assignment

The largest flaw in the improved RANSAC algorithm used with regards to accuracy is its rigidity to the order in which planes are discovered. Points, once found to be on an optimal plane, are removed from the point cloud entirely for subsequent planes. While this increases performance by reducing the size of the point cloud every iteration, it does prevent some points from being assigned to their closest plane in the final output. Re-assigning points

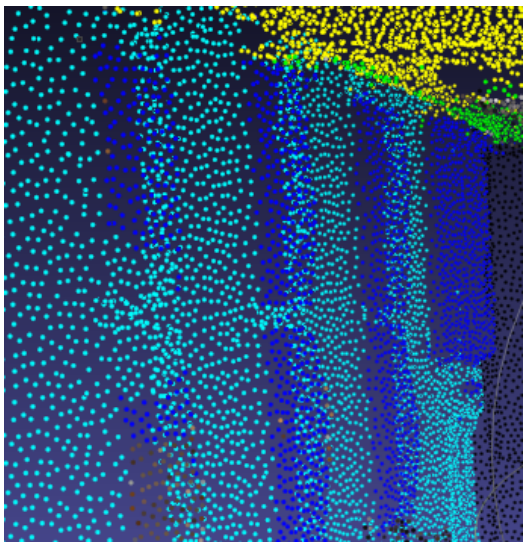

Figure 8: Section of colored points ; the closer dark blue plane should be covering all of the wall protrusions instead of the earlier-found light blue plane.

to their closest plane after all planes have been found is the simplest solution to increase accuracy in a serial discovery of planes, but a naïve comparison of every point with every plane is computationally inefficient and unnecessary. The next goal of this project is to implement point reassignment with algorithms to reduce the number of points and planes compared for acceptable performance.

5.2 Space Subdivision

Octree subdivision of 3D space is an efficient way to encode the approximate closeness of unorganized points. In plane detection, this could reduce the number of comparisons needed for assigning and reassigning points. The capacity to efficiently compute plane intersections with octree bounding boxes is already in the program, as two Eigen::Hyperplane objects can compute their intersections in the model's coordinate space. Finding plane intersections within the model could also reduce the number of unique plane colors needed to distinguish points.

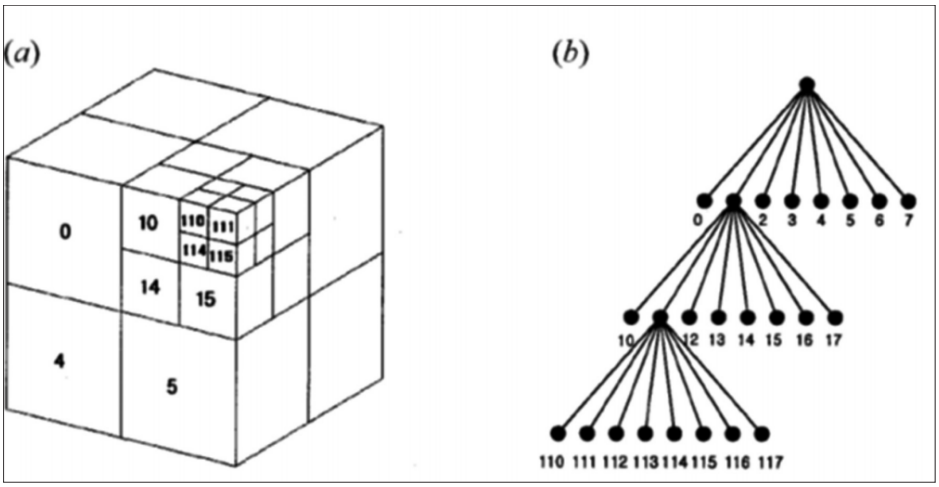

Figure 9: Octree diagram, Alba, Mario, et al. *Filtering vegetation from terrestrial point clouds with low-cost near infrared cameras.* Ital. J. Remote Sens 43 (2011): 55-75.

5.3 Storage of Multiple "Best" Planes

An avenue for further improvement lies in the observation that the minimum number of planes that would be acceptable to find in a scene is quickly noted by the user when viewing the model; the simple cube used to test the basic RANSAC implementation is noted to have 5 sides facing the camera, and the studio model must have at least 6 by the virtue of its closed rectangular geometry. This minimum number of planes could be inputted by the user and used to store as many "best" RANSAC trials for one plane as there are

minimum planes remaining, and using those planes as a starting point for detecting subsequent planes when they still remain viable with overlapping points removed.

5.4 Neural Network noise reduction

A stretch goal of this project would be to implement more accurate noise classification with a neural network. As the clutter in the point cloud is clearly recognized as not part of the subject (for example, the music equipment in the Albany St studio) by the human eye, a neural network mimicking the visual cortex holds promise to do the same without the explicit algorithms required of a Computer Vision approach. A pre-trained model, if enough training and testing point clouds were obtained to give reasonable accuracy, would have minimal performance impact when classifying which points to exclude from the plane detection.

6 Conclusion

In this report I have outlined the benefits of planar models over point clouds for high-fidelity architectural modelling, approaches taken to plane detection in the field and the improvements made to the basic RANSAC plane detector. Auto-generation of parameters has increased accuracy of the planes detected without prior knowledge of the point cloud, and parallelization has given an impressive level of speedup to the program. The future work on this project will consist of further improvements on accuracy, performance and quality of the plane detector's output, until a complete and useful model of the planes in a point cloud of any size can be produced in reasonable time.

References

[1] Fischler, Martin A., and Robert C. Bolles. *Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography.* Communications of the ACM 24.6 (1981): 381-395.

[2] Limberger, Frederico A., and Manuel M. Oliveira. *Real-time detection of planar regions in unorganized point clouds.* Pattern Recognition 48.6 (2015): 2043-2053.

[3] Botterill, Tom, Steven Mills, and Richard D. Green. *New Conditional Sampling Strategies for Speeded-Up RANSAC.* BMVC. 2009.

[4] Nistér, David. *Preemptive RANSAC for live structure and motion estimation.* Machine Vision and Applications 16.5 (2005): 321-329.

[5] Hofer, Manuel, Michael Donoser, and Horst Bischof. *Semi-Global 3D Line Modeling for Incremental Structure-from-Motion.* BMVC. 2014.

[6] Mills, Steven. *Finding Planes In Point Clouds.* University of Otago COSC450 Assignment 2. 2017. https://www.cs.otago.ac.nz/cosc450/assignments/assignment2_2017.pdf

[7] Nistér, David. *Preemptive RANSAC for live structure and motion estimation.* Machine Vision and Applications 16.5 (2005): 321-329.

[8] Ayguadé, Eduard, et al. *A proposal for task parallelism in OpenMP.* International Workshop on OpenMP. Springer, Berlin, Heidelberg, 2007.

Appendix A Aims and Objectives

Original

Aims The aim of this project is to produce a methodology for generating simple, meaningful 3D models of real-world objects with planes, rather than with point clouds. Planar models are accurate and computationally efficient for certain subjects such as buildings, and this project aims to improve the capture and modelling of these (such as the University's model of Albany St recording studio). This methodology should be either applied to existing point cloud models, or 2D images of the object via plane detection. It should improve on current methods in the field in its accuracy, performance and/or flexibility.

Objectives

- background reading; research in the field
- completing the past COSC 450 assignment; set up a basic RANSAC plane detector

- research possible extensions to plane detector; decide on improvements to focus on for accuracy/speed/flexibility (as proposed below):
- remove intermediate point cloud step; implement plane detection directly from 2D images
- improve geometric accuracy; detect and model more complex mathematical forms e.g. paraboloids, spheres
- parallelize plane detection; improve performance (potentially to real-time levels)
- reduce error in plane detection; apply machine learning to reduce point cloud noise

Timeline

- March: complete RANSAC plane detector
- March-April: test and assess plane detector on Albany St studio model
- April: further research in the field; assess improvements for plane detector
- May: implement first improvement
- June: write interim report/presentation
- June-July: implement further improvements
- September: write final report
- October: research potential approaches to unimplemented improvements; add to final report

Revised

Aims The aim of this project is to produce a methodology for generating simple, meaningful 3D models of real-world objects with planes, rather than with point clouds. Planar models are accurate and computationally efficient for certain subjects such as buildings, and this project aims to improve the capture and modelling of these (such as the University's model of Albany St recording studio). This methodology should be applied to existing point cloud models via plane detection. It should yield improved performance over standard RANSAC implementations, and produce a compact representation of the model accurate to key structural points (e.g. architectural components of a room), that also lacks the extraneous object noise of the original.

Objectives

- background reading; research in the field
- completing the past COSC 450 assignment; set up a basic RANSAC plane detector
- research possible extensions to plane detector; decide on improvements to focus on for accuracy/speed/flexibility (as proposed below):
- improve performance; parallelize plane detection
- increase accuracy of plane assignment: reassign points after RANSAC with octree space subdivision and/or plane intersections
- improve noise removal; apply machine learning to classify points not on any plane

Timeline

- April: complete RANSAC plane detector
- April-May: test and assess plane detector on Albany St studio model
- May: further research in the field; assess improvements for plane detector
- May-June: parallelize RANSAC algorithm
- June/July: write Interim report and presentation
- July: profile full Albany St Studio model on lab machine with 1-16 threads, implement space subdivision
- August: use space subdivision for reclaiming points and optimal colour usage
- September: write final report, render results as planes, investigate texture mapping for planes
- October: add discussion of further improvements into final report (texture mapping if not implemented, neural network noise reduction etc)