Honours Project Report

# Spatial Management of African Sites and Heritage

*Author:*
Timothy Trewartha

*Supervisor:*
Prof. Hussein Suleman

| | Category | Min | Max | Chosen |
|---|---|---|---|---|
| 1 | Requirements Analysis and Design | 0 | 20 | 10 |
| 2 | Theoretical Analysis | 0 | 25 | 0 |
| 3 | Experiment Design and Execution | 0 | 20 | 0 |
| 4 | System Development and Implementation | 0 | 15 | 15 |
| 5 | Results Findings and Conclusion | 10 | 20 | 20 |
| 6 | Aim Formulation and Background Work | 10 | 15 | 15 |
| 7 | Quality of Report Writing and Presentation | 10 | | 10 |
| 8 | Adherence to Project Proposal and Quality of Deliverables | 10 | | 10 |
| 9 | Overall General Project Evaluation | 0 | 10 | 0 |
| **Total** | | **80** | | **80** |

University of Cape Town

Department of Computer Science

October 29, 2012

# Abstract

This report explores an aspect of the big data problem faced by the members of the Zamani project. It investigates the application of common big data techniques, such as level of detail and subdivision, to the problem of managing large point clouds, as are commonly produced by laser scanners. These large point clouds typically contain billions of points and are often over 50GB.

Using an indexing based scheme, the system developed is able to quickly and efficiently perform region extractions from previously unstructured point clouds. It is also able to extract the point cloud at varying resolutions, depending on the requirements of the user. For example, during a visualisation one may want to view the cloud at a low resolution. Alternatively, a user may request a sub region of the point cloud at a higher resolution, and send it to a mesher for example. The system is also able to stream the points from a central server to various clients via a network connection.

The key findings of the project are that subdivision and level of detail are two effective techniques for dealing with large point cloud data. The evaluation indicates that the times for typical extractions are reasonable. Another finding is that, when varying the resolution of the extraction using the chosen technique, the time scales linearly, as one would hope.

# Acknowledgements

The author would like to thank and acknowledge the following people for their guidance and assistance during the course of this project. Firstly, Hussein Suleman, for supervising the project and for giving much needed advice and encouragement throughout its course. In addition, he committed much time to reading through and giving feedback on the various iterations, drafts and demonstrations. Secondly, the author would like to thank all the members of the Zamani project and in particular, Heinz Rüther, Roshan Bhurtha and Ralph Schroeder. Many hours were spent in discussion with them about various aspects of the system. Also, Roshan Bhurtha and Ralph Scroeder provided the project with test data and committed much time to explaining the processes involved in laser scanning as well as their day to day work. The author would also like to thank his project partner Michiel Johan Baird for advice and encouragement during the course of the project. Finally the author would like to acknowledge the National Research Foundation (NRF) for their financial support.

# Contents


**1 Introduction** **5**
- 1.1 Big Data . . . 5
- 1.2 Zamani Project . . . 5
- 1.3 General Outline . . . 6
- 1.4 Research Question . . . 6
- 1.5 Report Overview . . . 7

**2 Background** **9**
- 2.1 Introduction . . . 9
- 2.2 Zamani Project . . . 9
- 2.3 File Types . . . 10
- 2.4 Level of Detail . . . 11
- 2.5 Indexing 3D Data . . . 12
- 2.6 Out-of-core Storage . . . 13
- 2.7 Visibility Culling . . . 13
- 2.8 Summary . . . 14

**3 Design** **15**
- 3.1 Introduction . . . 15
- 3.2 Design Goals . . . 15
- 3.3 System Overview . . . 16
- 3.4 Design Principles . . . 16
- 3.5 File-Based Index . . . 17
- 3.6 Partitioning . . . 17
- 3.7 The Index File . . . 19
- 3.8 Binary . . . 19
- 3.9 Multiresolution . . . 19
- 3.10 Indexing Algorithm Summary . . . 20
- 3.11 Extraction Algorithm . . . 21
- 3.12 Further Subdivision . . . 21
- 3.13 Client/Server Streaming . . . 22
- 3.14 Conclusion . . . 23

**4 Implementation** **25**
- 4.1 Introduction . . . 25
- 4.2 Development Environment . . . 25
- 4.3 Indexing . . . 26
- 4.4 Extraction . . . 26
- 4.5 Client/Server . . . 26
- 4.6 Conclusion . . . 26

**5 Evaluation and Results** **27**
5.1 Introduction . . . . . 27
5.2 Binary . . . . . 27
5.3 Comparison with a Naïve Approach . . . . . 28
5.4 Varying Resolution . . . . . 29
5.5 Region Extraction . . . . . 33
5.6 Increasing the Level of Subdivision . . . . . 35
5.7 Streaming . . . . . 37
5.8 Conclusion . . . . . 38

**6 Conclusion and Future Work** **39**

# Chapter 1

# Introduction

## 1.1 Big Data

The term big data refers to collections of datasets that are so large and complex that it becomes difficult to process them using traditional software packages and database management tools. Over the past few years the amount of data being produced and processed has increased rapidly. This data is often produced by wireless sensor networks, software logs, cameras, microphones, information-sensing mobile devices or laser scanners, and can sometimes exceed exabytes of data [Watters, 2010]. The size of the data can often limit the amount of processing that can be performed, due to the amount of time it takes to access the data. The size of the data also means that transferring it to main memory for quick and efficient access is not possible. As a result, novel approaches need to be developed to enable quick and efficient access to this big data.

This project investigates one aspect of the big data problem, namely, dealing with the vast amount of data produced by laser scanners. Recent advances in 3D laser scanning technology have seen a rapid increase in the amount of data produced by 3D scanners. These scanners are now able to capture points with sub-millimetre accuracy. This results in datasets that are hundreds of gigabytes in size and contain billions of points. Managing these datasets is difficult, and it is often impossible to view the entire scan, even with modern technology and advanced rendering methods. The approach taken in this project is to decrease the size and complexity of the problem by enabling users to perform quick and efficient extractions from the point cloud, at a specified resolution.

## 1.2 Zamani Project

The number of projects using laser scanners has increased rapidly over the past few years, ranging from acquiring small mechanical parts to the acquisition of archaeological sites or even entire cities [Akbarzadeh et al., 2006]. One such project that aims to capture the spatial domain of African cultural heritage sites using laser scanners is the Zamani project. It was started in the Department of Geomatics at the University of Cape Town and is led by Professor Heinz Rüther [Ruther et al., 2012]. The project has produced detailed laser scans of over 40 sites all over Africa, such as the Gede ruins in Kenya, the Great Zimbabwe ruins and the Songo Mnara Palace in Tanzania.

In discussions with the members of the Zamani project they indicated that they have difficulties managing the vast scale of data produced by their laser scanners. These laser scanned models of cultural heritage sites are often very large, some of them containing over 8 billion points. In this project, ways of managing these vast point clouds are investigated. Even using

advanced technology and well developed methods for point cloud rendering, it is not possible to view these point clouds interactively.

## 1.3 General Outline

The project is divided into two major components. The first is a workflow management system developed by Michiel Johan Baird. The second is an indexing and streaming system for large point clouds developed by Timothy Trewartha. These two components aim to solve some of the challenges faced by the members of the Zamani Project.

### Workflow Managment System

In order to assist with the difficulty of managing the data and the tasks involved in creating the heritage data, a Workflow Management System was developed, following the requirements of the Zamani team. This system is able to handle both automated and manual user tasks. The aim is to increase the efficiency of the task execution and improve reusability between different heritage sites.

### Point Cloud Indexing

The other component tackles the difficulty involved in managing large point clouds. The approach taken is to build an index for the point cloud. This index allows for efficient region extraction. It also allows the user to specify the resolution at which they wish the extraction to be performed. The system also enables the extractions from the point cloud index to be streamed from a central server. For example, using the index a low resolution representation can be efficiently extracted and sent to the client. Alternatively, the user may request a high resolution extraction of a subregion.

These two separate components can interact in a number of ways. For example, at various stages the Workflow Management System requires point clouds as input to certain processes. This input can be efficiently extracted from the indexed point cloud using the point cloud streaming system. Alternatively, if the Workflow Managment System produces a point cloud as output (after cleaning for example) it can be sent to the indexer so it can be more efficiently accessed further down the workflow pipeline.

## 1.4 Research Question

The key research question posed for this project is:

> "Is it feasible to support efficient multiresolution extractions from large point clouds containing billions of points?"

Ideally it should be able to perform these extractions quickly and efficiently, even when the underlying point cloud is very large. The term 'efficient' is of course open to some interpretation. Typical processes on these point clouds (such as registration) and the resulting models (such as texturing) can often take several hours [Ruther et al., 2012]. Consequently, in this definition of efficiency, it is hoped that typical extractions will take on the order of a few seconds to several minutes.

## 1.5 Report Overview

This report is structured as follows: firstly, in the background chapter, important background information is presented, as well as a review of relevant literature. In the design chapter, the design principles are discussed as well as the design of the system developed. The implementation chapter discusses the development environment and gives information about the code written. Finally, the evaluation chapter presents a thorough evaluation of the capabilities of the system developed.

# Chapter 2

# Background

## 2.1 Introduction

This background chapter discusses important information which is relevant to the system developed as part of the project. First, the workflow process of the Zamani project is discussed, since, in developing the system it was important to understand the needs of the members of the project. Following this, the different file types used are discussed, and then, following that some general techniques for dealing with big data and in particular the indexing of 3D data are discussed. Finally, some common techniques for dealing with data too large for main memory are presented.

## 2.2 Zamani Project

The Zamani Project was started by the Department of Geomatics at the University of Cape Town in 2005 [Ruther et al., 2012]. It aims to preserve African cultural heritage by documenting heritage sites. The project has produced detailed laser scans of over 40 sites all over Africa.

Documenting a site involves a lengthy process of scanning, cleaning, registering, downsampling, and meshing. First, multiple scans of the site must be obtained to capture the full spatial domain of the site. For example, when documenting the Gede Ruins in Kenya, 423 individual scans were produced in order to capture the full extent of the site, including the inside of buildings such as the mosque and palace (this data was part of the data provided for testing purposes). These scans must then be cleaned in order to remove unwanted objects such as trees or people. After cleaning each individual scan, a registered point cloud is produced from the individual scans using Leica Cyclone©. Given that each individual scan contains around 8 million points, this registered point cloud is huge, often containing billions of points. For the Gede Ruins, 4 main registrations were produced: the Mosque (59 million points), the Small Mosque (224 million points), the Area (883 million points) and one containing all scans (2.4 billion points).

Given the vast sizes of these point clouds, it is extremely difficult to work with them. Even on modern hardware using advanced rendering methods, it is difficult to view these point clouds and interact with them. Most point cloud algorithms, such as K-Nearest Neighbour, also struggle, leading to long running times. Many software packages also cannot cope, as the size of the point clouds far exceeds the amount of memory available, even on high end machines. For example, just storing the raw points in the full registration requires over 40GB of memory. As a result, it is necessary at this stage to decimate the original point cloud and produce a lower resolution point cloud that can then be used to produce the final mesh. These triangle based

meshes are preferred to point clouds as rendering methods for meshes are more developed and a mesh more closely represents the actual surface than just storing the points. Additionally, the mesh allows for texturing, which provides a higher level of visual detail. The final mesh typically contains only a few million points, not billions like the underlying point cloud. These mesh based models can then be used to produce many other data objects, such as plans, sections, elevations, virtual tours, digital terrain models or geographic information systems.

## 2.3 File Types

There are many different file types that are used for storing point cloud data. Some of them simply store the raw points while others contain additional information, ranging from basic statistics such as the number of points to more advanced information contained in some of the proprietary Leica© formats. Some basic information on the file types, gathered from the members of the Zamani project, is provided here.

### ZFS

The scanners used by the Zamani project (such as the Leica Geosystems HDS 6000) produce the output from each scan as a ZFS file. This ZFS file stores the raw information from the scanner in binary format. Essentially, it stores the x, y and z coordinates of the point as well as any additional colour information or simply the intensity captured by the scanner. Since the format is not open, direct interaction with these raw files is avoided in this project.

### IMP

IMP files are another proprietary format used by Leica© software. The IMP files are database files that typically contain all the information for one site. The ZFS files are imported, cleaned and registered. The registered scans are stored in these IMP files, so they are typically very large. Once again, this is a proprietary format so it is not directly used in this project. Point clouds stored in this format were exported to a free format whenever necessary.

### XYZ

XYZ is the simplest open format for storing point cloud information. It stores only the x, y and z coordinates of each point. It does not store any additional information, such as a bounding box, the number of points or any intensity or colour information. This file format, although simple, is sufficient for the purposes of this project. Although having the point cloud size or the dimensions of its bounding box would be useful additional information, it is not required.

### PTX

PTX is an open format for storing point cloud information. Like the XYZ format, it stores the x, y and z coordinates of each point. In addition, however, it stores either R, G, B colour values or a single intensity value with each point. It also contains additional header information, such as the number of points, and x, y, z offset (for registration) and two transformation matrices (also for registration). This file format is also very easy to parse and is suitable for this project. However, any colour information is disregarded as the index is not designed to incorporate that information at this stage.

### PCD

PCD is a simple point cloud format that is provided by the PCL C++ library [PCL, 2011] . It stores the raw points, as well as some auxiliary information such as the size of the cloud. It also allows the points to be stored in binary format. However, there was no reason to use the PCD format over PTX/XYZ so it is simply noted here for the sake of completeness.

### PLY

The final format that is used extensively by the Zamani project and is of interest is the PLY format. This is typically the final 3D output format produced. It is a mesh, rather than a point cloud and contains the vertices and polygons that make up the final, cleaned and registered model. Generally it has far fewer primitives when compared to the underlying point cloud and so rendering and interacting with this final output model is generally not problematic. For the purposes of this project, it will not be necessary to interact with these PLY models as this system fits in at the intermediary stage, after registration and before downsampling and meshing.

## 2.4 Level of Detail

Level of Detail (LoD) is a well developed technique for dealing with complex scenes and detailed objects, containing millions, or even billions, of primitives. The basic idea is to avoid dealing with the full complexity of the scene and deal with only a well-defined subset. For example, when applying level of detail to rendering, the full original detail of the model is not required if it is being viewed from a distance. It is only when zoomed in closely that one needs to increase the amount of represented detail, and, since at this point one is only viewing a small portion of the model, one does not require the entire dataset to be loaded into memory and each primitive rendered.

As early as 2000, systems were being developed for handling models with hundreds of millions of primitives, perhaps the most notable of which is QSplat [Rusinkiewicz and Levoy, 2000]. This system made use of a bounding sphere hierarchy to implement a level of detail scheme and achieve a significant speedup. For example, they were able to render models containing over 8 million points interactively using a multiresolution scheme.

Around the same time, an alternative approach was suggested for rendering complex geometric objects. This is described in the paper 'Surfels: Surface Elements as Rendering Primitives' [Pfister et al., 2000]. Surfel is short for surface element; surfels are point primitives but without any explicit connectivity as is common in mesh based schemes. To enable interactive rendering, the authors chose to use an octree as a multiresolution hierarchical structure. Although the results they achieve allow for interactive frame-rates, the sizes of the models are small compared with some of the sites documented by the Zamani project.

More recently, level of detail schemes have made use of other data structures such as octrees. Wand et. al. describe a new, out-of-core multiresolution data structure for real-time visualization and interactive editing of large point clouds [Wand et al., 2007]. Their chosen data structure consists of a dynamic octree with a grid-quantization-based dynamic multiresolution representation in each inner node. The octree is able to easily handle dynamic operations due to its regular structure, and using this data structure the authors were able to achieve real-time walkthroughs and interactive modifications of a dataset containing 2.2 billion points and

totalling 63.5GB, although it took over 14 hours to build the required data structure.

Gobetti and Marton also present an effective level of detail system for dealing with large point clouds[Gobbetti and Marton, 2004]. The multiresolution approach creates a hierarchy of the point cloud as an offline process, by reordering and clustering them into point clouds of approximately constant size arranged in a binary tree. It is thus possible to obtain the required level of detail by accumulating point clouds as the hierarchy is traversed top-down. The root node is thus the coarsest available model.

Finally, Cigoni et al. present another spatial partitioning multiresolution data structure in their paper [Cignoni et al., 2008]. The authors use a regular conformal hierarchy of tetrahedra to spatially partition the model. The resulting technique is fully adaptive and is able to retain all the original topological and geometrical detail, even for massive datasets. Additionally, it is not limited to meshes with a particular subdivision connectivity and is strongly GPU bound. It is over one order of magnitude faster than previously existing adaptive tessellation solutions since the data structure is able to exploit on-board caching, out-of-core representation and prefetching for efficient, real-time rendering.

## 2.5 Indexing 3D Data

As well as developing efficient level-of-detail hierarchies to allow for real-time rendering of large models, it is also important to consider the indexing of the 3D data. Although there are many different spatial indices, such as kd-trees and cell-trees, they are not well suited to 3D applications [Zhu et al., 2007]. Based on the idea of B-trees, Guttman presented the R-tree as a new way of indexing multi-dimensional information [Guttman, 1984]. Subsequent modifications to the R-tree include the $R^+$-tree, which allows one object to exist in multiple nodes [Sellis et al., 1987] and the $R^*$-tree, which has better clustering [Beckmann et al., 1990]. More recently, Zhu et al. have introduced a new spatial cluster grouping algorithm (k-means clustering) that uses 3D overlap and coverage volume as well as the minimum bounding box shape as the integrative grouping criteria [Zhu et al., 2007]. Using these methods, the authors were able to gain significantly better performance when querying the spatial data.

A more mathematical approach to indexing 3D data is based on Hilbert Space Filling Curves. These can be used to partition the dataset, which can then be stored in a spatially indexed relational database [Wang and Shan, 2005]. In 3-Dimensions one can think of a Hilbert Curve as being a curve that passes through every point in the specified 3-Dimensional space. Although this seems counterintuitive, it is possible since the two spaces have the same cardinality. This provides a way to partition the space, as well as a mapping between a 1D space (for example the hard disk) and the 3D space in which the points are located. Additionally, Hilbert Curves have the advantage that points that are close together on the curve, are also close together in the 3D space. It is possible to consider other Space Filling Curves (there are many) but both mathematical analysis and practical applications suggest that the Hilbert curve has the best clustering ability and performance in data retrieval and response time [Faloutsos and Roseman, 1989]. In the paper by Wang J. and Shan J, this technique is described and used to gain better query performance on large 3D data consisting of millions of points [Wang and Shan, 2005]. It is not certain if this method will, however, scale to billions of points, as is required by the Zamani models.

There has also been some effort to find a way of integrating the level-of-detail hierarchies

discussed in the previous section with different indexing methods. Of particular interest is the paper by Kofler that attempted to combine the R-tree with LOD (level-of-detail), and presented the LOD-R-tree method in which the level of the R-tree represents the required level-of-detail representation [Kofler et al., 2000]. Zlatnova also tried to find a similar way of uniting R-trees with LOD and put forward various grouping methods that take into account location, shape and altitude [Zlatanova, 2000]. It seems, however, that this area has not been investigated thoroughly and there is still more work to be done, as noted by Zhu et al. [Zhu et al., 2007].

## 2.6 Out-of-core Storage

A common problem encountered when dealing with large amounts of data, such as with these vast point clouds, is the inability to store all the data locally in random access memory [Cignoni et al., 2008]. Consequently, all algorithms dealing with interactive rendering must take into account the location of the data that they are accessing, as access to data on hard-disk presents a significant bottleneck. Such algorithms are termed out-of-core algorithms, meaning that a portion of the working dataset must be stored on the hard-disk. The octree data structure previously discussed is fairly efficient, even when some data is out-of-core [Wand et al., 2007]. Standard virtualisation techniques are applied to the given data structure. This works and is efficient because only those nodes that are needed for rendering must be loaded into main memory. Given a certain required level-of-detail, only a small number of the nodes needs to be accessed while a large fraction of the data structure remains unused. The authors also built in two methods to support specific out-of-core operations: *fetch* and *access*. *Fetch* indicates that a node is to be used in the near future and should hence be moved to main memory (this is handled by a separate thread to hide disk-access latencies). *Access* asserts that the data is already in main memory and hence readily accessible. The authors opted for a Least-Recently-Used policy to swap out unused nodes to disk when the memory cache is full. Determining the correct block size is also an important parameter for gaining optimal efficiency.

## 2.7 Visibility Culling

Visibility culling is another important technique for achieving real-time rendering of large datasets. Visibility culling refers to the fast exclusion of portions of the data that are not visible from the current point of view. Greene et. al. describe a general algorithm to discard primitives that are blocked by closer geometry using a hierarchical Z-buffer [Greene et al., 1993]. This is a type of visibility culling known as occlusion culling. The methods presented in this paper performed well regarding the two key criteria for an ideal visibility algorithm, namely that it should quickly reject most of the hidden geometry in the model and, secondly, it should exploit the spatial and temporal coherence of the images being generated. This was a considerable step forward given that previous methods had only been able to satisfy either one or other of these criteria but not both. The key insight on their part was to use two hierarchical data structures: an object-space octree and an image-space Z-pyramid, thus making it possible to reject hidden geometry very rapidly.

In addition to occlusion culling, two other important types of visibility culling are frustrum and backface culling. Frustrum culling is the removal of objects that lie outside the view frustrum. As previously mentioned, the QSplat system implements both these techniques [Rusinkiewicz and Levoy, 2000]. The authors note that backface culling of primitives is commonly implemented in hardware, and Kumar and Manocha have presented an algorithm for hierarchical backface culling based on cones of normals [Kumar et al., 1996]. Finally, most of the speed

benefit from frustrum culling come as a natural consequence of implementing the data structure correctly.

## 2.8 Summary

In this chapter the background to the Zamani project, their workflow, and the various file types used for point clouds have been discussed. Various techniques such as level of detail and out-of-core storage were also presented, as they are particularly relevant to the problem of dealing with large point clouds. Additionally, various methods for indexing of 3D data used in the literature were presented, and visibility culling was discussed as a relatively easy way to boost performance when viewing 3D scenes. These various methods and approaches are considered and applied in the design and implementation sections of this project report.

# Chapter 3

# Design

## 3.1 Introduction

The design chapter provides an overview of the system developed. It also details the various algorithms used and describes them step by step.

## 3.2 Design Goals

The main aim of this project was to build a simple and efficient index for large, unstructured point cloud files such as those that are produced by the Zamani project. The index has two main objectives. Firstly, it should enable efficient region extraction. Secondly, it should allow for regions of the point cloud to be obtained at varying resolutions, depending on the level of detail required.

- Region Extraction: being able to extract a sub-region from the point cloud is an important use case. Often one needs only to visualise or interact with a small portion of the cloud at a time. Alternatively, a subregion of the cloud may be selected for meshing at a higher resolution, rather than the entire cloud at a lower resolution.

- Varying Resolution: Given the vast size of the point clouds, it is not possible to view the entire point cloud at full resolution, even with sophisticated rendering methods and hardware. Consequently, the index should enable the extraction of a portion of the point cloud at a specified resolution. For example, one may view the entire site at a low resolution to get an idea of what it looks like. One may then use the index to extract a smaller region, but at a higher resolution.

The index developed should support both of these important use cases. A brute force region extraction from an unstructured point cloud is naturally very expensive as it requires that every point be checked to see if it falls within the specified region. A more efficient region extraction would use the index to quickly disregard large sets of points that are completely outside the required region. The points inside the region could thus be quickly found and returned.

Similarly, a brute force downsampling on the large unstructured point cloud is infeasible. This is because it requires the entire cloud to be processed, stored in memory, and then downsampled. A more efficient index based solution would allow a representative point cloud to be quickly extracted. If more detail is required, it can still be obtained by requesting a higher resolution model from the index.

## 3.3 System Overview

In the proposal document the two separate components of the resulting system were outlined. These two components are:

- The workflow management system, developed by Michiel Johan Baird
- The point cloud indexing system, developed by Timothy Trewartha

These two components are easily separable and there is no specific integration required. Within the context of the Geomatics department and the Zamani project, the two components could interact as illustrated in Figure 3.1.

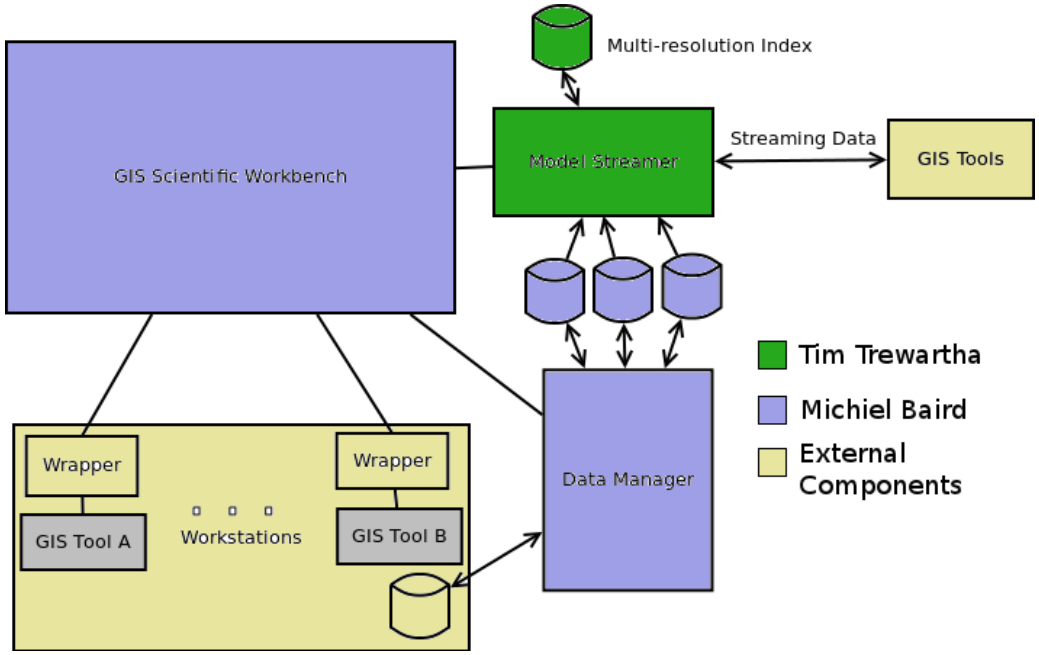


Figure 3.1: The different components of the project

As an example, a researcher may schedule a cleaning task. This task will be assigned to somebody. Once the individual point clouds have been cleaned and registered, they can be indexed for future use by the model streamer. At any time it will be possible to obtain the original point cloud at some specified resolution, or to extract a small region of the point cloud at full resolution. The extracted points can then either be viewed, or fed as input to some other process, such as a meshing procedure.

## 3.4 Design Principles

When designing an index it is important to consider the final goal of our indexing system. As already mentioned, this index aims to enable fast region extraction at varying resolutions. The speedup gained from any index typically comes as a result of the index being built such that it contains approximate solutions to the types of expected queries. The exact solution to the query is then constructed from these approximations, which is more efficient than searching the entire information space. For example, with the region extraction the approximations could be certain subregions that are either fully contained in the required region, partially contained or completely outside.

This type of indexing, where the index is built based on expected queries, is know as *request-oriented indexing* [Soergel, 1985]. It differs from other indexing methods such as *entity-oriented indexing* where the index is based only on properties of the items to be indexed and their properties. There are also several other commonly used indexing methods such as *probabilistic indexing* and *derivative indexing* [Soergel, 1985]. *Request-oriented indexing* was chosen as the method of preference simply because in this case there are several well-defined use cases, so it is possible to tailor the properties of the index to maximise performance based on those use-cases.

## 3.5 File-Based Index

There are many ways to approach any indexing problem. With a database index one constructs a data structure that improves the speed of data retrieval operations at the cost of increased storage space and slower writes. The aim is to be able to perform quick look ups without having to process the entire dataset, since for large datasets this is very time consuming. In this application it is also very important to be able to disregard large portions of our data when presented with a particular request.

Many of the techniques from database theory, such as duplication and partitioning, are applicable. The design decision to use a file-based index was made, primarily due to the simplicity of writing to and reading from these index files. Also, the performance is reasonably good and there is little overhead involved with opening and closing the files. Ideally, the performance of interacting directly with the disk (without the use of the file interface) should be investigated. However, it was decided that this was outside the scope of this project.

## 3.6 Partitioning

The assumption for the indexing process is that one starts with an unstructured point cloud, stored in either XYZ or PTX format. That is, one has very little additional information about the cloud, such as the bounding box or point distribution. Given that these files contain billions of points, linear time complexity is very expensive. For example, a simple count of the number of lines in the file *gede.xyz*, which contains 2,403,821,971 points, takes 11 minutes. The first stage of the indexing process is to partition the cloud into regions that can be queried independently. Region extractions can then be built up from this partitioning.

### Calculating Bounding Box

Before one is able to partition the cloud into regions, one first needs to calculate the bounding box. This is done as part of a first pass through the file containing the cloud to be indexed. On reading each point, one can simply expand the bounding box to ensure that it contains all the previously seen points. At the end of the first pass, one will have values for $x_{min}$, $x_{max}$, $y_{min}$, $y_{max}$, $z_{min}$ and $z_{max}$, which represent the bounding box of the cloud.

### Building the Regions

Once the bounding box has been calculated, it is possible to begin subdividing our region. The region is divided into regular blocks of equal size as illustrated in Figure 3.2. The size of the block is an important parameter in the system as it determines the number of blocks, as well as the number of points in each subregion. For now, all three dimensions of the block are

denoted as $r$ (so it is a cube). The importance of this parameter on performance is investigated.

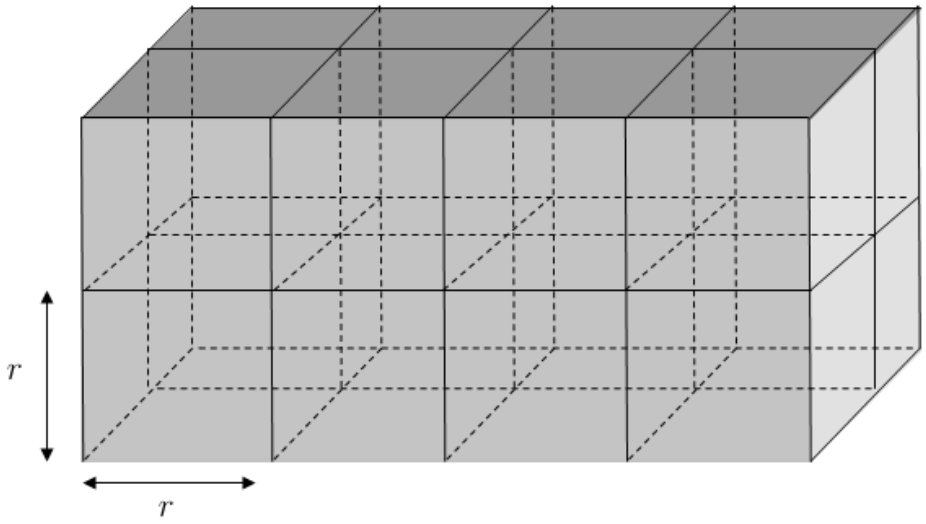


Figure 3.2: The point cloud is subdivided into subregions of equal size

Once the region has been partitioned, it is necessary to process each of the points in the original unstructured point cloud and determine in which region it lies. Since the partitioning provided by this scheme is regular, similar to that of an octree, it is possible to explicitly determine to which region a particular point belongs. The advantage of having an explicit formula is that it is very efficient to evaluate, and does not require any case checking. The regions are numbered sequentially starting at 0. Since the points are in 3D space, to determine which region they belong to it is necessary to perform a flattening operation. That is, one finds the difference between each coordinate and one of the corners of the bounding box, and then multiply by the number of regions along each axis. The details are given in the following code snippet:

```
int xindex = (x-min_x)/resolution;
int yindex = (y-min_y)/resolution;
int zindex = (z-min_z)/resolution;
int pos = ymult*zmult*xindex + zmult*yindex + zindex;
```

Once one has determined to which region a point belongs, it can be written out to a new file for all points in that region. If one subsequently requires the points in a given region, one simply has to open those files that are relevant for the required region and read those points, without having to process millions of other points that are completely outside the region. Thus, as will be seen, a significant speedup is gained from this partitioning.
Since the point distribution is fairly irregular, there is a significant range in the number of points in each region. In particular, many regions remain empty. In this case it is not necessary to create a file for that region. The number of points in each region can also have a significant impact on performance. In particular, if one region contains too many points, the region extraction would degenerate to a worst case brute-force extraction, where each point is checked for inclusion in the region. The ideal number of points in each region is also investigated. Related work suggests that performance degrades significantly once the number of points in each region exceeds 500,000 [Wand et al., 2007].

## 3.7 The Index File

The partitioning scheme described in the previous section creates a regular subdivision of the point cloud, creates a file for each region and writes all points in that region to that file. It is important to keep track of which files contain the points from which region. The index file describes, in a simple format, the bounding box of each region, the name of the file containing the points in that region as well as the number of points in that region. If the region is empty, it is recorded as containing 0 points. However no file is created for this empty region. The first line of the file also lists the number of points in the entire cloud. This meta information about the cloud is important as it helps us to optimise the multiresolution region extraction. The first few lines of a typical index file are shown in Figure 3.3.

```
1 59242631
2 -87.4886322 -36.86976624 30.03672791 79.67521667 -7.496688843 8.083267212
3 -87.4886322 -83.4886322 30.03672791 34.03672791 -7.496688843 -3.496688843 s0.xyz 0
4 -87.4886322 -83.4886322 30.03672791 34.03672791 -3.496688843 0.5033111572 s1.xyz 0
5 -87.4886322 -83.4886322 30.03672791 34.03672791 0.5033111572 4.503311157 s2.xyz 0
6 -87.4886322 -83.4886322 30.03672791 34.03672791 4.503311157 8.503311157 s3.xyz 0
7 -87.4886322 -83.4886322 34.03672791 38.03672791 -7.496688843 -3.496688843 s4.xyz 0
8 -87.4886322 -83.4886322 34.03672791 38.03672791 -3.496688843 0.5033111572 s5.xyz 0
9 -87.4886322 -83.4886322 34.03672791 38.03672791 0.5033111572 4.503311157 s6.xyz 0
10 -87.4886322 -83.4886322 34.03672791 38.03672791 4.503311157 8.503311157 s7.xyz 0
11 -87.4886322 -83.4886322 38.03672791 42.03672791 -7.496688843 -3.496688843 s8.xyz 0
12 -87.4886322 -83.4886322 38.03672791 42.03672791 -3.496688843 0.5033111572 s9.xyz 0
13 -87.4886322 -83.4886322 38.03672791 42.03672791 0.5033111572 4.503311157 s10.xyz 0
14 -87.4886322 -83.4886322 38.03672791 42.03672791 4.503311157 8.503311157 s11.xyz 0
15 -87.4886322 -83.4886322 42.03672791 46.03672791 -7.496688843 -3.496688843 s12.xyz 0
16 -87.4886322 -83.4886322 42.03672791 46.03672791 -3.496688843 0.5033111572 s13.xyz 17025
17 -87.4886322 -83.4886322 42.03672791 46.03672791 0.5033111572 4.503311157 s14.xyz 0
18 -87.4886322 -83.4886322 42.03672791 46.03672791 4.503311157 8.503311157 s15.xyz 0
19 -87.4886322 -83.4886322 46.03672791 50.03672791 -7.496688843 -3.496688843 s16.xyz 0
20 -87.4886322 -83.4886322 46.03672791 50.03672791 -3.496688843 0.5033111572 s17.xyz 353732
21 -87.4886322 -83.4886322 46.03672791 50.03672791 0.5033111572 4.503311157 s18.xyz 80169
22 -87.4886322 -83.4886322 46.03672791 50.03672791 4.503311157 8.503311157 s19.xyz 0
```

Figure 3.3: The first few lines of a typical index file

## 3.8 Binary

In general, reading binary files is significantly faster than reading a text file using the ASCII character set, for example. In our case it will be necessary to read millions of points as fast as possible. If the points are stored in a text file, reading these points will take significantly longer. This is because there are many additional steps involved in reading the numbers, such as parsing the digits and the sign of the number and separating the integer from the fractional part. In several tests, it was observed that an average of 5 times speedup could be achieved by storing the points in a binary format rather than in a text file. Consequently, the decision was made to store the raw points in binary. However, the index file is still kept as a traditional text file. This has advantages as it is human-readable. Since the file is not large, it is feasible to read it quickly without using binary format.

## 3.9 Multiresolution

The partitioning stage is important as it allows us to perform region extractions efficiently. However, it does not enable one to extract the cloud at varying resolutions. Since these clouds are very dense, being able to obtain the cloud at a lower resolution is very useful. The approach taken to building a multiresolution index is a simple reordering of the points. In the original raw point cloud file the order in which the points occurred in the file was irrelevant. By using a simple reordering, it is possible to gain an effective way to extract a multiresolution representation of the underlying point cloud. The order in which the points occur in the new indexed file now gives us implicit additional information. This is ideal, since it does not increase

the size of the index, and it does not add overhead to reading the points.
The aim is to construct a reordering of the points within in each region such that if the original region is required at half resolution, only the first half of the points are read in. The challenge is to ensure that, in reading the first half of the file (and thus ignoring the second half), one obtains a representative sample. This problem can be approached in several ways. One is to simply use a randomisation algorithm and a Russian roulette scheme [Gobbetti and Marton, 2004]. The problem with this scheme as noted in the paper is sometimes the points are not distributed evenly. As a result, several alternatives were investigated. The chosen approach is to process the point cloud region by region (as constructed in the previous stage). For each region, one reads in the points and builds an octree structure that contains points only in the leaf nodes. One can then iterate over the leaf nodes of the octree in turn, writing out a point to the new restructured file and then moving to the next leaf. Since one is moving from one leaf to another in order, a good point distribution in the region is ensured. As an additional step, the points from each leaf node are written out randomly. Without this, the resulting low resolution cloud will have clustering artefacts. That is, one will have many points close together and then a large region without any points. Figure 3.4 shows the results when the chosen multiresolution scheme is applied to a region extraction of the Gede Ruins.

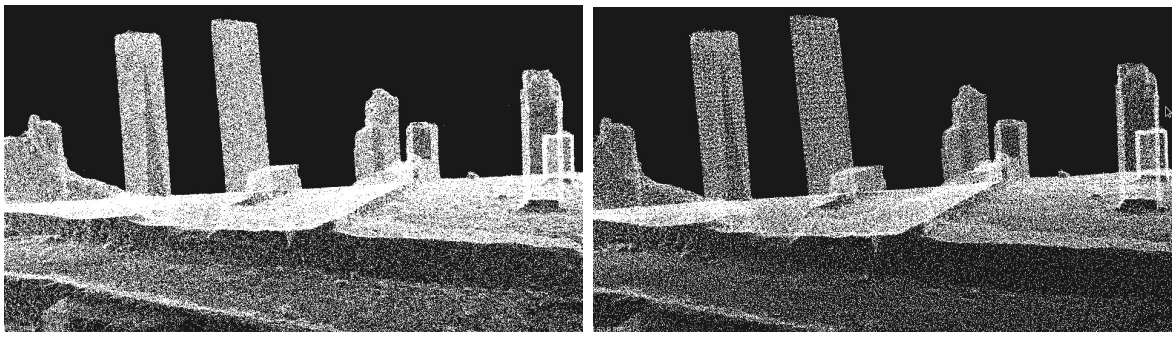

Figure 3.4: The same region represented at two different resolutions. The image on the left is at full resolution (10,621,148 points) and the image on the right is at 5% resolution (622,732 points)

## 3.10 Indexing Algorithm Summary

Having described the various stages of the indexing process, a summary is given here. The input to the indexing algorithm is an unstructured point cloud. The result is the indexed point cloud, stored in a user specified directory.

1. First the bounding box is calculated, as described earlier.
2. Next the region is subdivided and the index file is created, specifying the regions and the file corresponding to each region.
3. Since the entire point cloud is too large to fit into memory, as points are read from the input file they are written into their appropriate subregion files.
4. Once this subdivision process has been completed, the multiresolution reordering of points is performed. For each file:
   (a) All the points are read in and an octree is constructed, containing the points in the leaf nodes.

(b) The points are written sequentially into the new reordered file from alternating leaves of the octree to ensure that they are evenly distributed in space.

Once this algorithm is complete the point cloud has been successfully indexed. How to use this index to perform multiresolution extractions is detailed in the next section.

## 3.11 Extraction Algorithm

Given the indexing scheme described in the previous section, the extraction algorithm is now outlined. The parameters provided to the algorithm are the dimensions of the region for extraction, as well as the resolution at which the extraction is to be performed. Given these parameters, the algorithm for performing the extraction is as follows:

1. Read the index file and determine which regions are:

   (a) Completely excluded

   (b) Partially contained

   (c) Fully contained

   If a region is outside the region for extraction, it is ignored. If it is partially contained, the filename containing those points is added to a vector. If it is fully contained the filename is added to a different vector.

2. Next the points from those files that are fully contained in the region for extraction are read. If the full resolution is required, then all points are read. Otherwise only the fraction of the points that are required are read (for example, if `resolution=0.5`, it is only necessary to read the first half of the file). This is possible due to the reordering of points, which occurred during the indexing stage.

3. Finally, the files that are partially contained are processed. A point is kept if it is within the required region. If not it is discarded. If the extraction is to be performed at the highest resolution, then the entire file must be read. If not, one can keep reading until a significant fraction of the points have been accumulated that fall inside the specified region (the fraction is again represented by the resolution). Note, however, that it is not possible to simply read the first fraction of the file as with the files that are fully contained since there is no guarantee that these points will be in the specified region. This is why processing the partially-contained files takes significantly longer, and the number of large, partially contained files should be minimised.

4. After the points have been extracted, the cloud can be viewed or the extraction can be fed to some other process.

Using this algorithm, extractions can be efficiently performed even on large point clouds, provided the index had been created as a pre-process.

## 3.12 Further Subdivision

After the initial indexing phase, it is common that some files have very many points (even millions of points in some cases) while others have very few, due to the varying density of the point cloud. It is important to avoid having large files with many points as these will result in a significant performance hit if they are partially contained in a region for extraction. It was

thus suggested to use a variable amount of subdivision in our scheme. That is, if a particular file contains too many points, it is further subdivided into smaller regions. This process is continued recursively until the number of points in each region falls below a certain threshold. The threshold is determined empirically. The following algorithm is used to obtain the further subdivision:

1. Read the index file and determine which regions have more points than the specified threshold. If it is below the threshold, nothing needs to be done.
2. If the number of points is above the threshold, split the region into 8 separate regions (as with octree subdivision) and create 8 new files for the new subdivision.
3. Read the points from the original file and determine in which region they fall, writing them to the appropriate file.
4. Write the new regions and new file names to the index file.
5. Repeat until no region has more points than the specified threshold.

Using this algorithm, one is able to increase the level of subdivision until there are no more large files with many points that would adversely affect the performance. However, there is a natural trade-off between the size of the index file and the level of subdivision, which must be taken into account.

## 3.13 Client/Server Streaming

Having developed the indexing and extraction algorithms it is possible to deploy them in a client/server context to allow streaming of point clouds. Clients can connect to the server and make requests. The server will then perform the extraction and stream the points to the client. Due to the design of the index it is easy to stream points while the extraction is still being performed. This means that the client can still examine the point cloud, even if the entire transfer has not yet been completed. Also, if the points are being fed to some other process, such as meshing, the meshing process can begin even before the full extraction is complete. This is due to the subdivision in the indexing scheme. Figure 3.5 shows the simple client/server protocol implemented for this project.

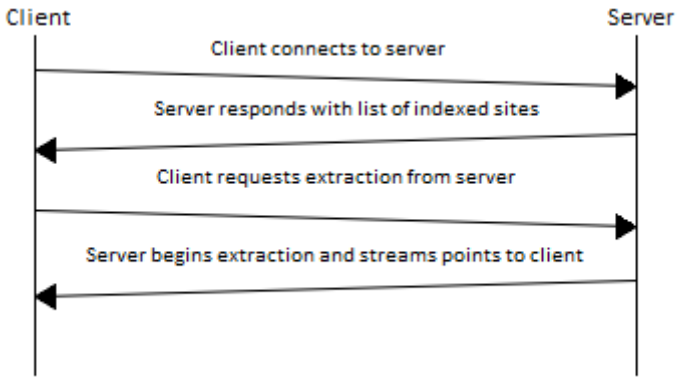


Figure 3.5: Client/Server Communication Protocol

The server is running all the time and is able to handle multiple client connections. Also, since the index files are small compared to the original point cloud files it is possible to store these fully in memory. Since disk accesses are so expensive, this results in a significant degree of latency hiding and the server is able to respond to requests faster.

## 3.14 Conclusion

In this chapter the design details of the indexing process, the extraction algorithm and the client/server protocol were discussed. Through the various stages of the algorithm an index has been created which will allow multiresolution region extractions. The design of the index was carefully considered to optimise performance. Although there are various ways of approaching this indexing problem, it will be seen in the evaluation section that the performance achieved is satisfactory.

# Chapter 4

# Implementation

## 4.1 Introduction

In this chapter the particular implementation details are provided. In particular, the development environment is described and the command line parameters for the various algorithms are listed.

## 4.2 Development Environment

A brief overview of the development environment and tools used is given in this section.

### Ubuntu Linux

Ubuntu Linux was chosen as the operating system for development. This is because of the large number of free open source tools available. It is also a convenient development environment and has good support for C++, the language that was used for the project.

### C++

Since efficiency is an important aspect of this project, C++ was the language of choice. C++ is powerful, efficient and has a large number of online resources to aid coding and development.

### PCL

The PCL (Point Cloud Library) is a library for C++ that provides several tools for working with point clouds [PCL, 2011]. Most important for this project is the point cloud viewer, as this easily enables one to view and interact with point clouds. This eliminates the need for some external viewer such as MeshLab. The PCL provides much functionality, such as the ability to construct kd-trees and octrees. However, all components of the PCL require all data to be in memory. The out-of-core component of the PCL is still under development.

### 64 bit architecture

Given the large nature of the datasets being worked with, a limitation of 4GB of addressable memory in the 32 bit architecture is far too restrictive and slows down several processes substantially. As a result, a 64 bit development environment was used throughout, with at least 8GB of memory. Although this is still far too little to store the entire point cloud in memory,

it speeds up the indexing process, and allows us to view larger point clouds without too large a performance hit.

## 4.3 Indexing

The indexing algorithm is performed as a pre-process to enable more efficient region extraction from previously unstructured point clouds. It also enables one to extract the regions at varying resolutions. It can be run using the following command with the specified parameters:

```
./index inputfile indexlocation resolution
```

The input file is the unstructured point cloud defined in the XYZ file format. The location is the folder where the index should be created, and the resolution defines the amount of subdivision of the point cloud.

## 4.4 Extraction

The extraction algorithm is implemented in *extraction.cpp* and can be run with the following parameters:

```
./extraction indexname x_start x_end y_start y_end z_start z_end view resolution
```

The index name specifies the indexed point cloud from which the extraction should be performed, for example, `mosqueindex` for the Mosque and `gedeindex` for the Gede Ruins. The next 6 floating point numbers specify the dimensions of the region from which the extraction should be performed. The next parameter, `'view'` is either $y$ or $n$ and specifies whether the user would like the cloud to be rendered once it has been extracted. Finally, the last floating point number, `'resolution'` specifies the resolution at which the point cloud is required. Note that $0 < resolution \leqslant 1.0$.

## 4.5 Client/Server

The server can be run with the following command and is implemented in `server.cpp`

```
./server
```

Similarly, the client is implemented in `client.cpp`. It takes a single argument, the name or IP address of the server to connect to. For example:

```
./client nala.cs.uct.ac.za
```

## 4.6 Conclusion

In this chapter only the particular implementation details of the system were discussed. The algorithms used and the details of the indexing process are in the design chapter.

# Chapter 5

# Evaluation and Results

## 5.1 Introduction

In this chapter a thorough evaluation of the system developed to facilitate multiresolution point cloud indexing is presented. Key tests are run to evaluate the performance of the core components of the system. In particular, the ability of the system to perform multiresolution extractions is evaluated. This is done by performing extractions on various datasets and varying either the size of the extraction of the resolution of the extraction. Various comparisons are also made to show the effects of different design choices, and the impact that these choices have on the overall efficiency of the system.

The test data used was acquired from the members of the Zamani project in the Department of Geomatics. The two main test point clouds are the Mosque (59,242,631 points) and the Gede Ruins (2,403,821,971). Indices are created for these point clouds as detailed in the design chapter. In the first few sections a fixed level of subdivision was used (`resolution=4.0`). Later on a variable level of subdivision is evaluated, as noted.

It is very important to recognise the effects of caching on the times recorded when evaluating any system that reads large amounts of data from disk. The first run of any test is always considerably slower than subsequent runs. This is because subsequent runs can take advantage of the fact that some of the data that was previously read from the hard disk, is now cached in main memory. Since reading from main memory is considerably faster than reading from the hard disk these subsequent runs tend to take about half the time of the first run (note that, in general, not all the read data can be cached). After this first run, the times differ very little and so it is often referred to as the steady state. Although in some cases it is fair to assume that some of the data to be read has been cached, this is generally not the case. The assumption is that the system should be able to handle varying requests for different regions at different resolutions. Thus, in general, it is assumed that no data has been cached as, in most cases, this would be an unfair assumption. If necessary, precautions are taken to clear cached data in order to ensure that test results are valid. In addition to this precaution, all tests are performed 4 times, and the results averaged. It is also noted here that the difference between running times was generally small, and subsequent tests differed by at most a few milliseconds.

## 5.2 Binary

Early in the design process, it was decided to represent the raw points in binary format. This is because of the significant boost in performance, due to the greatly decreased amount of time required to read the points. Figure 5.1 clearly illustrates that reading the points in binary

format takes around one tenth of the amount of time to read the points if they are represented in the ASCII format.

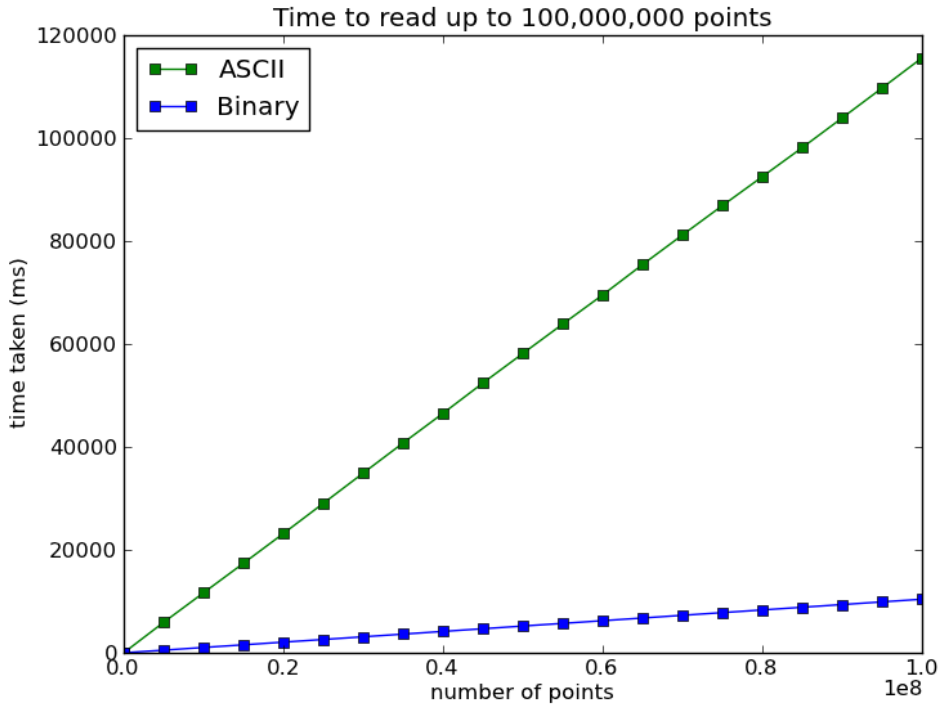


Figure 5.1: Reading the points in binary format is significantly faster

Figure 5.1 also illustrates that, as expected, the amount of time to read $n$ points scales linearly with the number of points.

## 5.3 Comparison with a Naïve Approach

As an initial feasibility demonstration, it is interesting to compare our region extraction algorithm with a more naïve approach that does not use any form of indexing. The naïve algorithm would use the following approach:

1. Specify a rectangular region for extraction.
2. Process the point cloud, point by point. For each point, one checks if it is in the specified region.
3. If the point is outside the region, it is discarded. If not, it is included in the output.

This algorithm was tested against the indexing algorithm on a point cloud of the Gede Mosque containing 59,242,631 points. The results are shown in Figure 5.2. It is clear that the performance of the index is far superior. Note that the time taken for the brute force approach is more or less constant. This is because, regardless of the dimensions of the region, the entire cloud must be processed each time since there is no subdivision. Additionally, the example considered is relatively small compared to the much larger clouds that contain billions, not millions, of points. In this case the brute force approach is simply infeasible as the input cloud is simply too big. It will be seen, however, that the indexing approach is still able to handle requests, even on these large clouds.

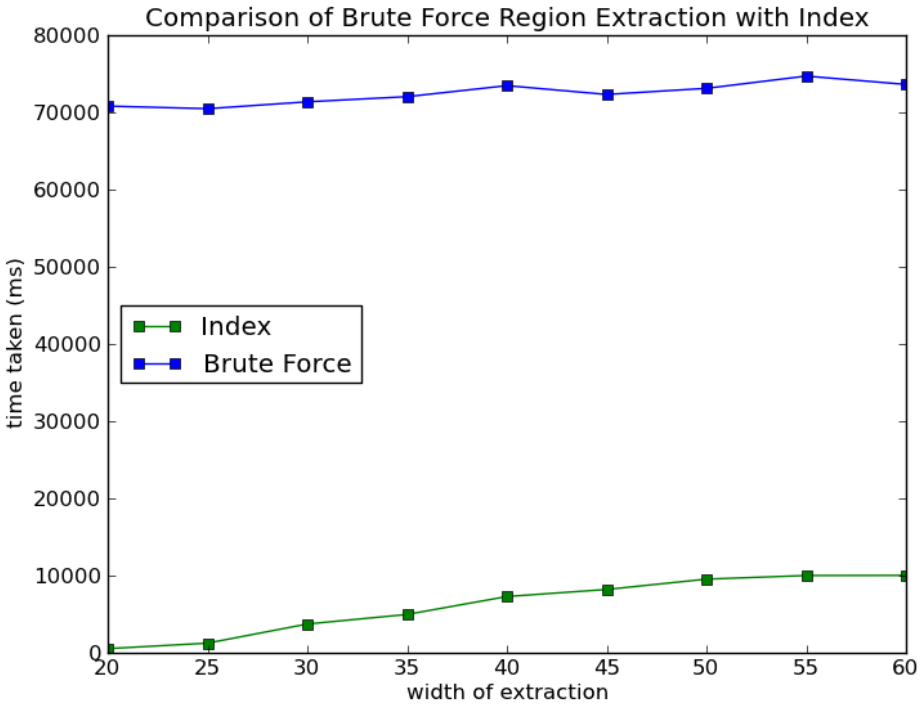


Figure 5.2: Brute force is considerably slower than the index

## 5.4 Varying Resolution

In this section the ability of the system to extract point clouds at varying resolutions is tested. Ideally, one would hope to see that the amount of time taken scales linearly with the resolution required. If not, it means that the index is adding a significant amount of undesirable overhead. The different sections are split depending on which dataset is being used. Also, there are three main contributors to the running time: the time taken to read the index file, the time taken to read those files that are partially contained in the region for extraction and the time taken to read those regions that are fully contained. Due to the required additional bounding box checks involved in reading the partial files, this can often be a large component in the time taken. All the points in the regions that are fully contained are read in and used so the only way to speed this up is to increase the rate at which one is able to read the points themselves. The following evaluation extractions illustrate the capabilities of the extraction algorithm, and one sees that the performance is satisfactory.

### The Mosque (subregion)

In this test the capability of the system to extract a region at varying resolutions from low resolution (10%) to high resolution (100%) is evaluated. The original point cloud is of the mosque and contains 59.2 million points and the subregion considered contains 19.2 million points. The results are shown in Figure 5.3.

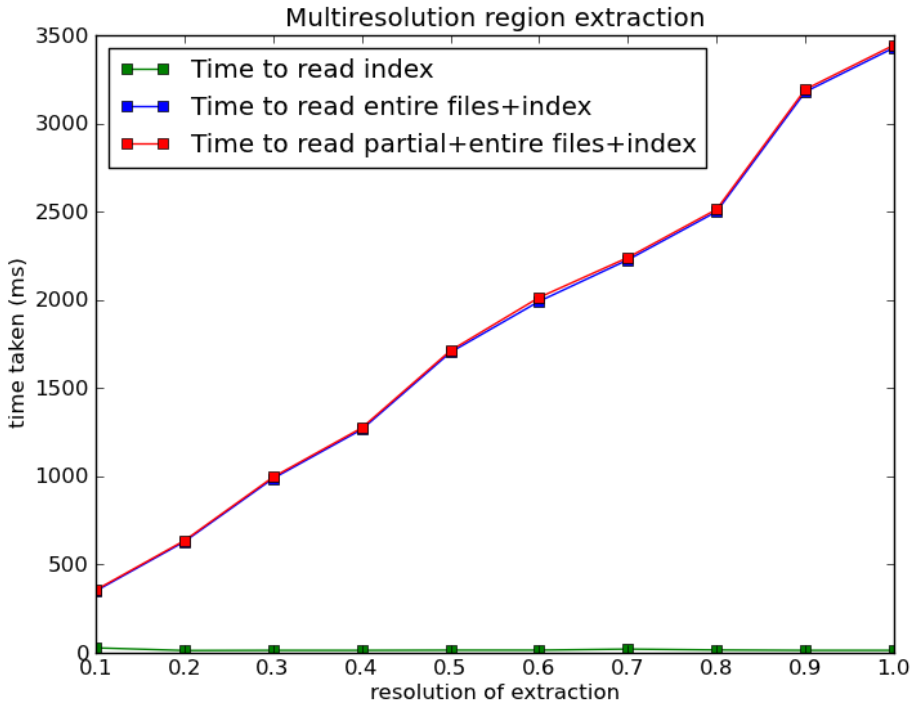


Figure 5.3: Multiresolution extraction of the Mosque

One can see that in this case the system is able to extract the low resolution region very quickly, as hoped. It also scales well and at full resolution the extraction takes just under 3.5 seconds. It is also worth noting that the time spent reading the partial files is, in this case, almost negligible. This indicates that the majority of the time is spent reading only points that are included in the specified region, and no time is wasted reading unnecessary data (as one would have to do in the naïve approach without any indexing).

### The Mosque (entire point cloud)

Here the entire mosque point cloud (59.2 million points) is extracted at varying resolutions. Since one is extracting the entire point cloud, and not a subregion, there is no contribution from files that are partially contained.

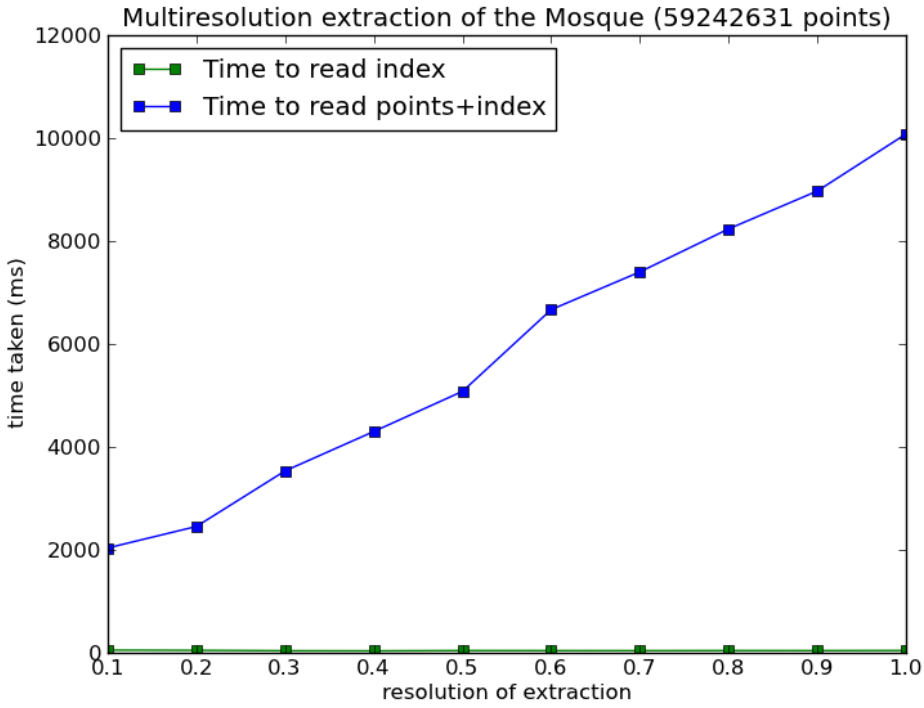


Figure 5.4: Multiresolution extraction of the Mosque

As can be seen, the time taken scales more or less linearly as hoped. Additionally, one can see that it is very quick to obtain a low resolution version of the point cloud (about 2 seconds for 10% resolution). This is ideal, as one can quickly get a low resolution version of the point cloud, and then scale up as necessary.

### Gede Ruins (pillar region)

The previous test cases have been fairly small. It is important to test the indexing system on much larger point clouds. The point cloud of the Gede Ruins scanned by the Zamani project contains over 2.4 billion points. Doing a brute force or naíve extraction in this case would be simple infeasible as it would take far too long (several hours). Using the indexing algorithm, an index was created for the Gede Ruins. Here it is evaluated by extracting a subregion at varying resolutions starting at a low resolution (10%) up to the full resolution (100%). The entire point cloud contains 2,403,821,971 points and the subregion for extraction contains 137,839,454 points.

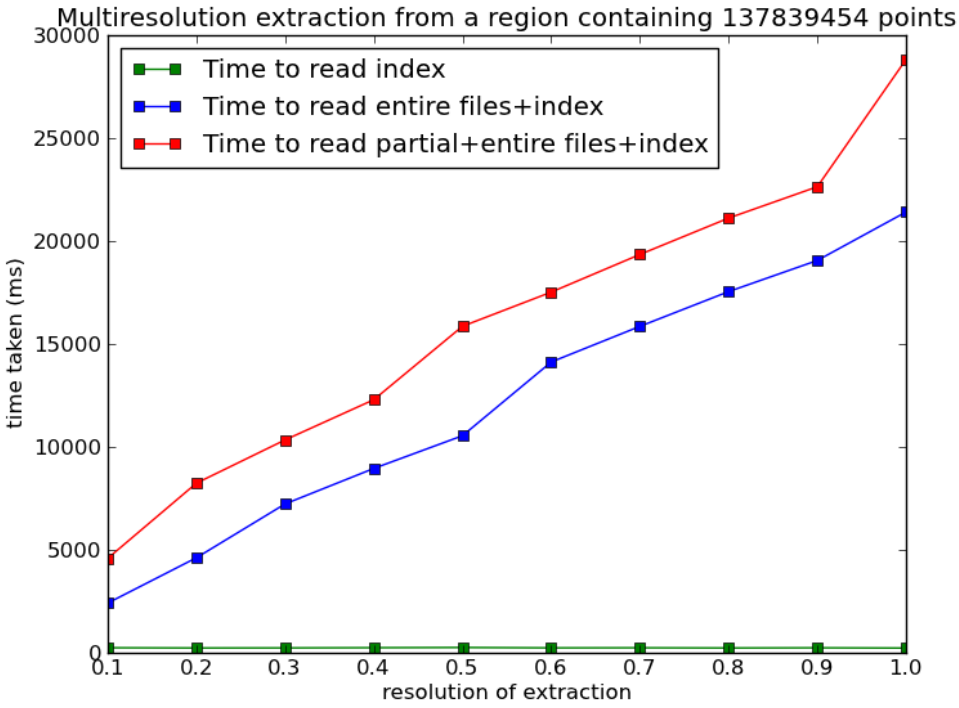


Figure 5.5: Multiresolution extraction of a region of the Gede Ruins

As Figure 5.5 indicates, even though the original point cloud is very large, containing billions of points, one is able to quickly and efficiently extract a region at varying resolutions. In this case considerably more time is spent reading the partial files than in previous examples. This indicates that some time is wasted reading in unnecessary data. It is for this reason that the level of subdivision is an important consideration when indexing the point cloud. The effects of varying levels of subdivision is discussed later.

### Gede Ruins (entire point cloud)

Most often only a subregion of the point cloud is required. Sometimes, however, one requires the entire point cloud. In this test, the full scan of the Gede Ruins is used (containing over 2.4 billion points). The capability of the system to extract the entire region at varying resolutions is evaluated. The resolution is varied from 1% to 10%. Due to the large number of points in the full cloud, extracting at full resolution is a lengthy process. The results are shown in Figure 5.6.

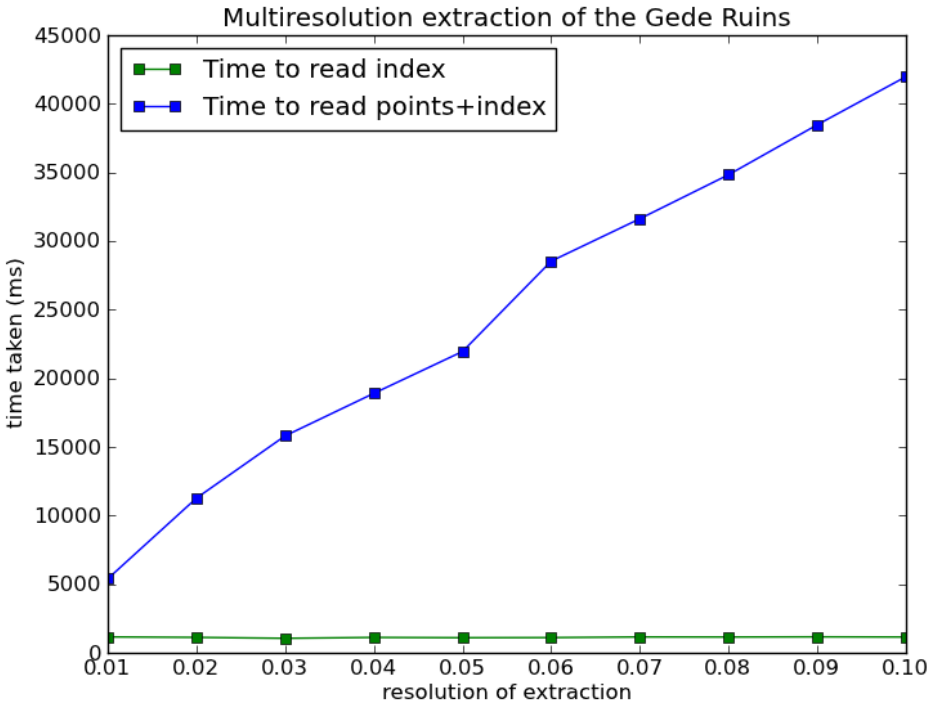


Figure 5.6: Multiresolution extraction of the entire Gede Ruins site

Even with such a large point cloud, it is apparent that, as hoped, the time spent reading the index is minimal compared to the time spent reading the required points. It is reasonably quick to obtain the low resolution version (about 5 seconds) and, as with the previous examples, it scales well as the resolution of the extraction is increased.

## 5.5 Region Extraction

Next the region extraction capabilities of the system are evaluated. In the previous section various regions were extracted at varying resolutions. Here, the resolution is kept constant while the size of the region is varied. Whereas the multiresolution extraction varied more or less linearly, one will see that when extracting regions, there are many additional factors that need to be considered. The two most important factors are:

1. The varying density of the point cloud
2. The transition between different regions in the subdivision scheme

If the density of the point cloud increases significantly, then naturally it will take longer to read that region. Also, in the indexing structure, if a region has many partially contained files it will take longer than average. However, as the results show, provided a sufficient level of subdivision is obtained, the results are good and scale reasonably well.

### Mosque (varying region width)

First, a region extraction of the Mosque is evaluated. The height and length of the region are kept constant (containing the entire point cloud) and only the width of the region is varied. For each width, the time taken to perform that extraction is recorded. In this case, all extractions are at full resolution. The results are shown in Figure 5.7.

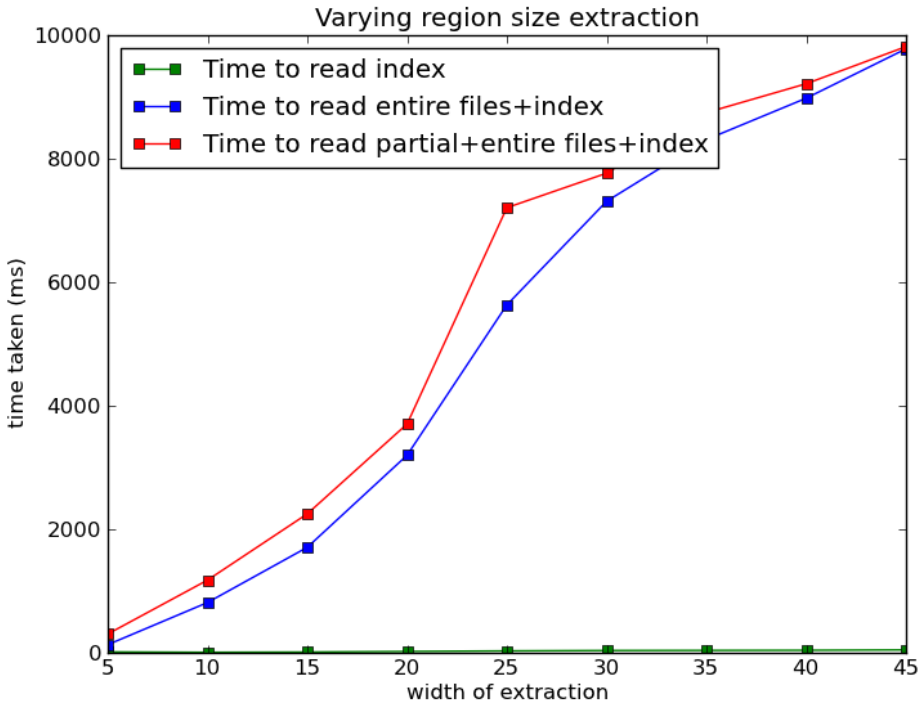


Figure 5.7: Varying region size extraction of the Mosque

The small regions are extracted very quickly, as expected. As the region size increases, the time increases proportionally. In the middle, however, there is a significant increase in time. This is due to the fact that at this point the point cloud is most dense, so increasing the region width results in significantly more points that need to be read in and processed.

### Gede (varying region width)

As in the previous section, it is also important to evaluate with a much larger point cloud (billions of points, not millions). As before, the index for the Gede Ruins is used. A region with a high density of points was selected for testing. The height and length of the region was kept constant and the width was varied and used as an independent variable. The results are shown in Figure 5.8.

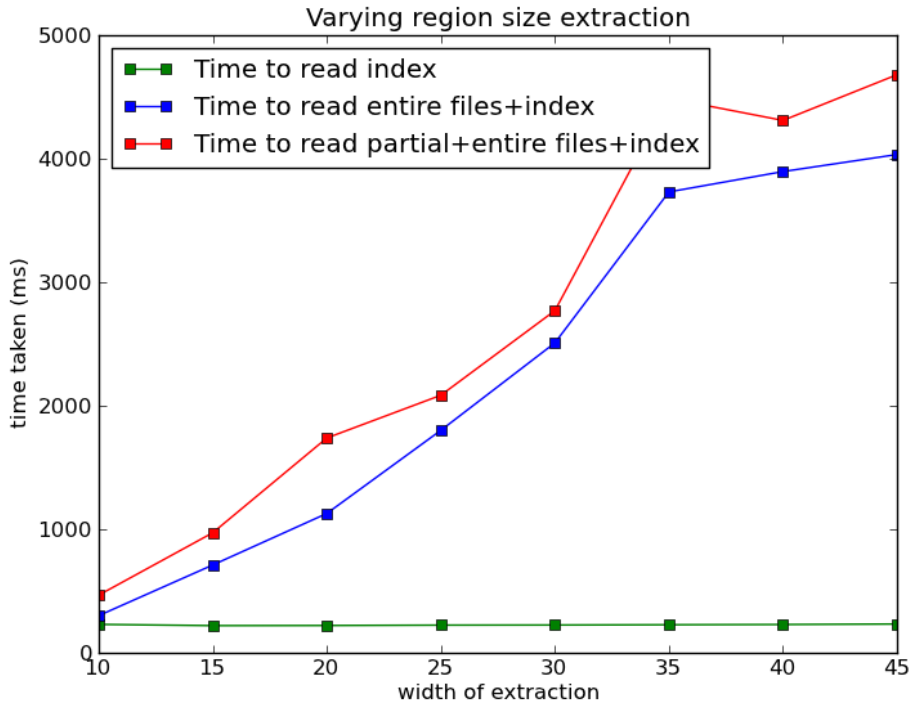


Figure 5.8: Varying region size extraction of the Gede Ruins

As can be seen, as one increases the size of the region the time taken scales reasonable well. The reason for the dip at the end is due to the aforementioned factors: the varying density of the point cloud and also the transition between different regions in the subdivision. In particular, when the width of the region was 40, it was slightly faster to extract this region than when the width was 35, even though it was larger. This is because there were fewer partial files compared with a width of 35 and so there was less overhead involved in performing the bounding box check each time.

## 5.6 Increasing the Level of Subdivision

An important parameter in the system is the level of subdivision. If one has no subdivision (and no reordering of points from the multiresolution scheme) then essentially the algorithms reduce to brute force. However, if the level of subdivision is too fine, the index file will be extremely large indeed and each node/file will contain too few points to justify the overhead of reading the index, opening the file and closing this file. In this section, the various effects of increasing the level of subdivision are investigated.

### Gede Ruins

In the previous sections, the indices used had a fixed level of subdivision (`resolution=4.0`). However, there is no specific reason for the level of the subdivision to be fixed. In regions where the point cloud is very dense, it is possible to further subdivide in order to get a more efficient index. In particular, large files that contain millions of points are avoided. This improves performance because, if one of these large files is partially contained in a region for extraction, it can add significant overhead. Thus the indexing system was refined to allow a variable level of subdivision. Here, the level of subdivision is increased up to the point where no file contains more that 500,000 points. This number was a guideline given in related literature [Gobbetti

and Marton, 2004]. The following figures illustrate the improved performance when the level of subdivision is increased.

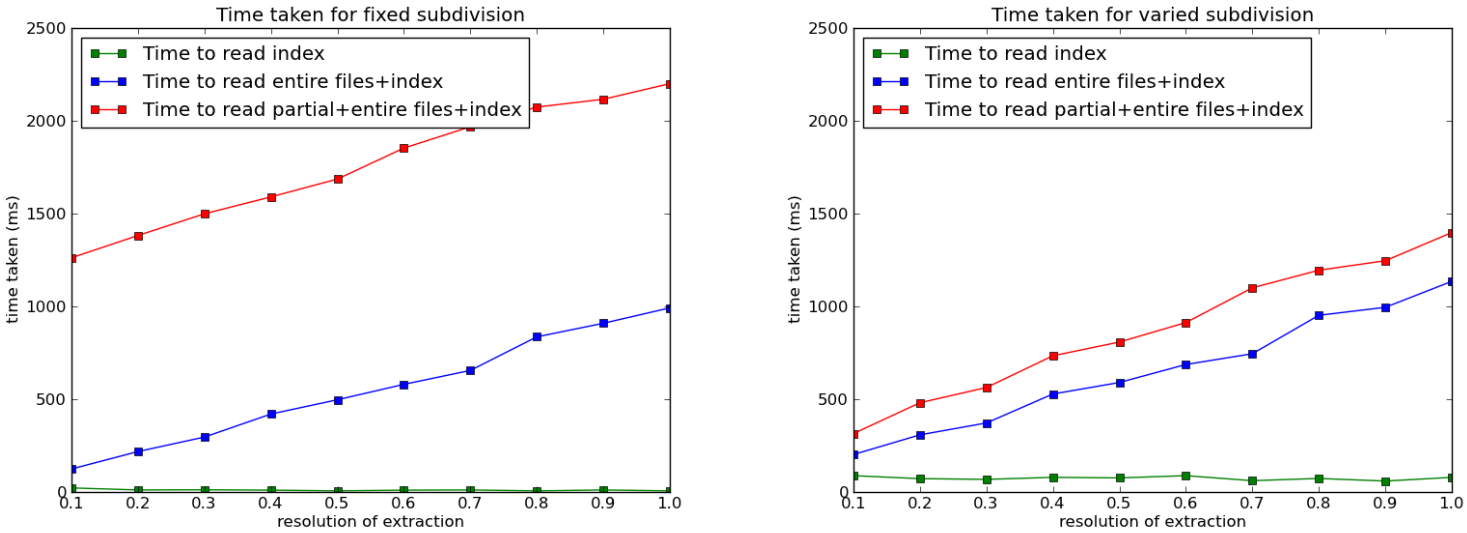


Figure 5.9: The figure on the left illustrates the time taken with a fixed level of subdivision. The figure on the right shows the times when a variable level of subdivision is used.

From Figure 5.9 it is clear that, although in the second case reading the index file takes longer, the overall performance is improved. In particular, for this region the time taken to read the partial file in the first case was very long indeed. By dividing this large file into significantly smaller chunks, it was possible to greatly reduce the time spent reading unnecessary data. Thus, by using a variable level of subdivision it was possible to approximately halve the time taken to perform this extraction (at all resolutions). Figure 5.10 directly compares the overall time taken in each case.

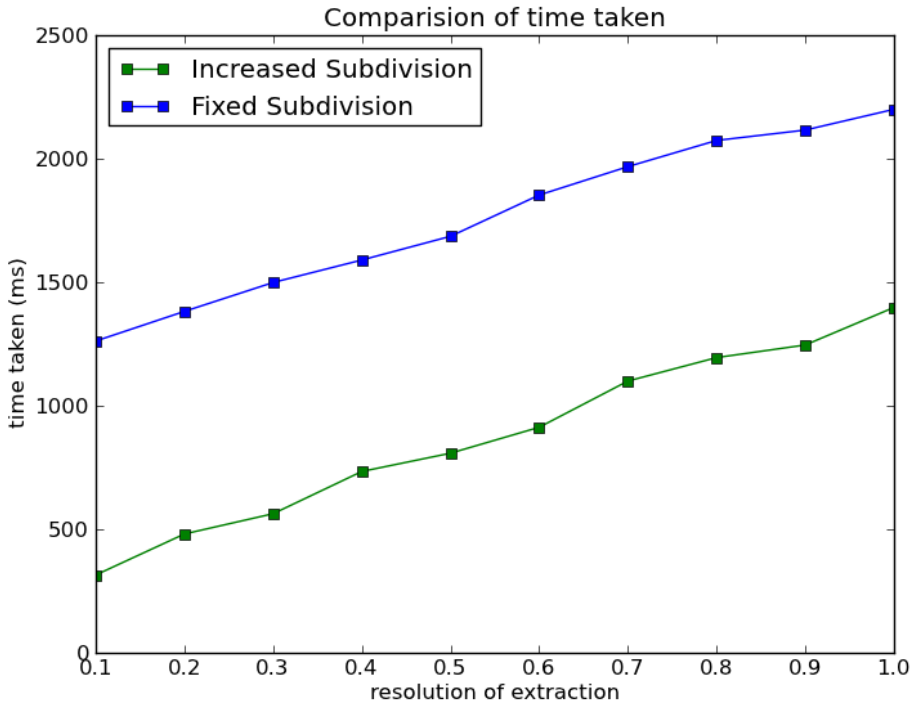


Figure 5.10: Comparison of Differing Levels of Subdivision

### Optimal Level of Subdivision

As has been seen, increasing the level of subdivision often leads to better results and more efficient extractions. Determining the optimal level of subdivision, however, is a difficult problem. In related literature, it was suggested that performance only degrades significantly when the size of each subdivision exceeds 500,000 points [Gobbetti and Marton, 2004]. This seemed to be a reasonable guideline in this case as well since reading a binary file containing 500,000 points takes less than a second ($\sim$ 0.59 seconds). Without the increased subdivision, some files contained as many as 10,000,000 points. Processing these large files can add additional overhead of up to several minutes.

## 5.7 Streaming

The streaming infrastructure developed for this project allows the points extracted from the index to be sent from a server to a client machine over a network connection. Most of the speedup in the streaming comes as a result of the index. The points are streamed in binary format, the benefits of which have already been discussed. Figure 5.11 illustrates the additional overhead introduced when streaming the points.

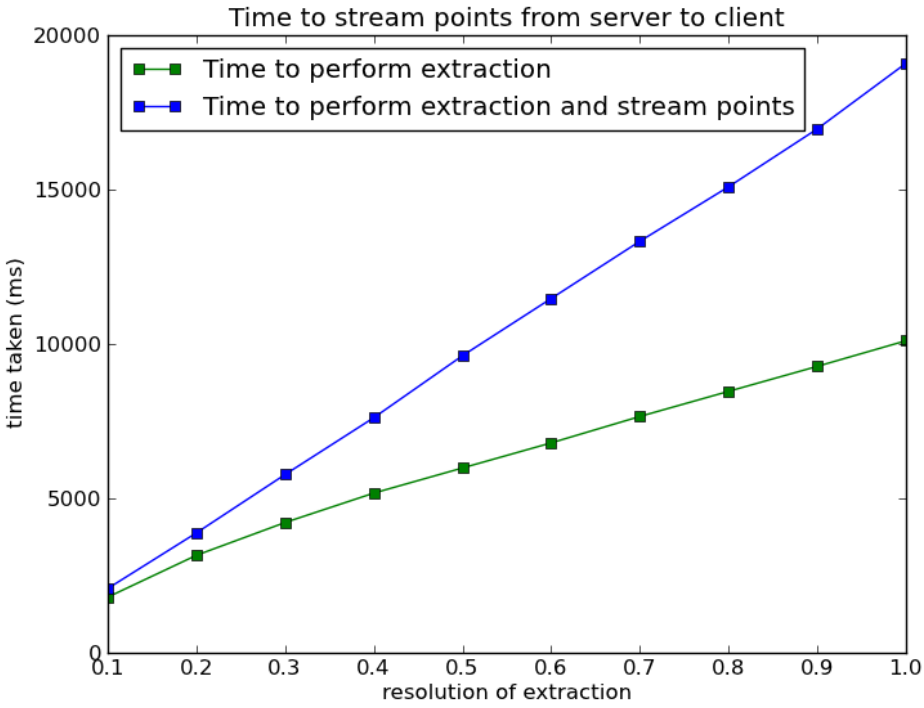


Figure 5.11: Comparison of Differing Levels of Subdivision

As Figure 5.11 illustrates the majority of time is spent performing the extraction. However, especially when the number of points is large, a significant amount of time is spent transferring the data to the client (48% when `resolution=1.0`). A natural limitation is the speed of the network connection which in this case was 100 Mb/s. Using a gigabit connection would mean that the streaming of the points to the client machine would add less overhead. Additionally one could investigate ways of reducing the amount of data that needs to be transferred by using techniques such as differential encoding or compression. However, it is important to note that the system was very quick to obtain a low resolution representation of the point cloud and stream it to the client. Since this can be done quickly, one can view this point cloud, or feed it to some other process, and then stream additional points on demand.

## 5.8 Conclusion

In this evaluation section the ability of the system to perform extractions of varying regions at varying resolutions was evaluated. One of the key findings was that when increasing the resolution of an extraction, the time taken scaled linearly as hoped. Also, in general, the amount of time taken to perform the extractions was reasonable (less than 20 seconds in most cases), even when the underlying point cloud was very big and contained billions of points. Finally, it was found that increasing the level of subdivision tended to yield better results for most region extractions. However, the optimal level of subdivision needs to be determined empirically.

# Chapter 6

# Conclusion and Future Work

This project developed a system for dealing with the indexing of large amounts of data, and in particular large 3D point clouds. Algorithms were developed allowing both for efficient region extraction and a scalable approach to the extraction of the regions at varying resolutions. The capabilities of the system were also evaluated. The system performed well with the test datasets and was able to achieve reasonably fast extraction times. It was also demonstrated that increasing the resolution scaled linearly, as desired. Finally, the client/server architecture allowed for streaming of point clouds from a central server to client machines.

The original research question posed was, "Is it feasible to support efficient multiresolution extractions from large point clouds containing billions of points?" In the evaluation section, many tests were performed on various datasets of different sizes. Even when the original point cloud was very large, the amount of time taken was reasonable. Although one can always endeavour to improve the efficiency, it was determined that the times achieved were efficient enough for the purposes of this project.

There are certainly many more avenues to explore with a project such as this. There are almost always more ways to be explored to improve the efficiency of the system. In future work it would be interesting to investigate the effects of differential encoding of the points. This would reduce the size of the files containing the points, and consequently the time taken to read the data. One could also investigate the use of other data structures such as KD-Trees and compare the performance with this system.

In this project one aspect of the big data problem was examined, namely, dealing with large point clouds. It is also possible to apply some of the results in a more general context. In particular, it was seen that the big data problem can often be tackled by subdividing the dataset into more manageable chunks and processing them individually. The final step is to recombine the individual answers to obtain the solution to the original problem. If one is able to do this, it can greatly simplify many of the difficulties involved in dealing with big data.

# Bibliography


[Akbarzadeh et al., 2006] Akbarzadeh, A., m. Frahm, J., Mordohai, P., Engels, C., Gallup, D., Merrell, P., Phelps, M., Sinha, S., Talton, B., Wang, L., Yang, Q., Stewenius, H., Yang, R., Welch, G., Towles, H., Nistér, D., and Pollefeys, M. (2006). Towards urban 3d reconstruction from video. In *in 3DPVT*, pages 1–8.

[Beckmann et al., 1990] Beckmann, N., Kriegel, H.-P., Schneider, R., and Seeger, B. (1990). The r*-tree: an efficient and robust access method for points and rectangles. *SIGMOD Rec.*, 19(2):322–331.

[Cignoni et al., 2008] Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F., and Scopigno, R. (2008). Adaptive tetrapuzzles: efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. In *ACM SIGGRAPH ASIA 2008 courses*, SIGGRAPH Asia '08, pages 33:1–33:8, New York, NY, USA. ACM.

[Faloutsos and Roseman, 1989] Faloutsos, C. and Roseman, S. (1989). Fractals for secondary key retrieval. In *Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, PODS '89, pages 247–252, New York, NY, USA. ACM.

[Gobbetti and Marton, 2004] Gobbetti, E. and Marton, F. (2004). Layered point clouds: a simple and efficient multiresolution structure for distributing and rendering gigantic point-sampled models. *Computers & Graphics*, 28(6):815 – 826.

[Greene et al., 1993] Greene, N., Kass, M., and Miller, G. (1993). Hierarchical z-buffer visibility. In *In Computer Graphics (SIGGRAPH '93 Proceedings*, pages 231–240.

[Guttman, 1984] Guttman, A. (1984). R-trees: A dynamic index structure for spatial searching. In *INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA*, pages 47–57. ACM.

[Kofler et al., 2000] Kofler, M., Gervautz, M., and Gruber, M. (2000). R-trees for organizing and visualizing 3d gis databases. *Journal of Visualization and Computer Animation*, 11(3):129–143.

[Kumar et al., 1996] Kumar, S., Manocha, D., Garrett, W., and Lin, M. (1996). Hierarchical back-face computation. In *Proceedings of the eurographics workshop on Rendering techniques '96*, pages 235–253., London, UK, UK. Springer-Verlag.

[PCL, 2011] PCL (2011). Point cloud library. Website. `http://pointclouds.org/`.

[Pfister et al., 2000] Pfister, H., Zwicker, M., van Baar, J., and Gross, M. (2000). Surfels: surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 335–342, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

[Rusinkiewicz and Levoy, 2000] Rusinkiewicz, S. and Levoy, M. (2000). QSplat: A multiresolution point rendering system for large meshes. In *Proceedings of ACM SIGGRAPH 2000*, pages 343–352.

[Ruther et al., 2012] Ruther, H., Held, C., Bhurtha, R., Schroeder, R., and Wessels, S. (2012). From point cloud to textured model, the zamani laser scanning pipeline in heritage documentation. *South African Journal of Geomatics*, 1(1):44 – 59.

[Sellis et al., 1987] Sellis, T. K., Roussopoulos, N., and Faloutsos, C. (1987). The r+-tree: A dynamic index for multi-dimensional objects. In *Proceedings of the 13th International Conference on Very Large Data Bases*, VLDB '87, pages 507–518, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Soergel, 1985] Soergel, D. (1985). *Organizing information : principles of data base and retrieval systems / Dagobert Soergel*. Academic Press, Orlando, Fla. :.

[Wand et al., 2007] Wand, M., Berner, A., Bokeloh, M., Fleck, A., Hoffmann, M., Jenke, P., Maier, B., Staneker, D., and Schilling, A. (2007). Interactive editing of large point clouds. In Chen, B., Zwicker, M., Botsch, M., and Pajarola, R., editors, *Symposium on Point-Based Graphics 2007 : Eurographics / IEEE VGTC Symposium Proceedings*, pages 37–46, Prague, Czech Republik. Eurographics Association.

[Wang and Shan, 2005] Wang, J. and Shan, J. (2005). Space-filling curve based point clouds index. *Geocomputation*.

[Watters, 2010] Watters, A. (2010). The age of exabytes: Tools and approaches for managing big data. Slideshare.

[Zhu et al., 2007] Zhu, Q., Gong, J., and Zhang, Y. (2007). An efficient 3d r-tree spatial index method for virtual geographic environments. *ISPRS Journal of Photogrammetry and Remote Sensing*, 62(3):217 – 224.

[Zlatanova, 2000] Zlatanova, S. (2000). *3D GIS for Urban Development*. PhD thesis, International Institute for Geo-Information Science and Earth Observation, Netherlands.