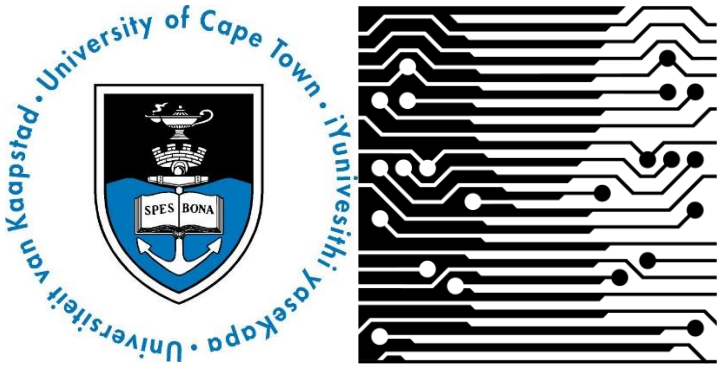
Honours Project Report



Adapting an existing mesh simplification algorithm into a hierarchical level of detail mesh processor

BRNDAN022

Daniel Burnham-King

Supervised by Dr Patrick Marais

	Category	Min	Max	Chosen	
	Requirement Analysis and Design	0	20	10	
	Theoretical Analysis	0	25	0	
	Experiment Design and Execution	0	15	15	
	System Development and Implementation	0	20	10	
	Results, Findings and Conclusion	10	15	15	
	Aim Formulation and Background Work	10	10	10	
	Quality of Report Writing and Presentation	10		10	
	Adherence to Project Proposal and Quality of Deliverables	10		10	
	Overall General Project Evaluation	0	10	0	
Total marks		80			

Acknowledgements

This project would not have been possible without the continued support and guidance of everyone who was willing to put up with me over the last couple of months.

In particular I would like to express my gratitude towards our project supervisor, Patrick Marias, who provided us with guidance throughout the project.

A special thanks to the awesome people in my group, Benjamin Meier and Justin Cossuti, through whom countless hours turned into a fantastic piece of software.

To the adventurers and researchers in the Zamani group, who do amazing work recording the treasures of Africa. For supplying us with meshes and including us in such an incredible initiative.

To my family and friends, who kept the encouragement flowing, you guys are awesome!

This project would not have been possible without funding from the National Research foundation.

Abstract

State-of-the-art laser scanning allows the Zamani group to generate highly detailed models that are used to document archaeological sites. These models contain anywhere from 20 million to as many as 2 billion, exceeding the memory capacity most computers would need to render them. Further advances in scanning technology will continue to cause the size of scanned models to increase at a rate hardware development cannot keep up with.

In an effort to subdue current and future hardware limitations we have implemented a view dependant hierarchical level of detail scheme. Models are pre-processed and various levels of detail are loaded in on demand, drastically increasing the performance of these models.

This project focuses on implementing the simplifier component of the mesh viewing suite, called SimplFy. It was developed by experimenting with and extending an existing simplification program from the VCG library.

After modifying the boundary preservation behaviour of the library, while evaluating the system it was found that only minor changes were required to meet requirements.

CONTENTS

TABLE OF FIGURES

1 INTRODUCTION

The Zamani Project [1], run by the Geomatics Department at the University of Cape Town, aims to digitally capture accurate three dimensional architectural models of African heritage sites for storage in the African Cultural Heritage and Landscape Database. The project has scanned 32 different sites in 12 countries. The models are used for the preservation, education, research, and restoration of the sites. The project uses phase-based laser scanners to record range data from 300 to 800 positions around the site. Buildings are typically scanned with a 1 - 2 cm point spacing while terrain is captured at a 10 - 50 cm point spacing. The range information is used to create a dense point cloud that represents the surface of the building or terrain being scanned.

The point cloud is “cleaned” by hand to remove any unwanted objects or points, such as trees, before undergoing surface reconstruction to determine a continuous surface. Surface reconstruction is the automatic conversion of a point cloud to a mesh. For the Zamani project, such a mesh can contain billions of faces and vertices. The researchers at Zamani use Meshlab, an open source 3D triangular mesh processing system to navigate and interact with the resulting models. On our test machine a mesh with 3 million faces is rendered below 5 frames per second, far below an interactive frame rate[1].

Meshes can be algorithmically simplified (decimated) to reduce the amount of geometry and thus increase rendering speed. Unfortunately, this can compromise details such as crevices, engravings and other surface texture. This is detail Zamani do not want to lose. There is thus a need for a far more effective method for displaying highly detailed meshes while maintaining interactive frame rates.

To achieve this, we endeavored to create a ‘massive mesh viewing suite‘ based on a hierarchical level of detail model. This suite is composed of a mesh preprocessor and a renderer.

1.1 PROJECT DESCRIPTION

This report documents the development and evaluation of SimplFy, the mesh simplification component of our massive mesh viewing suite. This component is used in conjunction with a mesh divider to create a hierarchical level of detail representation of the mesh. After the mesh is pre-processed, the renderer is then able to load suitably detailed pieces of the mesh into memory on demand. By displaying lower detail when the mesh is further away from the camera, we are able to attain the interactive display of models while still preserving the full detail of mesh areas near to the camera.

[1] An interactive frame rate is the display update rate required for user to effectively interact with a model. Slow frame rates can produce stuttering when objects on screen are animating. For the purpose of our project, we have set our target frame rate to 30.

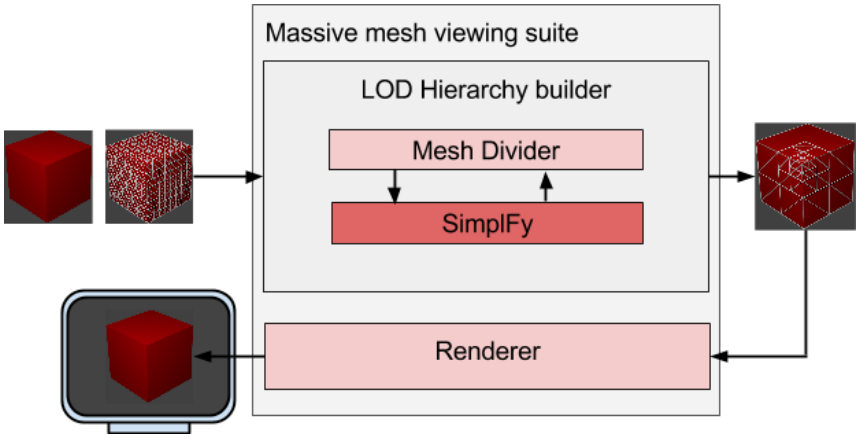


Figure 1 Group project architecture overview

Due to honours project requirements, the mesh viewing suite was separated into 3 sub-projects (Figure 1). The mesh divider spatially divides the original mesh into a hierarchical data structure. The leaves of the tree (nodes containing mesh segments) are then stitched (merged) together and simplified recursively. Generating a hierarchical level of detail (HLOD) representation of the mesh.

The renderer then loads mesh segments containing different levels of detail from the hierarchy on demand. Low detail segments are loaded for parts of the mesh that are distant from the camera and higher detail for nearby ones.

The mesh divider uses SimplFy (The scope of this report) to simplify the mesh segments. SimplFy is an extension of an existing implementation of a simplification tool found in the Visualization and Computer Graphics Library (VCG). This report discusses why the VCG implementation, called tridecimator is insufficient and focuses on evaluating the changes I made within the context of the whole system.

1.2 RESEARCH QUESTION

3D triangle mesh simplification is currently a well-defined and very mature area of research with many complex algorithms. Due to the time and scope constraint of honours projects, developing and implementing a novel simplification scheme is not feasible. With these prior considerations in mind, the principal research question that this project sought to answer is:

"Can an existing mesh simplification implementation be adapted to work with an out-of core hierarchical level of detail mesh scheme?."

1.3 System overview

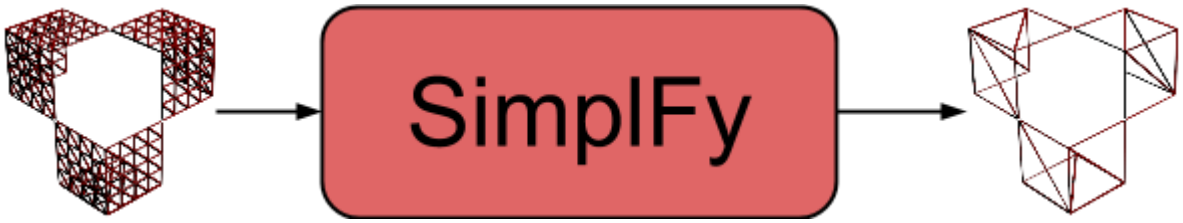


Figure 2 a high level overview of SimplFy

SimplFy (Figure 2) is a mesh simplification tool that extends the Tridecimator simplifier found in the virtualization and computer graphics library (VCG). Adapting it to be used in our mesh viewing suite. This required experimenting with different configurations. SimplFy also features a refined heuristic for specifying border triangles, preventing seams from appearing in simplified mesh segments due to the splitting, simplifying and stitching process. It also enables support for meshes which contain per vertex colour.

1.4 Legal and ethical considerations

The provided models used during development and for testing the performance of the implemented algorithms remain the property of the UCT Geomatics department and the Zamani project. While they will be used to obtain benchmark results, the files themselves will not be included in the original project hand-in, to prevent accidental public distribution. Access to the supplied models was provided via a password protected FTP account from UCT's Geomatics Department. The list of authorized parties was limited to the client, ourselves and the project supervisor. SimplFy is licensed under the GNU public license, since code from VCG was used in its implementation.

1.5 Report structure

The background research for this project is presented in chapter 2. Following this, the design chapter describes the experimental design process. The implementation and Evaluation strategies of the project are detailed in chapters 4 and 5. Finally, chapters 6 and 7 present the results and wrap up the report. Additional test results can be found in the appendix.

2 BACKGROUND

This chapter presents the domain specific background research that influenced the development of this group project. We will be focused on the techniques and tools used to simplify meshes.

Before discussing mesh decimation techniques, we present a brief overview of meshes, rendering and spatial partitioning. We then present techniques for displaying and decimating meshes

2.1 MESHES AND RENDERING

3D meshes are used to represent objects in 3D computer graphics. They are composed of a collection of vertices, edges and faces. Vertices are points in 3D space. Lines that connect vertices are called edges and a face is made up of a collection of polygonal structures enclosed by a set of edges (Figure 3). In modern rendering, triangles are typically used as the structure for each face. For the purpose of this report, “models” and “meshes” will be used interchangeably and mesh geometry will refer to a meshes triangular structure.

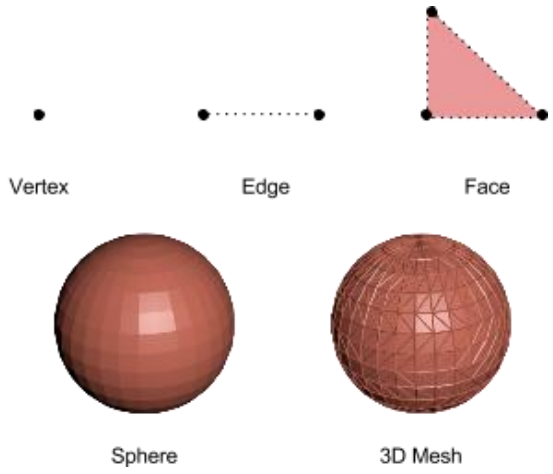


Figure 3Components of a typical 3D mesh

The process of displaying a mesh on a screen is called rendering. The mesh is rendered in an environment (or scene) by transferring mesh data is from system memory to the memory of a graphics processing unit (GPU). GPU’s are responsible for translating the mesh from a 3D space to a 2D space, displaying the mesh on screen (also known as rasterization).

2.2 SPATIAL PARTITIONING

Spatial data structures [2] [3] exploit spatial coherence and are based on the principle of recursive decomposition. K-d trees and oct-trees can be used to spatially divide meshes into chunks that can be handled can be used to hierarchically group parts a virtual environment or model by spatially dividing it.

2.3 LEVEL OF DETAIL FRAMEWORKS

Level of Detail (LOD) [4] describes a variety of rendering frameworks that use lower detailed representations for small, distant or unimportant portions in a scene. The lower the detail of the object, the faster it will render.

Figure 4 Level of detail in action, sourced from the Stanford 3D scanning repository

Traditionally, LOD's are implemented discretely (Discrete LOD), by swapping an object with a lower detailed (simplified) version. Swapping usually occurs when the object is too far from the camera to discern finer detail (Figure 4), meaning little or no visual quality is lost. The lower detailed representation of objects are typically processed offline (before rendering), which means that there is practically no performance overhead when swapping between levels of detail. "Popping", where a model noticeably loses or gains detail as it is swapped for another representation is common artefact of discrete LOD's. This can be softened by using many different levels of detail. The different levels of detail of Figure 4 are presented in Figure 5.

Instead of creating individual level of detail representations of an object, the simplification process on an object can create a data structure encoding some or all lost detail. This detail can then be loaded in at run time (Continuous LOD) providing finer control of the objects' detail.

View-dependant LOD extends continuous LOD by using view-dependant criteria to dynamically load the most appropriate detail for the current view. This allows an object to span multiple levels of simplification at any time. The parts of an object nearest to the camera could then contain higher detail than parts that are distant.

Hierarchical level of detail (HLOD) [5] data structures generalize discrete, continuous and view dependant LOD methods into a hierarchical aggregation of mesh segments (nodes). The mesh is then spatially divided into a recursively defined data structure such as an oct-tree. Each parent node can then be simplified so that its children contain more detail. The tree can then be traversed at run time to load higher detail segments of a mesh in as it approaches the camera. This effectively allows HLOD to behave as a discrete LOD scheme for different segments of the same mesh.

2.4 Out-of-core mesh simplification and rendering of massive meshes

Memory in modern computers can be categorised as main or external memory. Main memory includes memory directly accessible by the processor and in modern computers is typically around 4 - 8 Gigabytes (GB). External memory includes storage devices like hard disk drives which typically able to store from 120 to 1000 gigabytes. Algorithms designed to process data within the confines of main memory are said to be in-core. Out-of-core algorithms are generally designed to operate on datasets that are too large to be loaded into main memory, instead relying on the slower external memory.

QSplat [6] has been able to demonstrate real time progressive rendering of large models. Through the use of bounding sphere hierarchies, QSplat was able to render a complex model (containing more than 127 million points). Decreasing render time greatly by only rendering 8 million points when the model was static (less when it was moving). The detail to which each bounding volume is rendered is dependent on the distance of the volume to the screen. Volumes that are far away are rendered with simplified points

The progressive mesh [7] representation provides a scheme for "keeping track" of the iterations used to simplify a mesh with mesh decimation, allowing them to be undone iteratively. Providing a level-of-detail model as well as selective refinement, allowing a specific region of a simplified mesh to be de-simplified (refined). The mesh can be constructed using an edge collapse operation together with a minimizing energy function to insure that surface geometry is preserved.

Lindstrom [8] and Cignoni et el. [9] proposed systems that used out-of-core simplification as well as an oct-tree based hierarchical view dependent level of detail system. The large amount of data that 3D scanned models can contain renders most simplification algorithms inadequate, since they require the mesh to be loaded into main memory. However, scanned meshes may greatly exceed the memory available in modern computers. Lindstrom's method performs all computations on disk, allowing for large models to be processed with minimal memory (tested on a model containing 372,767,445 vertices).

2.5 Mesh decimation

Mesh simplification, also known as mesh decimation involves approximating an initial mesh with less geometry (Figure 5), in our case this involves reducing the number of triangles that make up a surface. This reduces the memory and processing overhead of rendering the model, reducing the time needed to update the display (higher frame rate). Simplified meshes can trade fine detail for better performance.

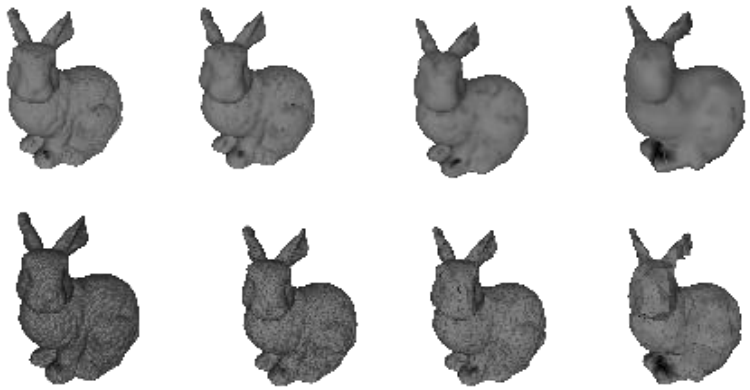


Figure 5 Different levels of detail of a 3D mesh. Detail decreasing left to right, mesh view at the bottom.

There are many existing mesh simplification methods. They are devised to minimize the number of vertices or faces (or both) and may preserve topology, vertex locations and feature edges. In terms of maintaining a level of accuracy or maintaining features from the original models, different methods may also make use of global or local error metrics [10]. Meshes can be simplified through controlled vertex, edge or face decimation methods. These methods are typically used to decimate edges, faces or vertices iteratively according to some local quality maximization criteria using edge, face or vertex collapses. Edge collapses can either collapse to vertices lying on the same edge into a single new vertex, or collapse one vertex into the other (half edge collapse). Vertex clustering on the other hand, spatially groups or 'clusters' vertices together. The method is very efficient, but small shape details may be lost. (Figure 6).

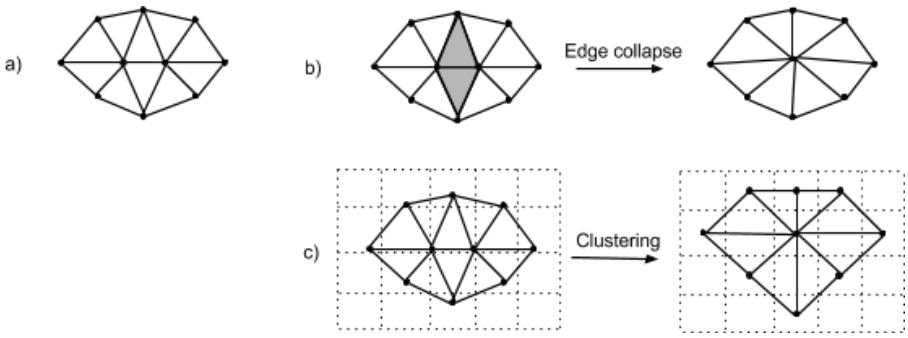


Figure 6 A comparison of edge collapse and vertex clustering techniques. a) is the original mesh.

Surfaces can be simplified by using collapsing vertex pairs that don't necessarily lie on the same edge. Quadric Error metric surface simplification [11] performs iterative vertex contractions (collapses). These contractions are guided by pre calculated 4x4 matrices called quadrics. The algorithm initially computes the quadric (Q) for each vertex. An optimal vertex placement is

calculated for all valid vertex pairs (those that lie on the same edge or are within the specified distance/threshold of each other). The Vertex pairs are then added to a heap keyed prioritising low cost vertex pairs. These vertex pairs are then iteratively removed, updating the corresponding costs of pairs involving the decimated vertex. Quadric error edge collapse was evaluated against various other methods [10]. An implementation of the algorithm is free to use in Meshlab.

2.6 MESH PROCESSING TOOLS AND LIBRARIES

The Computational Geometry Algorithms Library (CGAL) [12] provides easy access to efficient and reliable geometric algorithms CGAL contains a triangulated surface mesh simplification package based making use of a half edge collapse. Supported platforms include Linux, MacOSX and Window (32 and 64 bit variants).

Meshlab [13] is advertised as an open source, portable, and extensible system for the processing and editing of unstructured 3D triangular meshes. The tool of choice for Zamani, Meshlab supports point clouds and reconstructing surfaces from point clouds. Meshlab also has support for Windows, Linux and Mac.

The Visualization and Computer Graphics library (VCG) [14] is an open source portable C++ templated library for manipulation, processing and displaying with OpenGL of triangle and tetrahedral meshes. The library has been released under a GPL license and it forms the base for tools like Meshlab and metro. It is advertised as a library that is tailored to triangular meshes with functionality like high quality quadric error collapse based simplification (reference paper). It was also found that VCG also has an implementation of vertex clustering, reading/writing to .ply files. Documentation for VCG does not specify platform support.

2.7 COMPARING MESH QUALITY

Comparing the accuracy of a simplified meshes approximation to that of the original is not easy .Identifying critical flaws when comparing hundreds of millions of points and surfaces is practically infeasible to do manually and many simplification algorithms do not provide a measure of accuracy of their approximations. For this purpose Metro [15] has been developed (Metro has since been incorporated into the Visualization and Computer Graphics library [14]). It is defined as general and simple to implement tool which numerically compares two triangle meshes describing the same surface at different levels of detail. Metro does this by finding the closest position on the surface of one model to a given position on the second using a metric called the symmetric Hausdorff distance.

The tool is ignorant of the simplification algorithm used, thus it can be used to compare the geometric accuracy simplified models regardless of the method used to simplify them. Metro is freely available and has become the standard tool for mesh comparison, cited in hundreds of papers.

2.8 DISCUSSION

QSplat seems to be successful at rendering massive meshes, however, the models it rendered only contained a few million points - not the potentially billions of points that Zamani is currently working with. QSplat does not render meshes, instead it effectively displays a multi-resolution point cloud. The Zamani group have expressed that they do not believe that a point based renderer will be sufficient to maintain the required detail and would prefer that we seek a solution that renders surfaces. Progressive meshing seemed promising, however their implementation was deemed out of the scope of an honours project.

We decided to follow a similar approach to Lindstrom and Cignoni et el, by implementing a hierarchical level of detail scheme. We divided the project into three separate components, the mesh divider, simplifier and renderer.

CGAL has an existing implementation of a triangle mesh simplification algorithm. However, Zamani expressed interest in the possibility of extending Meshlab if something new was needed or discovered during the course of our project. VCG is the back bone for Meshlab, has an implementation of a high quality triangle mesh quadric error metric decimation algorithm and it is open source. I naturally begun my implementation by experimenting with Tridecimator from VCG.

Metro will be used to measure the quality of simplified meshes, however, high detail is not necessarily a requirement of the simplifier. Meshes must be easily recognisable from a distance for the level of detail hierarchy to perform adequately. This will be evaluated visually, by critiquing the simplified meshes.

2.9 SUMMARY

The Zamani group requires a means that will allow them to display massive models on current modern hardware interactively. A task they have found difficult due to the complexity of the models. Through our background research we have decided to build a Hierarchical level of detail system. I have examined papers on mesh simplification methods and have decided to begin by experimenting with a mesh simplification implementation in the VCG library.

3 DESIGN

In this chapter, the details and justifications of the design choices made throughout the development of the simplifier tool, 'SimplFy' are specified. Requirements were identified through discussions with Zamani, my project group and by experimenting with Tridecimator. The design was revised when a greater understanding of Tridecimator, and its capabilities was achieved.

3.1 REQUIREMENTS AND INITIAL DESIGN.

The requirements were identified through discussions with Zamani, evaluation of both tridecimator and Trimesh clustering and by planning of how the suite's components would operate and interact.

The components mesh viewing suite can be separated into two categories. The pre-processor and the renderer. The pre-processor is composed of the mesh divider and simplifier, these two tools interact to create the HLOD mesh representation (Figure 7).

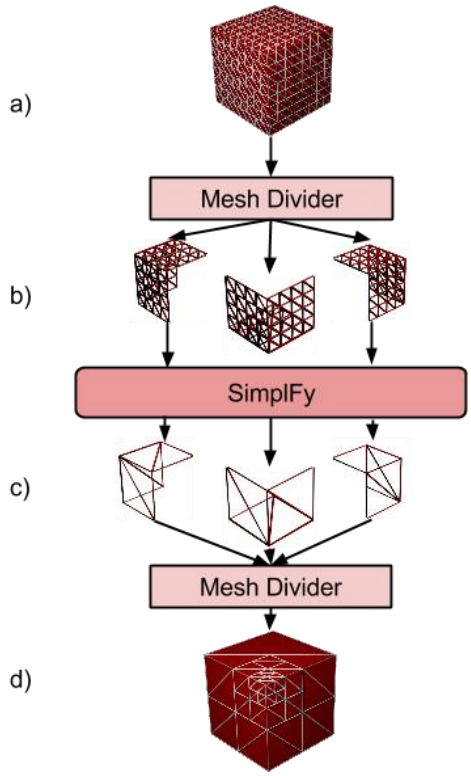


Figure 7 an overview of the interaction between the mesh divider and the simplifier. a) The original mesh. b) The mesh divided into segments. c) The simplified segments. d) The level of detail hierarchy created by stitching simplified segments together

Zamani meshes will be input in the .ply file format. The divider will divide the mesh into segments and decimate each segment using the simplifier. To do this, each segment will be saved as a .ply file with its location, decimated target (target face count) and output location specified as run time

arguments. These segments will then be processed by the simplifier, with each decimated segment output as a .ply file. The mesh divider will calculate to what degree a segment needs to be decimated as this depends on the data structure it will be using to create the hierarchical level of detail (see companion report). The boundaries formed by splitting meshes up needs to be preserved to prevent seams from forming when the segments are merged (stitched) back together.

The following requirements were identified

- Read and write mesh data stored in the Stanford (.ply) file format. Zamani store their meshes in .ply files and it is commonly used file format.
- Support for x64 systems. Zamani models may exceed multiple gigabytes, 64 bit support will remove the 4GB ram limitation of 32 bit applications. Allowing the simplifier to be used on massive files.
- Preserve mesh features. Zamani meshes contain architectural features like arches and doorways must be preserved as far as possible. This will be evaluated visually and where possible, with Metro.
- Boundaries must be preserved to prevent seams from forming when segments are simplified and then stitched back together (Figure 8).
- Support for both windows and Linux. The mesh divider is written in Java and supports both platforms.
- Vertex colour must be preserved. Some Zamani meshes contain colour data per vertex.

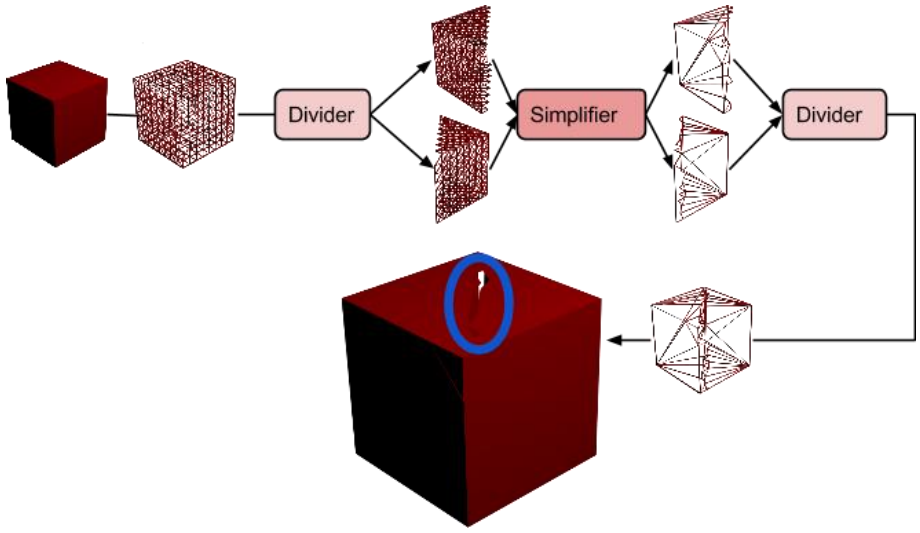


Figure 8 an example of how potential seams are created

3.1.1 Initial Design

In order to identify a starting point for the development of my project, I evaluated the two standalone triangle mesh simplification tools that are bundled with VCG. The tools were evaluated by simplifying the meshes that Zamani had supplied us with at the time (Figure 9). Each mesh was

decimated if possible to 50, 10, 1 and .01% of their original face count to exaggerate any potential problems when simplifying Zamani data.

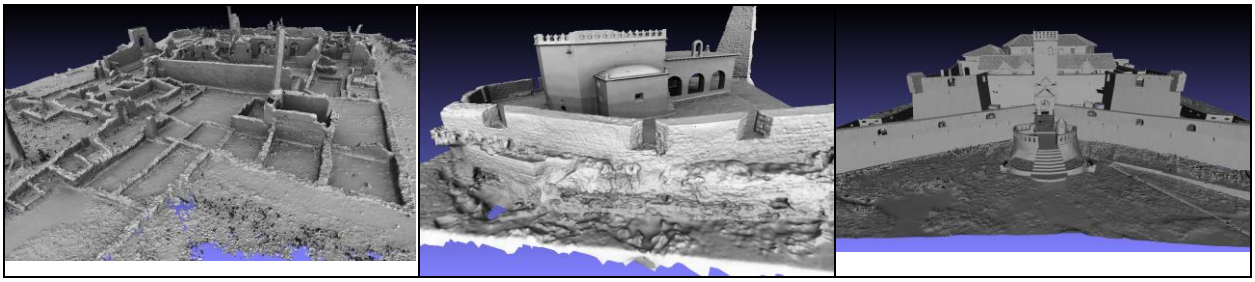


Figure 9 the models that were initially provided by Zamani. Gede (3 million faces), Chapel of Nossa (9.5 million faces) and Jago (30 million faces).

Some results of the tridecimator evaluation, shown in Figure 11 reveal potential problems with Zamani Meshes and the simplifier. Large holes have formed in b), these holes have formed due to the way the mesh was triangulated. By analysing the mesh in Meshlab, I discovered that Gede is composed of 4 aligned segments (Figure 10). When the mesh is heavily decimated the faces bordering these segments peel back revealing massive seams. The simplified mesh in c) has 135 964 faces, even though a target of 3000 was specified. Mesh colour is also lost in the simplification process. Additional results can be found in Appendix B.

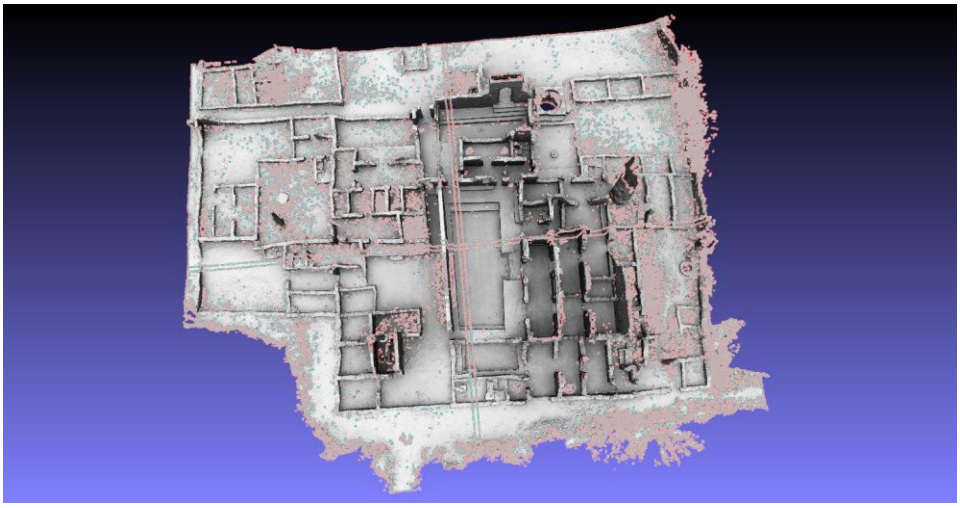


Figure 10 Revealing Gede's hidden borders in Meshlab

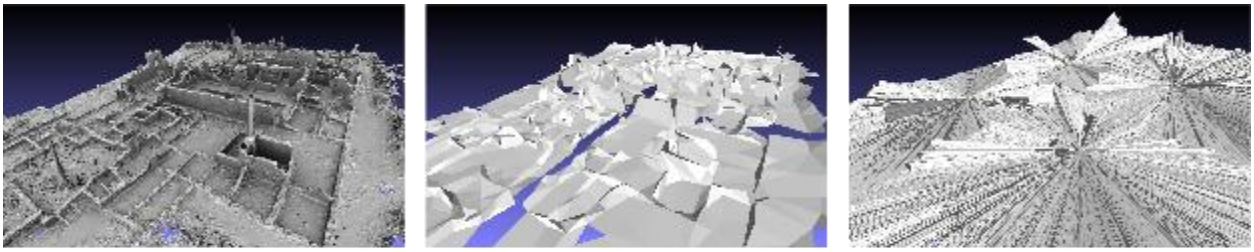


Figure 11. Different levels of simplification. a) Original Gede model 3 million faces, b) Gede simplified to 3 thousand faces without border preservation. c) Gede simplified to 3 thousand faces with border preservation.

Following the preliminary evaluation of Tridecimator and Trimesh clustering, we concluded that Tridecimator satisfied many of the requirements we had initially identified, such as: .ply file support, border preservation, high quality mesh simplification and the ability to specify a target face count. Therefore, I decided to begin by focusing on adapting Tridecimator to the suite, refining its boundary preservation and adding support for meshes with per vertex colour. Once this was successful, I had intended to integrate the triangulated surface simplification from CGAL into the application as a comparison.

The purpose of the initial design was to create a platform that could be used gain insight into the inner workings of VCG and CGAL through implementing some of their functions. This was done to better understand what was possible under the scope of this sub project, evaluating the need to for both VCG and CGAL.

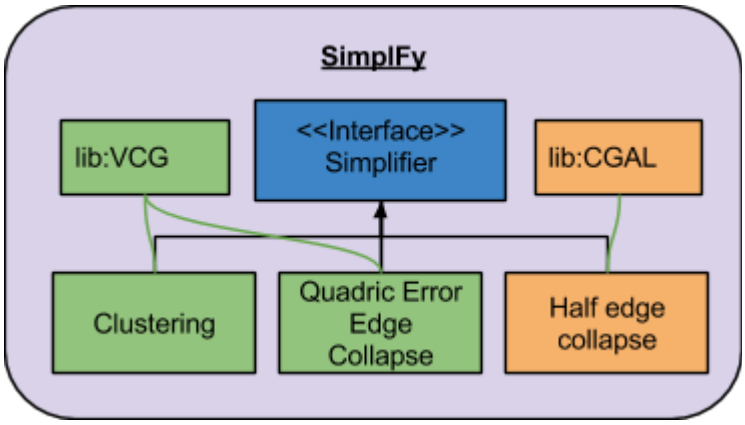


Figure 12 the initial design

Initially the simplifier application was designed to be built by reorganizing code samples from the CGAL and VCG libraries. Structured as in Figure 12, custom classes are in blue, VCG and its implementations in green and CGAL and its implementation in orange. All to be integrated into a single package if possible.

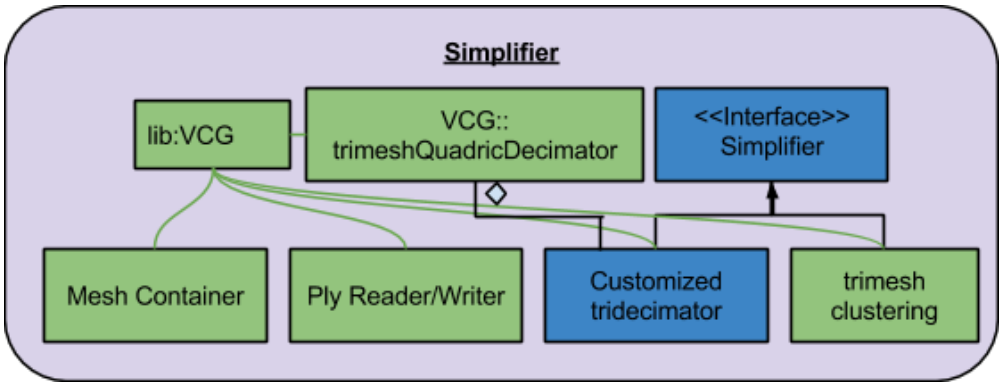
3.2 Revised Design

Successful progress of the overall system, time constraints, a greater understanding of VCG and facing difficulties during the initial implementation lead to the discovery new requirements and a revision of the design.

The implementation of CGAL was decidedly left out of this revision due to time constraints. Shifting focus to adapting VCG's tridecimator.

New requirements

- Border triangles must be specified as only those which lie outside of a bounding box specified by the mesh splitter. It was discovered during testing that some of the meshes supplied by Zamani are constructed as a collection of multiple disconnected mesh parts (cube segments). Leading to millions of extra triangles being detected as borders, increasing the minimum mesh size drastically, as well as producing graphical artefacts (Figure 11 c)).
- Unreferenced vertices must be removed after simplification. Tridecimator can clean meshes before simplification with the right argument. However, the simplification process occasionally results in faces being simplified to a single vertex. These standalone vertices were being saved after simplification, caused lighting anomalies when the rendering HLOD mesh.



Tridecimator will be used as a guide to creating a customized version of the quadric error metric algorithm. Tridecimator extends a triangle mesh quadric tridecimator class in VCG. Due to the way local optimizations are implemented with C++ traits in VCG, I have decided to follow the same principal as the sample by extending the class which defines the simplification algorithm and achieve different behaviour by adding my own methods and overriding others.

3.3 Experiment Design

In order to evaluate the effect both different arguments and the changes made to Tridecimator, I will need to compare SimplFy and Tridecimator within the mesh divider. The visual quality of the output hierarchies will be assessed and the effects of the geometry measured.

1. **Geometric accuracy** of simplified models will be evaluated by finding the Hausdorff distance between the original and simplified models with Metro.
2. **Visual quality** of the Hierarchies will be assessed by hand. Different detail levels within the hierarchy will be documented by taking screen shots of the suite's renderer at different camera distances. This will allow me to test the effect of the revised boundary preservation.

3.4 SUMMARY

Experimenting with Tridecimator and Vertex clustering independently and within the mesh viewing suite, furthered my understanding of the VCG code structure. This also led to the identification a few requirements, after which I revised my initial design. I was then able to extend Tridecimator by redefining its definition of boundary triangles, adding colour support and adjusting when unreferenced vertices are removed from the mesh.

4 IMPLEMENTATION

This chapter is an account of how the designs in chapter 3 were realized. Describing the process behind the realization as well as the tools and techniques used. To determine the scope of this project, a prior understanding of how simplification methods were implemented was required. I found the documentation of VCG and CGAL insufficient and proceeded with a more hands on approach. I began by experimenting with the quadric error edge collapse and vertex clustering implementations found in VCG, namely Tridecimator and Trimesh Clustering. I then restructured their source into my own application with adjustments that would benefit the massive mesh viewing suite.

4.1 DEVELOPMENT ENVIRONMENT

4.1.1 Programming Languages

SimplFy was written using the C++ programming language, due to its use in the VCG library. C++ Binaries can be executed from within Java, allowing the simplifier to be executed from within the mesh divider.

4.1.2 Development tools

Application development was conducted from within the Eclipse and Qt Creator integrated development environments (IDE's). Qt creator is the primary development environment used for Meshlab and the VCG library. Qt also offers a reliable step-through debugger. Simplification methods were initially compared by evaluating simplified meshes with the Metro tool and inspected visually with Meshlab.

4.2 THE IMPLEMENTATION PROCESS

4.2.1 Initial experimentation

Tridecimator and Trimesh clustering were compiled on Ubuntu x64 using "qmake" and "make" commands (qmake generates the make file from the tridecimator.pro dependencies file). On Windows 8.1 x64 Tridecimator was compiled using Eclipse. I moved all development to QT creator when I discovered that it supports .pro project files natively, and that it was used as the primary development environment Meshlab and VCG.

The effects of Tridecimator and Trimesh clustering simplification was initially tested by simplifying the Gede and Chapel of Nossa (Appendix A) supplied by Zamani, as these were available to us at time. Tridecimator failed to decimate Jago 30 million due to excess memory requirements, and was only simplified by Trimesh clustering. The meshes were simplified to 50, 10, 1 and 0.1% of the original face count. These ratios of simplification were chosen to simulate simplifying massive meshes to a fragment of their original size to exaggerate flaws in either Trimesh clustering, tridecimator or the mesh data itself. For Tridecimator, the effects of enabling boundary preservation were also recorded. Noticeable results are presented in Figure 13, the full set of results may be found in Appendix B.

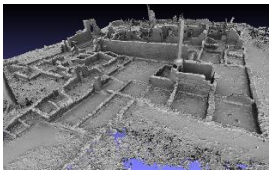
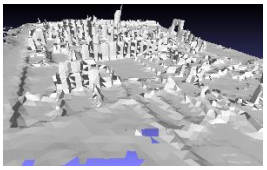
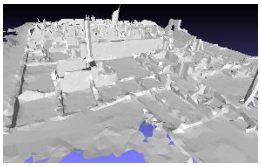
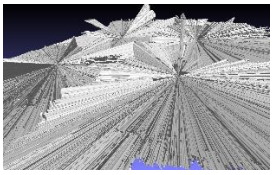
Original	Cluster	tridecimator	tridecimator with boundary preservation
			
3 000 000	28 937	30 000	135 964

Figure 13 comparison of clustering and the tridecimator. The target number of faces was 30 000

Tridecimator allows meshes to be simplified to a target face count with the option of preserving borders. Border preservation is aggressive and enabling it creates a lower bound on the simplification target that is higher than the number of border triangles. For example, all three of the meshes in Figure 13 were set to be simplified to 30 000 faces, however, Gede has 121 419 border faces. It seems that when a mesh is decimated close to its boundary face count, the non-boundary faces end up converging to only a few points, effectively rendering the mesh unidentifiable.

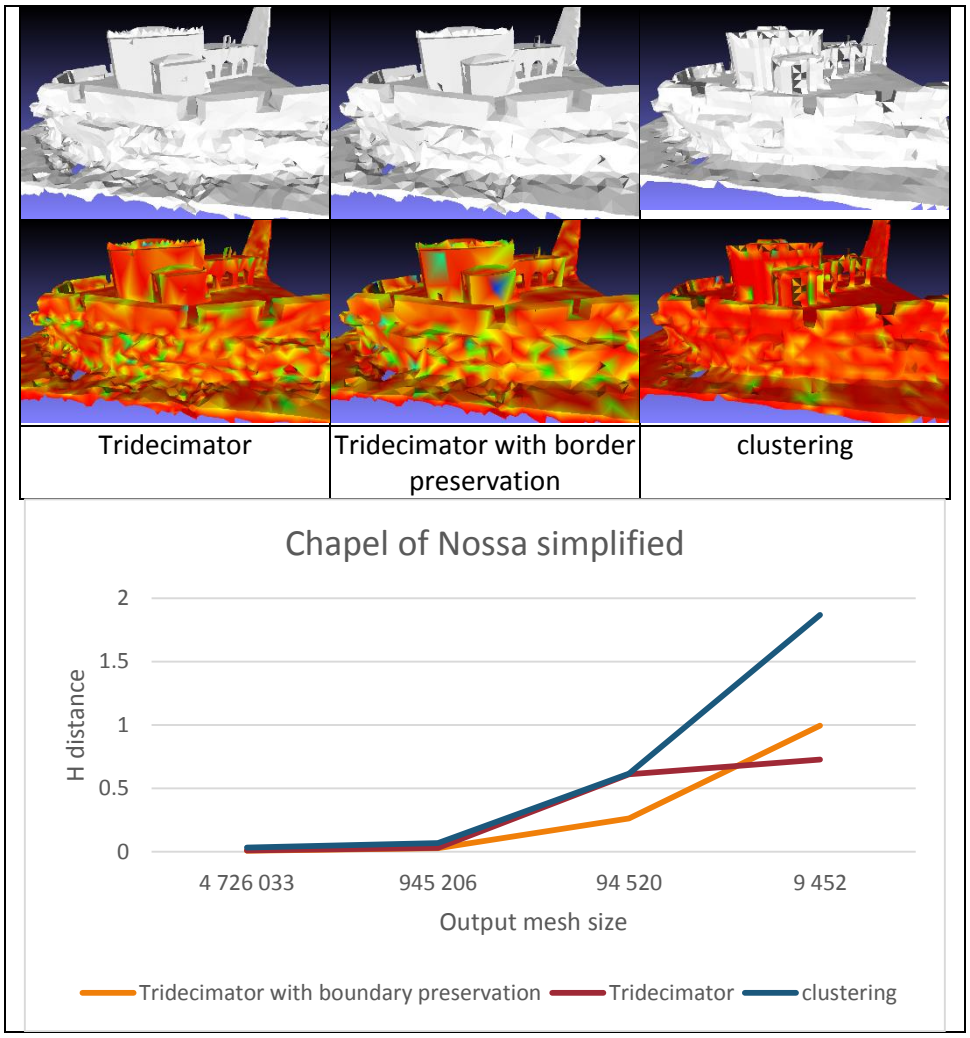


Figure 14 Results of experimenting with Metro

The simplified meshes to experiment with Metro. We can see that a higher Hausdorff distance correlates directly with the visual mesh quality in Figure 14.

4.2.2 Initial implementation

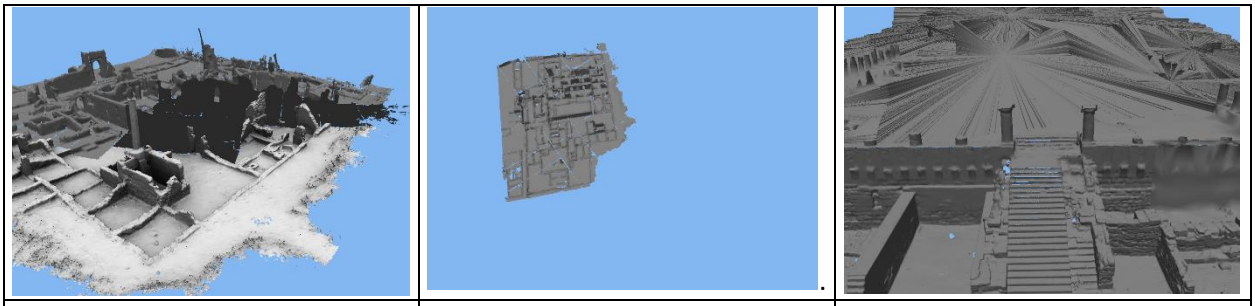
By examining the Trimesh Clustering and Tridecimator source code, I discovered that both applications use the same VCG schemes for defining mesh properties and reading mesh files. Tridecimator contains a class which extends the class containing the quadric edge collapse algorithm within VCG. This extension allows additional functionality like mesh cleaning to be added. I decided to begin implementation by restructuring the code from both Trimesh Clustering and Tridecimator into a single application. This was done to help me build an understanding of how Tridecimator was implemented, and how methods within VCG interact.

Following the initial design in Figure 12, I created a templated Simplifier interface and reorganised code from Tridecimator and Trimesh Clustering into their own respective classes. I then structured the code base so that both classes shared the same file handling and mesh properties. Once

complete, this initial application was integrated into an early version of the mesh viewing suite and tested.

4.2.3 Initial integration testing

The mesh divider was configured to save the temporary files it created when building the HLOD file. One of these temporary files always contains a simplified version of the whole mesh. I then attempted to identify potential integration artefacts by experimenting with the mesh divider. I configured the mesh divider to simplify the lowest detail mesh to 30 000 faces and used different combinations of the mesh cleaning and preserve boundary options. Noteworthy results are presented in Figure 15: Additional results can be found in Appendix C.



a) b) c)

Figure 15 Initial integration tests with Tridecimator. The mesh divider was configured to simplify its root node down to 30 000 faces, a) Gede with boundary preservation. b) Gede, c) great temple with boundary preservation.

The following additional problems were identified

- Processing a mesh while preserving boundary triangles behaved as expected when the number of target faces was much greater than the number of border faces. However, detail on some individual mesh segments was corrupted when simplified too far[2] as shown with in Great temple in Figure 15 and Gede palace in Figure 11 .
- The renderer was failing to display the correct lighting for some segments. By examining meshes simplified directly (outside of the mesh divider), I found that the Tridecimator was leaving unreferenced vertices in the mesh after decimation, this was causing the renderer to fail when calculating the values needed to illuminate surfaces on the mesh.

[2] Simplifying close to the number of boundary triangles with boundary preservation on. This causes the simplifier to cease the simplification process.

4.3 ADDING FEATURES TO TRIDECIMATOR

4.3.1 Removing duplicate and unreferenced vertices after simplification

Tridecimator already had a method for removing duplicate and unreferenced vertices, this was trivially moved to a later point in the application so that it would execute as soon as the simplification process had ended.

4.3.2 Per vertex colour support

This was achieved adding the vertex colour attribute to the mesh definition and adding an integer mask argument to the file importer and exporter (Figure 16).

```
int mask=0;
int err=vcg::tri::io::Importer<MeshType>::Open(m,filePath,mask);
//and
vcg::tri::io::ExporterPLY<MeshType>::Save(m,outfilePath,mask);
```

Figure 16 the code to enable colour support

4.3.3 Bounding box boundary preservation

Each face in the mesh segment is iterated over, if any of the faces' vertices fall outside of a specified bounding box, the face is then marked as a border face. The bounding box is defined as an argument by the mesh divider as a run time argument.

4.4 CHALLENGES

4.4.1 The language and Tools

The project required a substantial amount to revise C++ features, components and find the right tools (Qt creator) for development.

After implementing the changes in the revised design on windows with Qt creator and the visual studio compiler, the gcc compiler in Ubuntu failed. This was solved by enabling the C++11 feature set (compiling with the -std=C++11).

4.4.2 VCG

The VCG documentation found in the Meshlab documentation is out of date and the sample code found in an included tutorial for defining meshes is based on a deprecated methods.

Working through the VCG source proved challenging. Discerning how functionality was implemented from the source code was difficult due to comments being either inaccurate, written in a foreign language or insufficient. I worked around this by stepping through code using Qt creators' step through debugger.

4.5 SUMMARY

The implementation of this project occurred in two main phases. The first phase was used to an entry point into development to identify what would be possible without modifying the library itself. This included comparing different levels of simplification with Tridecimator and Trimesh clustering. In the second phase, the design was revised to further meet project requirements by adding colour support, a custom boundary detection method and insuring useless data was removed from each mesh after simplification.

5 EVALUATION

This chapter describes the experiments that were conducted to evaluate SimplFy's performance. Performance measured by comparing the output of the mesh divider when using SimplFy or Tridecimator as the simplifier.

Renders of simplified meshes were taken with Meshlab, whereas renders of the Hierarchical meshes were captured from the suite's renderer using MSI Afterburner.

5.1 EVALUATING GEOMETRIC ACCURACY

The purpose of this experiment is to quantitatively evaluate the effects of SimplFy's within the Mesh divider. The Hausdorff distance between the simplified meshes and the originals meshes will be used to measure the quality.

Gede, Nossa will be simplified out-of-core using the Mesh divider, by configuring the simplifier in as:

1. SimplFy with mesh cleaning and boundary preservation.
2. SimplFy with mesh cleaning but without border preservation.
3. Tridecimator without border preservation.

These simplified meshes will also be compared with meshes simplified by the Tridecimator directly, with and without border preservation.

5.2 EVALUATING VISUAL QUALITY

The purpose of this experiment is to confirm that the changes made to Tridecimator in SimplFy have been effective within the scope of the massive mesh viewing suite. I expect that coloured meshes will retain their colour. Rendering lighting should not fail at any point (black segments) and the boundary preservation should noticeably reduce seams. The best configuration required to attain the best quality level of detail hierarchy should be determined by experimentally testing the SimplFy on the meshes. Most importantly, in order for me to answer my research question, processing meshes that are otherwise too large to be rendered (Shama fort, Jago 350 million) should result in a mesh that is easily recognisable.

In order to achieve this, each of the Zamani Meshes was processed with the simplifier in the mesh divider configured as:

1. SimplFy with mesh cleaning and boundary preservation.
2. SimplFy with mesh cleaning but without border preservation.
3. Tridecimator without border preservation.

The mesh divider was configured to output the root node of the Hierarchy (containing a simplified version of the whole mesh). These simplified meshes were inspected visually using Meshlab.

6 RESULTS AND DISCUSSION

The following chapter presents the results of processing Zamani models with the mesh divider using either the tridecimator or the proposed SimplFy tool.

6.1 GEOMETRIC ACCURACY

Unfortunately, only two of the meshes can be compared with Metro so no statistically significant results were possible. However, SimplFy with the new bounding box boundary preservation seems to be performing worse than not having any boundary preservation at all.

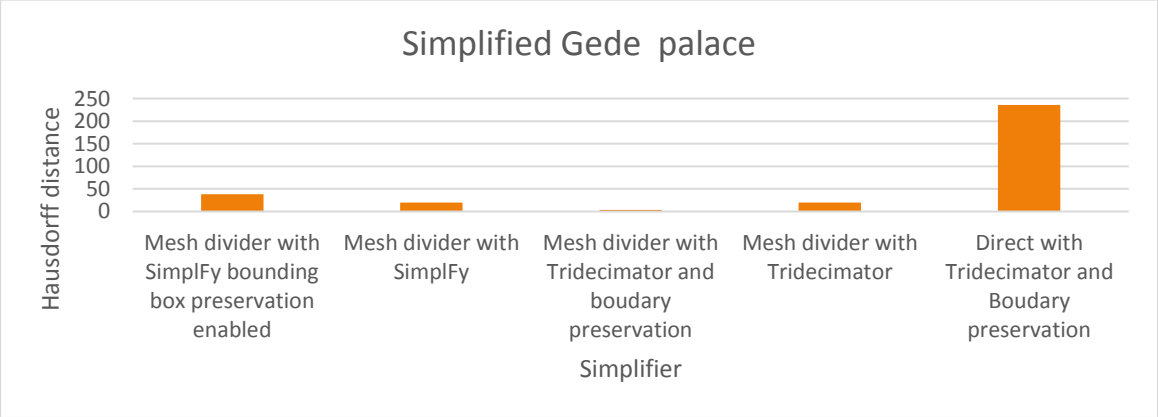


Figure 17 Results of analysing Gede simplification with Metro

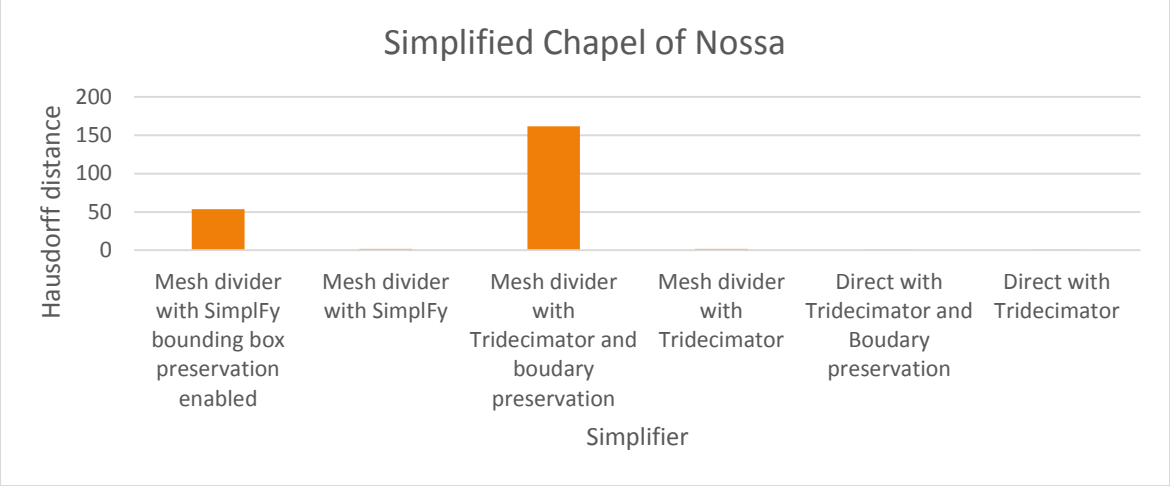
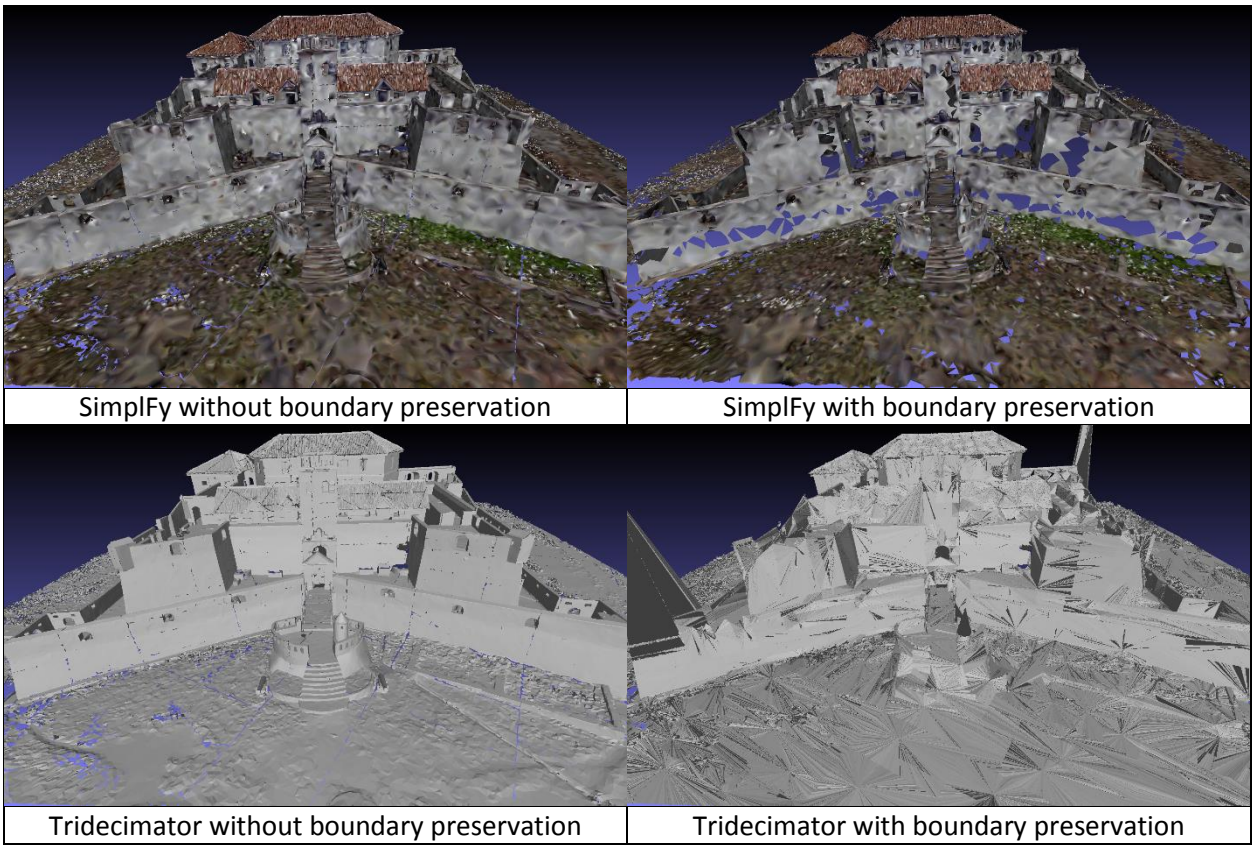


Figure 18 Results of analysing Gede simplification with Metro

6.2 VISUAL QUALITY

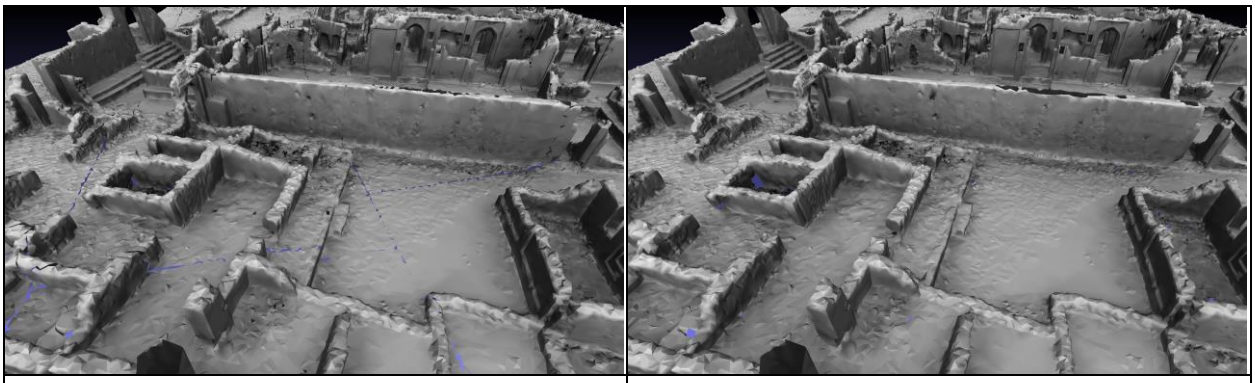
Noteworthy renders of the simplified meshes taken in Meshlab are presented Figure 19 - Figure 20 .Showing a comparison in the effects between SimplFy and Tridecimator. Useful screen captures from the suites renderer are presented in Figures below). The full set of results can be found in Appendix D.

There were no cases where the new bounding box boundary preservation produced a more appealing hierarchy, in fact, the best results were obtained without boundary preservation. Cleaning the mesh segments of any unreferenced vertices after simplification has successfully solved the lighting problem on every mesh it was tested on. Finally, mesh colour is successfully being preserved.



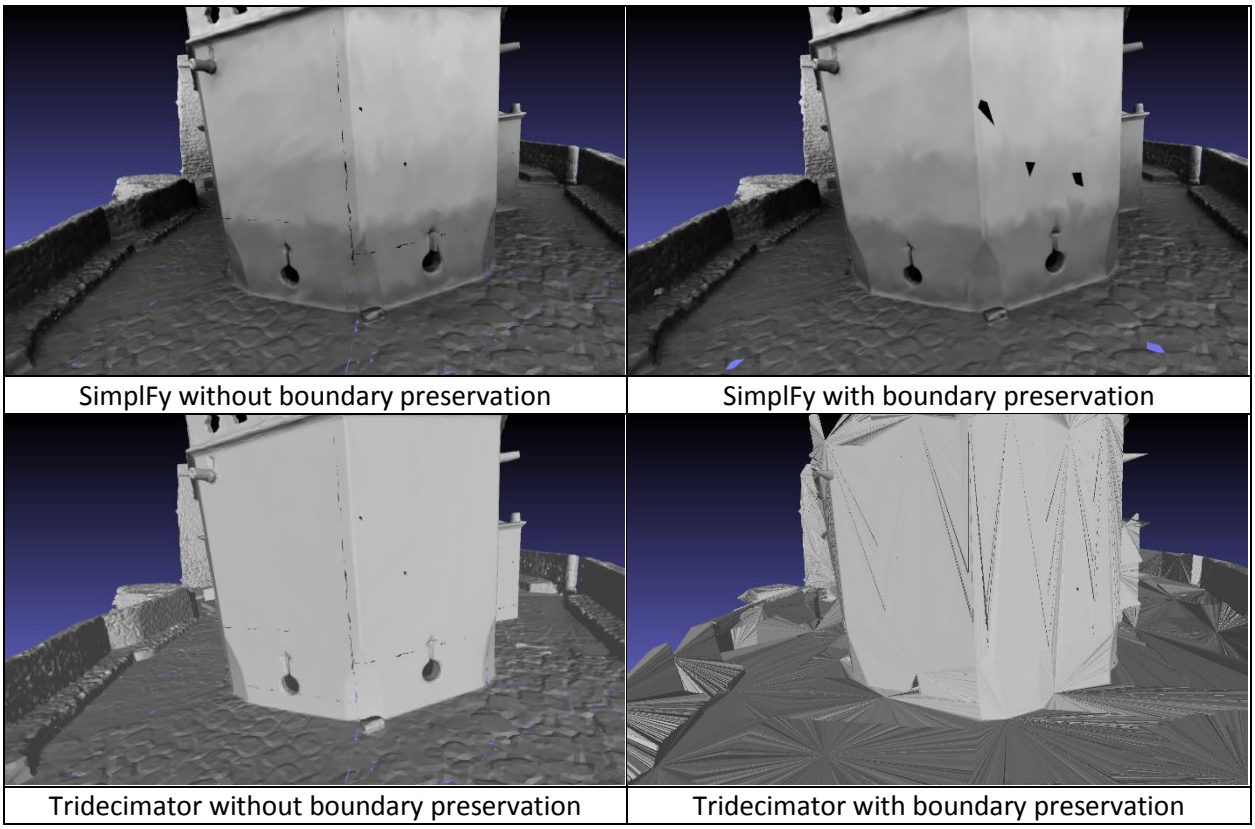
SimplFy without boundary preservation	SimplFy with boundary preservation
Tridecimator without boundary preservation	Tridecimator with boundary preservation

Figure 19 Meshlab results: Jago 30 million coloured



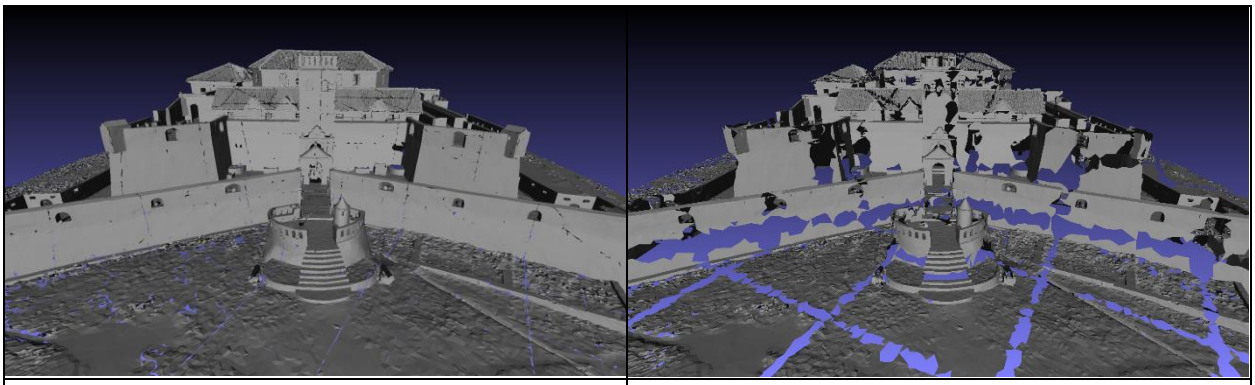
SimplFy without boundary preservation	SimplFy with boundary preservation

Figure 20 Meshlab results: Gede palace, the effects of boundary preservation

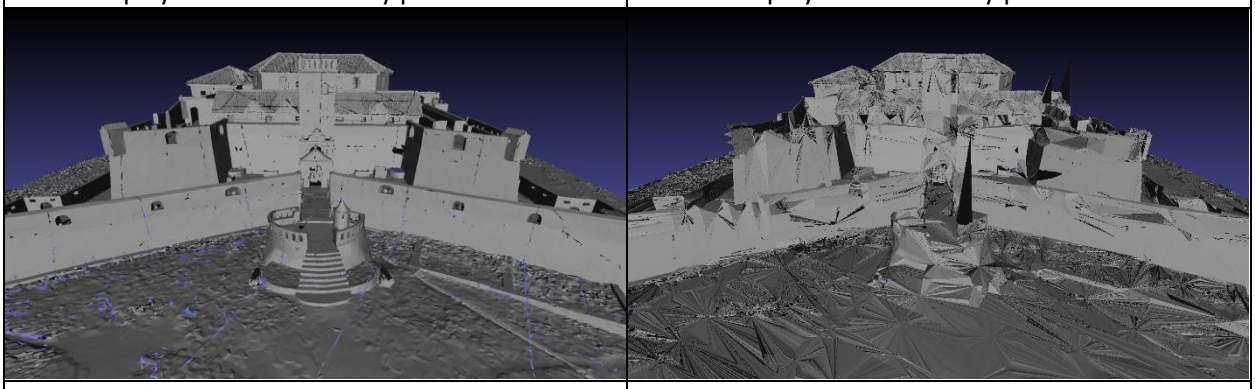


SimplFy without boundary preservation	SimplFy with boundary preservation
Tridecimator without boundary preservation	Tridecimator with boundary preservation

Figure 21 Meshlab results: Chapel of Nossa.

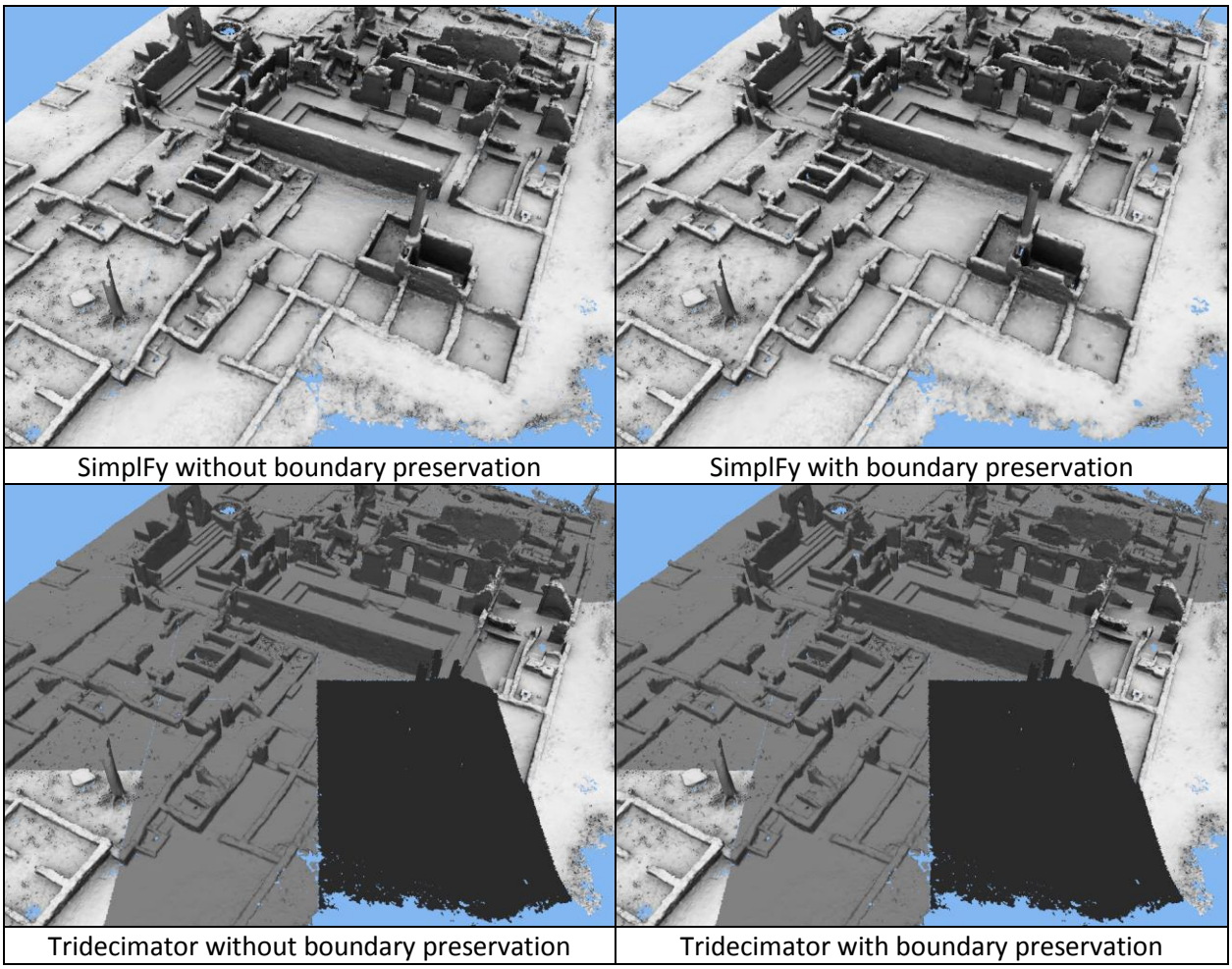


SimplFy without boundary preservation	SimplFy with boundary preservation



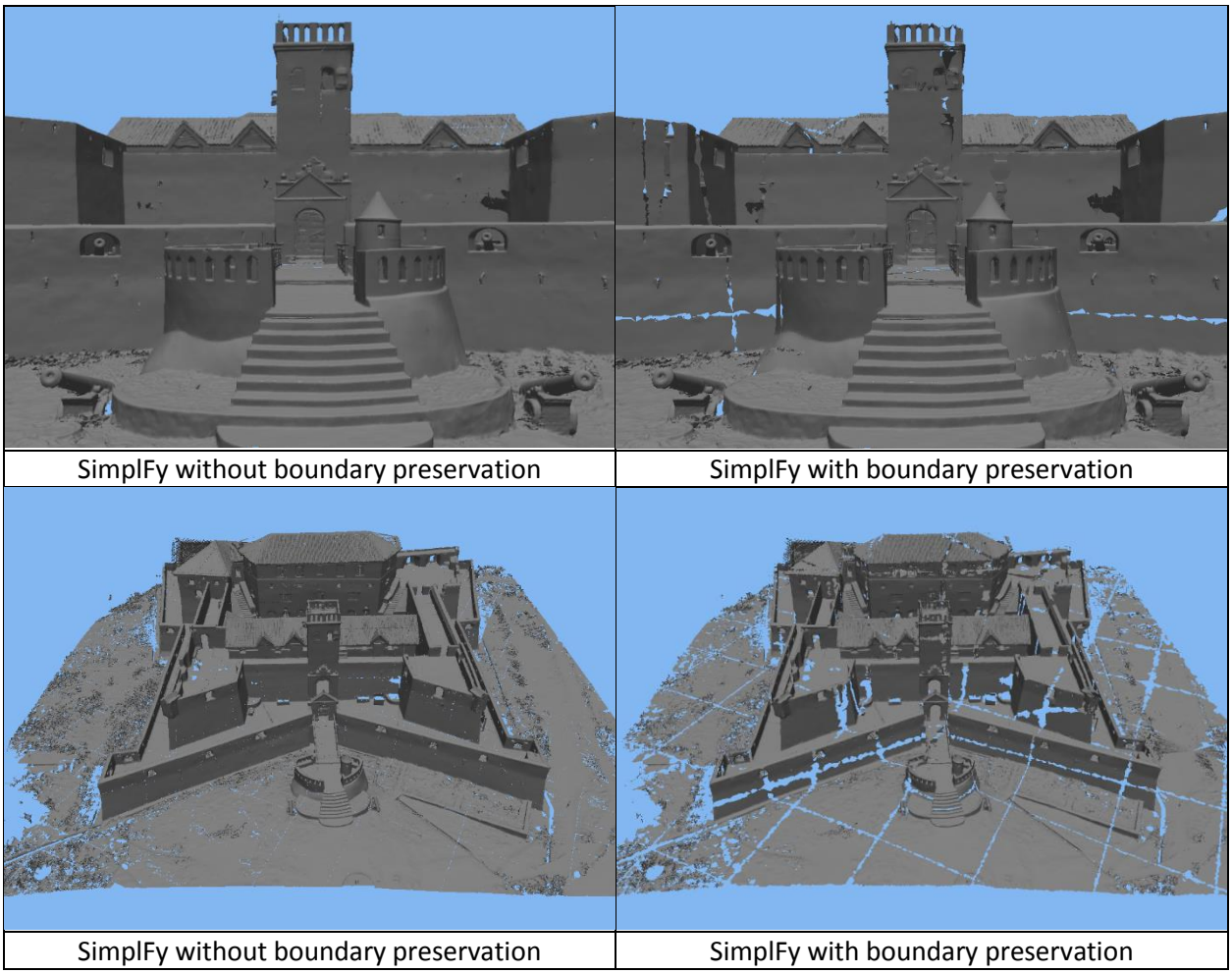
Tridecimator without boundary preservation	Tridecimator with boundary preservation

Figure 22 Meshlab results: Jago 30 million



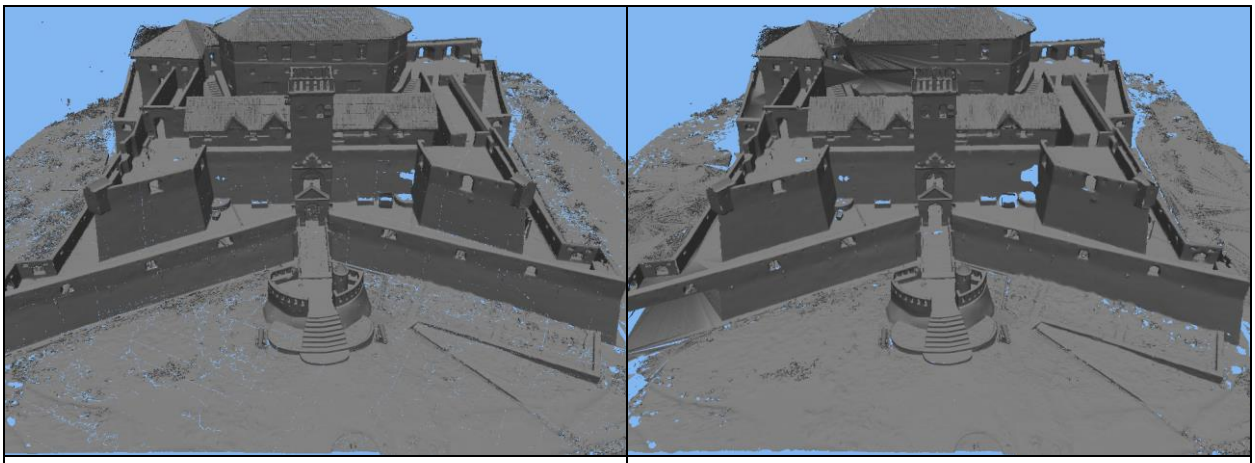
SimplFy without boundary preservation	SimplFy with boundary preservation
Tridecimator without boundary preservation	Tridecimator with boundary preservation

Figure 23 Mesh viewing suite results: Gede



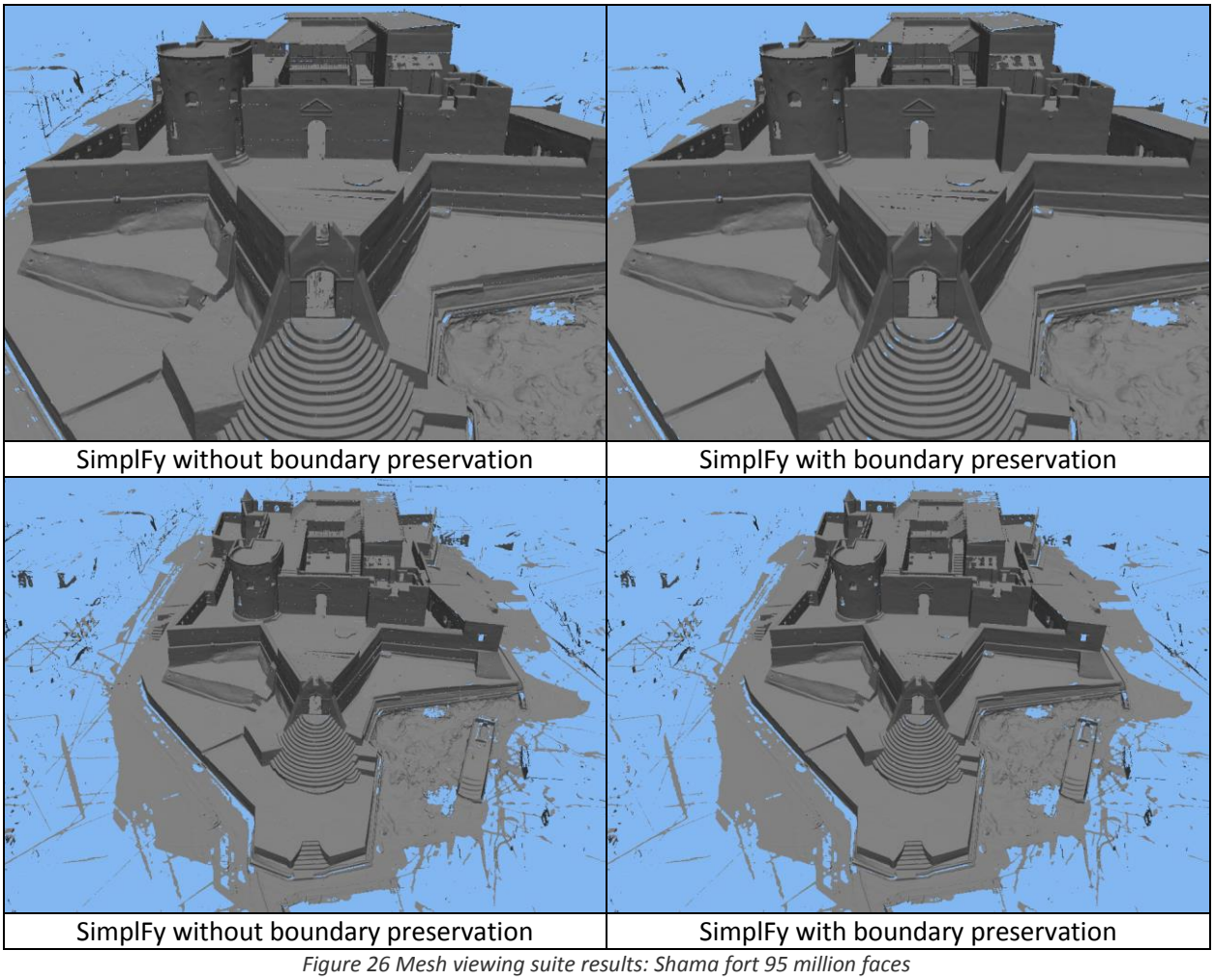
SimplFy without boundary preservation	SimplFy with boundary preservation
SimplFy without boundary preservation	SimplFy with boundary preservation

Figure 24 Mesh viewing suite results: Jago 30 million faces



SimplFy without boundary preservation	SimplFy with boundary preservation

Figure 25 Mesh viewing suite results: Jago 350 million faces



SimplFy without boundary preservation	SimplFy with boundary preservation
SimplFy without boundary preservation	SimplFy with boundary preservation

Figure 26 Mesh viewing suite results: Shama fort 95 million faces

6.3 Discussion

Mesh cleaning after simplification and adding colour support has worked successfully in every test. Colour is preserved and segments are never processed without lighting.

Massive seams have eroded away at the Jago 30 million model (Figure 24) when processed with the modified boundary preservation. These seams do not appear in other meshes and do not align with the practically invisible seams created by the mesh divider when boundary preservation is disabled. After inspecting the original Jago 30 million model in Meshlab, we discovered that the model is composed with multiple mesh segments that are aligned with each other, but the segment geometry is not connected. There are small gaps between these segments that are growing rapidly as an unintended side effect of redefining the boundary preservation. The same stretched triangle artefacts (Figure 11) are also present in the 350 million Jago. Like the original artefacts in Gede, these are cause by simplifying the mesh too close to the number of boundary triangles. The simplification ratio is controlled by the mesh divider.

The simplifier allows the massive mesh suite to process models exceeding 350 million faces, so that they can be rendered without losing finer detail. The Tridecimator has successfully been adapted to work in our massive mesh viewing suite.

6.4 Summary

The mesh divider was configured to output meshes using Tridecimator and SimplFy as the simplifier on different runs. The two meshes that could be evaluated by metro did not produce complimentary results. SimplFy was compared with Tridecimator again to evaluate its effect on the massive mesh viewing suite as a whole. Mesh detail hierarchies were inspected visually from the custom renderer and Meshlab at a set of distances to evaluate the level of detail quality and determine the best configuration for the simplifier.

It was discovered that modifying the way Tridecimator preserved boundaries had a negative effect on the visual quality, and that in fact the seams we tried to avoid a small enough that they barely factor in.

Enabling vertex colour and mesh cleaning after simplification operated as expected and I could conclude that Tridecimator could be adapted to the massive mesh viewing suite.

7 CONCLUSION

7.1 SUMMARY OF REPORT

The report aimed to document the experimental design, implementation and evaluation of a mesh simplification tool called SimplFy. The tool was needed as a component in our massive mesh viewing suite, and was implemented by extending an existing simplification tool. Tridecimator was extended by adding a new method for defining boundary triangles, adding support for coloured vertices and setting the application to remove unreferenced vertices after simplification. Different run time arguments were tested with SimplFy to find the best configuration for our massive mesh viewing suite. This also allowed us to evaluate SimplFy within the context of the mesh viewing suite.

7.2 CONCLUSION

The suite was tested with multiple meshes supplied by the Zamani group ranging from 3 million faces to 350 million faces. This meshes still retained a large amount of detail and remained recognisable. The need for a refined boundary preservation was also found to diminish when the mesh was simplified to a higher level of detail.

I have successfully adapted an existing simplification algorithm into our massive mesh viewing suite and shown that it returns visually appealing meshes.

7.3 FUTURE WORK

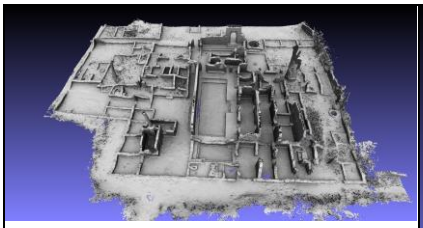
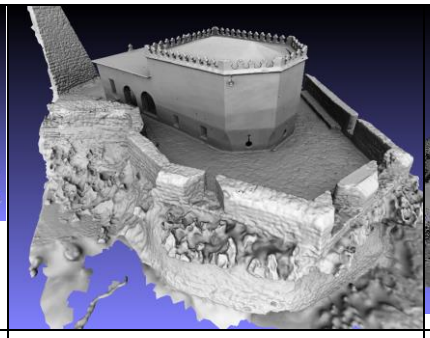
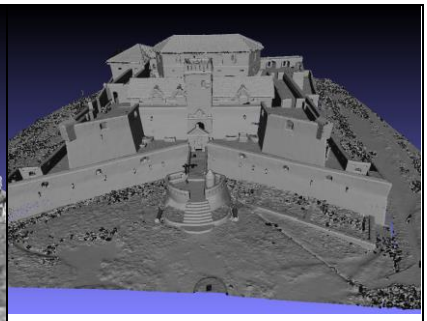
With the successful adaption of Tridecimator into the massive mesh viewing suite, there is still potential to increase the quality of the different levels of detail. This could be done by revising the boundary preservation mechanic again perhaps having different quadric weights for boundary triangles that would have fallen within the bounding box. A scheme for preventing small holes in the mesh could be investigated. A heuristic for adjusting the face count when target is close to the number of boundary triangles could be investigated. The CGAL simplification could be implemented and compared with the current version of Tridecimator/SimplFy. Texture support could be added to increase colour surface detail.

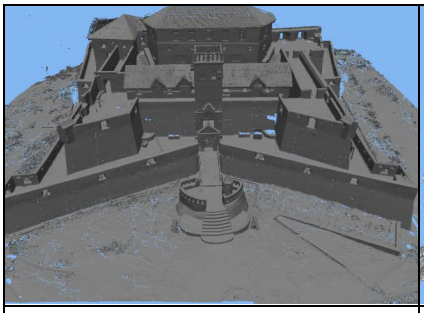
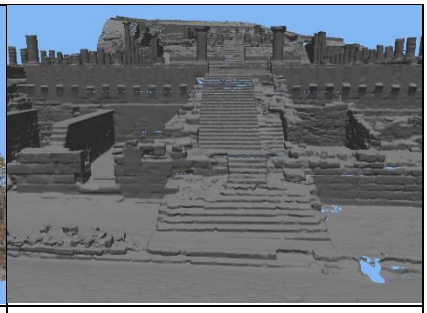
APPENDICES

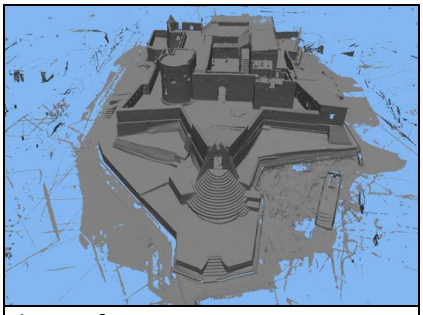
APPENDIX A - ZAMANI MESHES

This appendix describes the different Zamani Meshes that were used to evaluate this project.

Renders

		
Gede Palace	Chapel of Nossa	Jago 30 million

		
Jago 350 million	Jago 30 million coloured	Great temple


Shama fort

Mesh attributes

Model	**Verticex colour**	**Face count**	**Border triangles**	**File size**
Gede Palace	Y	~3 000 000	~125 000	68MB
Chapel of Nossa	Y	~9 450 000	~9000	212MB
Jago 30 million	N	~30 000 000	~2 500 000	632MB
Jago 350 million	N	~350 000 000	n/a	6.5GB
Jago 30 million coloured	Y	~30 000 000	~2 500 000	597MB
Great temple	N	~30 000 000	n/a	637MB
Shama fort		~95 000 000	n/a	1.7GB

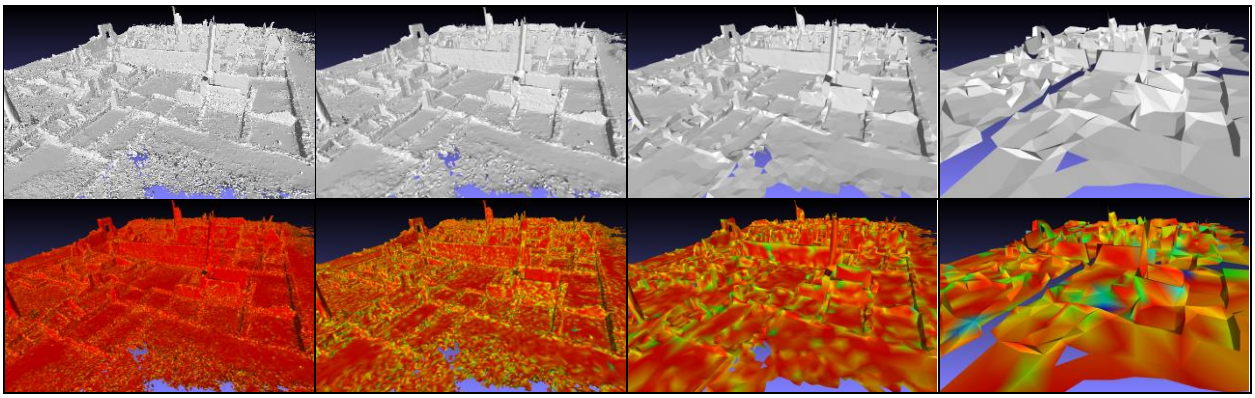
APPENDIX B - RESULTS OF INITIAL EXPERIMENTATION

Tri is the Tridecimator. Clustering is Trimesh Clustering

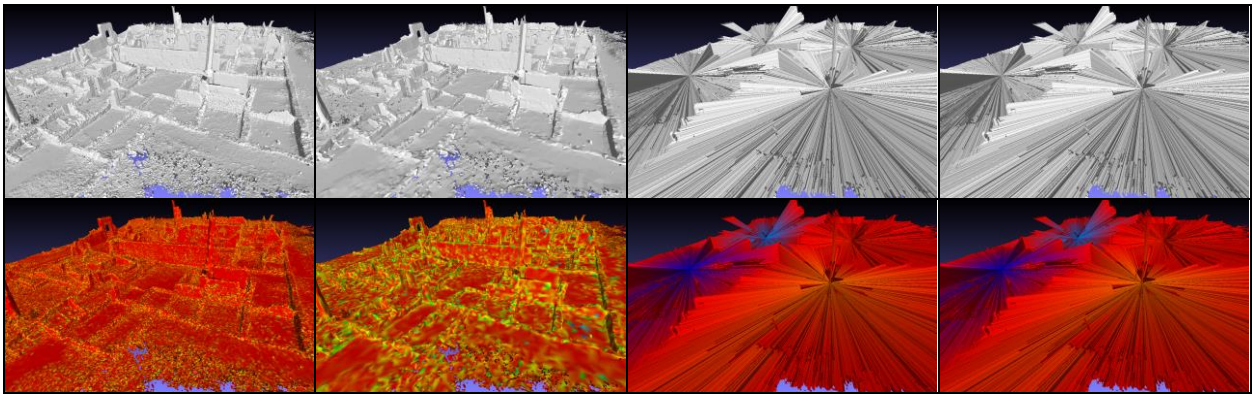
- -P is clean mesh
- -By is preserve boundary triangles
- -Bn
- -k specifics number of clusters the mesh is simplified into in clustering

Model	Simplifier	simplifier arguments	Target face count	Face count	Vertex Count	Hausdorff Distance
Gede	Tri	-By	1 500 000	1 500 000	807 312	213.382019
		-By	300 000	300 000	217 998	235.828766
		-By	30 000	142 714	135 964	3331.513916
		-By	3 000	142 714	135 964	3331.513916
		-Bn	1 500 000	1 499 999	786 806	935.191833
		-Bn	300 000	3 000 000	175 537	1614.295654
		-Bn	30 000	29 999	39 303	3014.657471
		-Bn	3 000	2 999	25 382	4367.446777
	Clustering	-k100000000		1 537 258	776 748	647.985779
		-k7000000		321 737	160 368	953.220825
		-k200000		28 937	13 633	1864.375
		-k7000		2 952	1 345	4157.42627
Nossa	Tri	-By	4 726 032	4 726 032	2 364 339	0.009559
		-By	945 206	945 206	473 947	0.029875
		-By	94 520	94 520	48 724	0.264555
		-By	9 452	9 452	6 457	0.994493
		-Bn	4 726 033	4 726 032	2 364 158	0.010942
		-Bn	945 206	945 206	473 306	0.03121
		-Bn	94 520	94 520	47 671	0.612031
		-Bn	9 452	9 452	5 014	n/a
	Clustering	-k600000000		4 764 944	2 381 374	0.033214
		-k40000000		955 660	476 653	0.069889
		-k1100000		92 748	45 924	0.61715
		-k37000		9 720	4 608	n/a

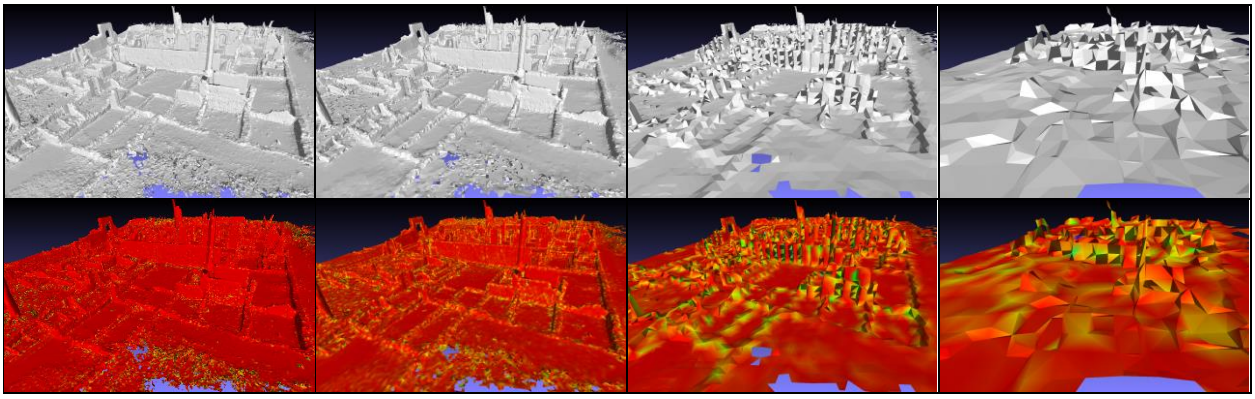
Gede tri –Bn



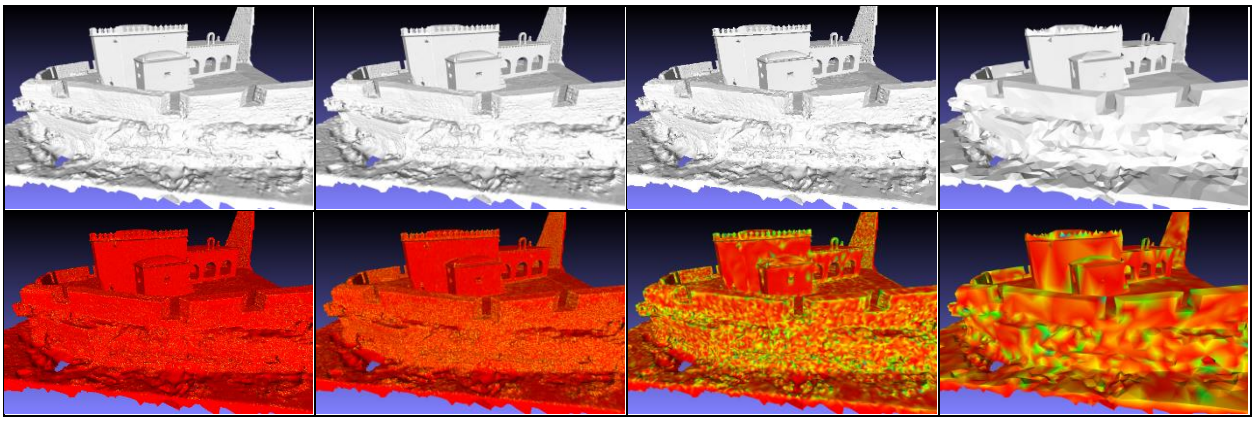
Gede tri –By



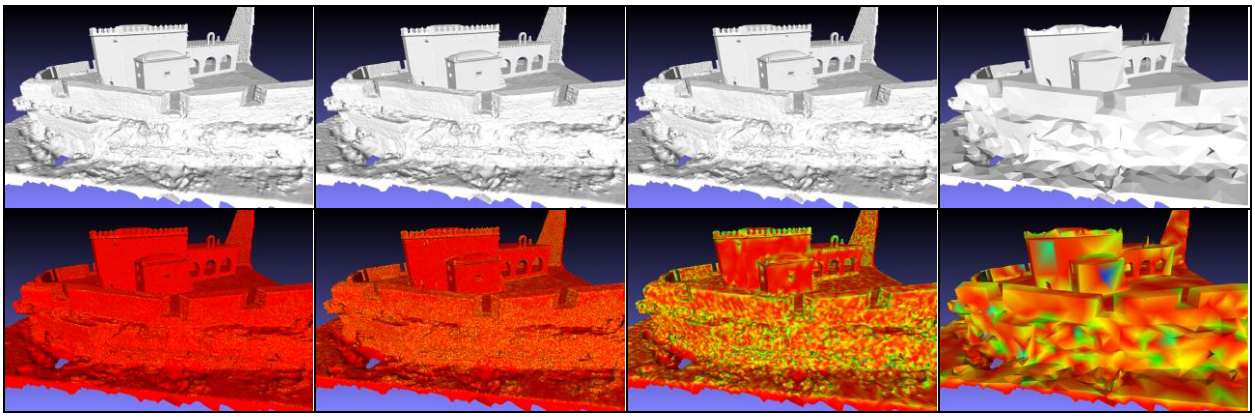
Gede clustering



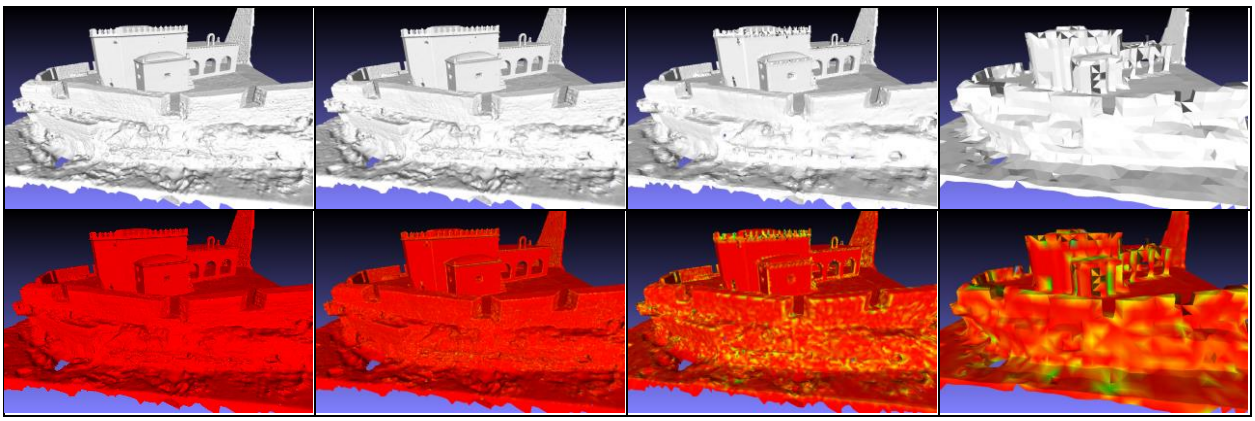
Nossa tri –Bn



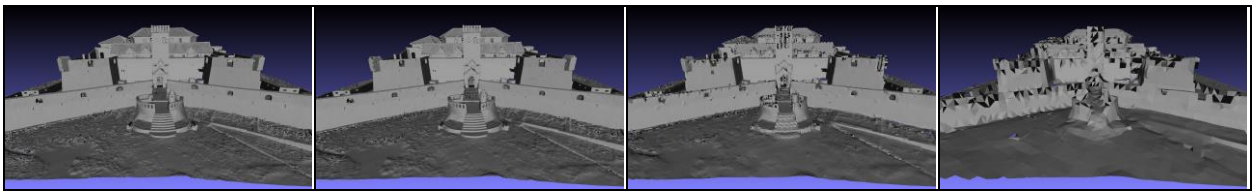
Nossa tri –By



Nossa clustering



Jago Clustering

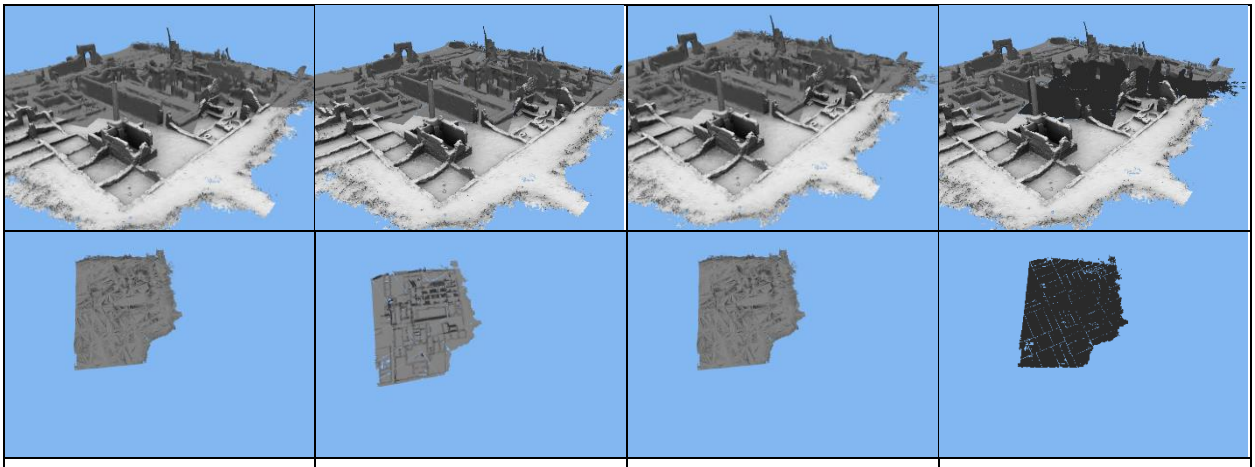


APPENDIX C – RESULTS OF EARLY IMPLEMENTATION TESTING

RESULTS OF INITIAL INTEGRATION WITH TRIDECIMATOR

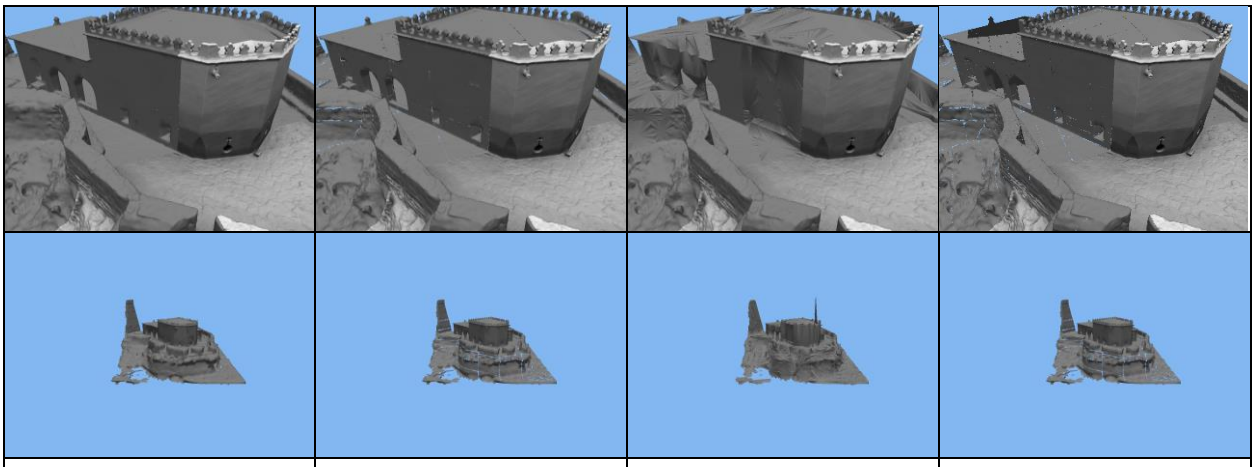
The mesh divider was set to simplify the hierarchical representation down to 30 000 faces using Tridecimator as the simplifier. Screen grabs of the models are presented below. –P enables mesh cleaning, -By enables border preservation (conversely, -Bn disables it).

Gede –P –By, -Bn, –By, -P -Bn



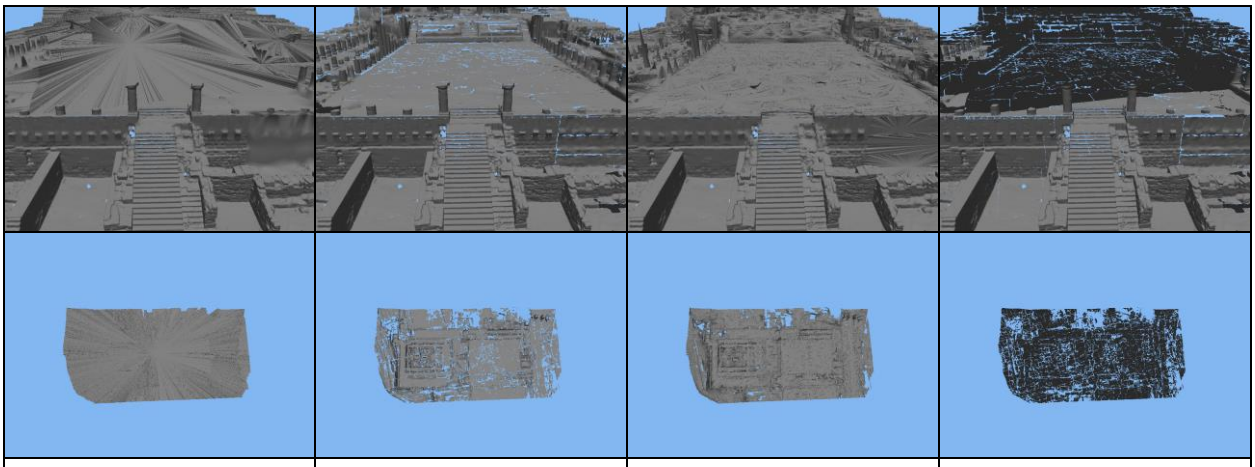
-P -By	-P -Bn	-By	-Bn

Nossa



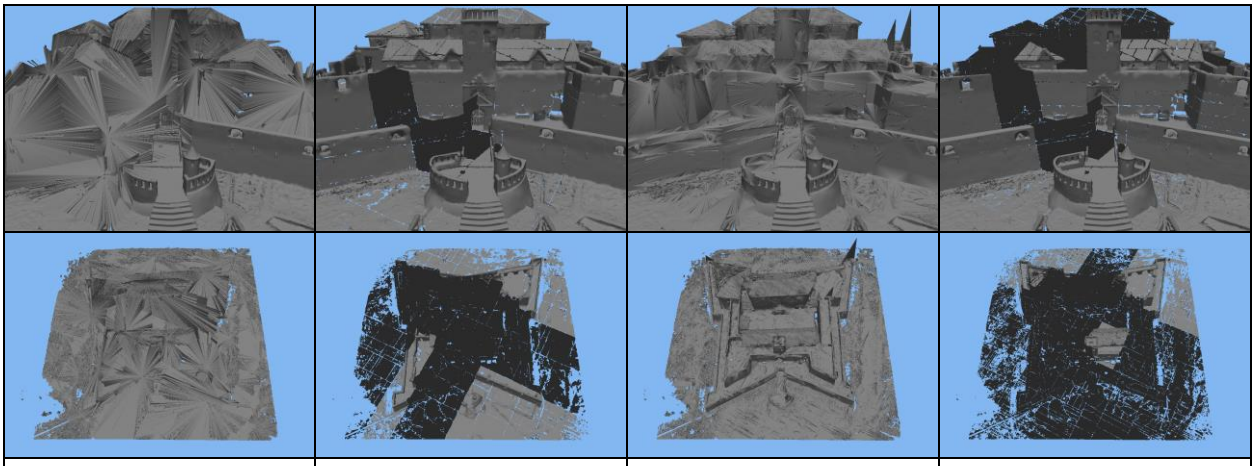
-P -By	-P -Bn	-By	-Bn

Great Temple



-P -By	-P -Bn	-By	-Bn

Jago

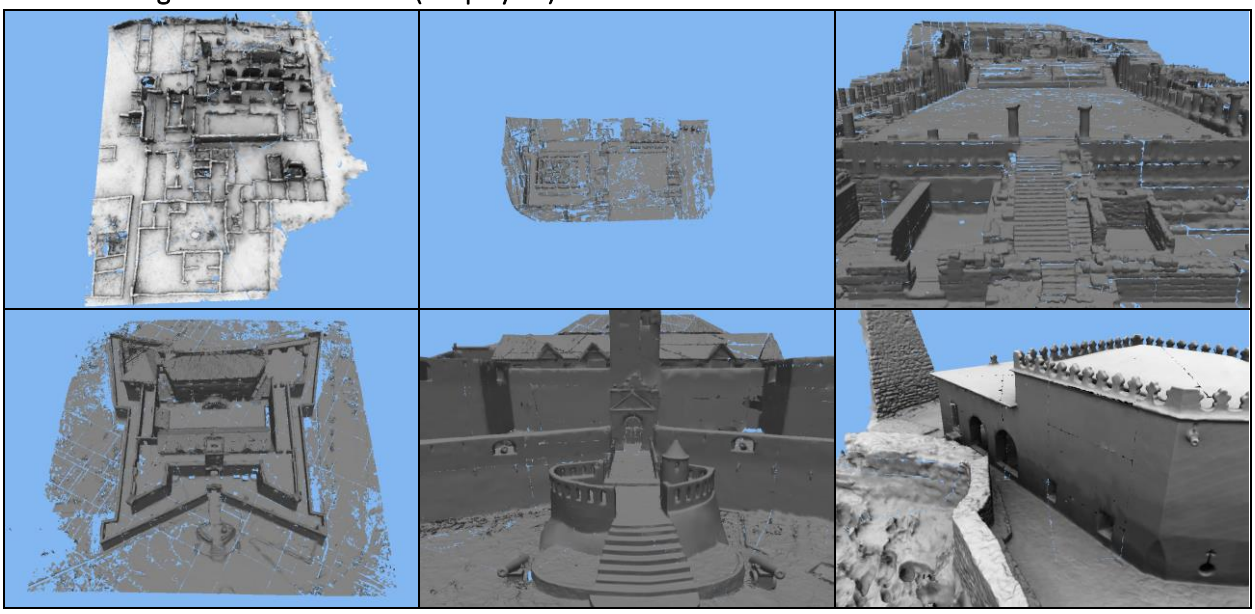


-P -By	-P -Bn	-By	-Bn

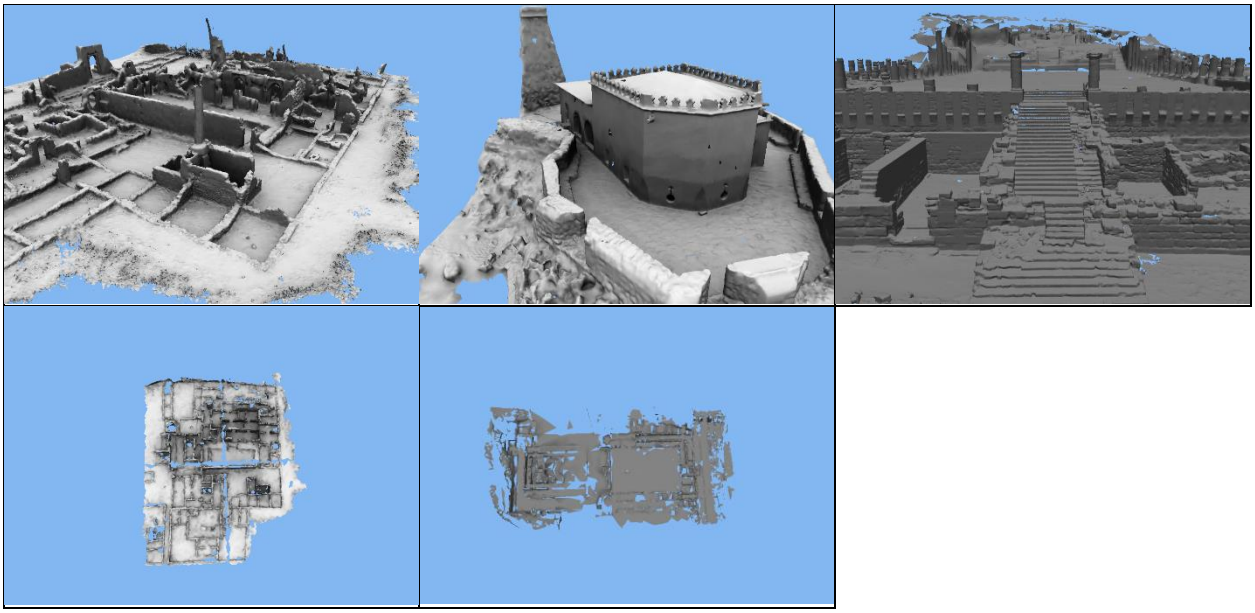
INITIAL RESULTS OF SIMPLFY IMPROVEMENTS

The mesh divider was set to simplify the hierarchical representation down to 30 000 faces using SimplFy as the simplifier. The following records the results of initial testing changes.

Mesh cleaning and colour enabled. (SimplFy –P)



Bounding box border preservation (SimplFy -P –By)



APPENDIX D RESULTS OF EVALUATION TESTS

RESULTS FROM GEOMETRIC EVALUATION

These results were generated by testing SimplFy and Tridecimator in the final version of the Mesh divider. The root node on the hierarchy was set to be simplified to the default 400 000 faces. The mesh divider was set to output temporary files so that the simplified root note could be extracted from the hierarchy. These simplified meshes were evaluated with metro. The simplifier and options are displayed below each image.–By enables the bounding box border preservation in SimplFy (default border preservation in Tridecimator).

Metro results

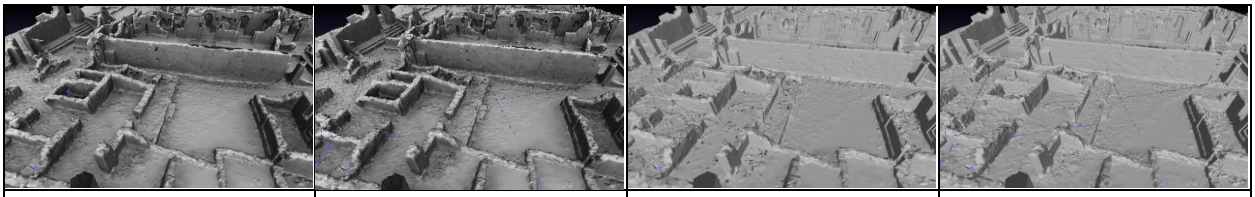
			Hausdorff distance
Gede	Mesh deivider and SimplFy	P -By	38.097351
		P Bn	19.434902
	Mesh deivider and Tridecimator	Bn	19.434902
		By	3.775464
	Direct with Tridecimator	By	235.828766
		Bn	1219.634888
Nossa	Mesh deivider and SimplFy	P -By	53.518326
		p Bn	1.951767
	Mesh deivider and Tridecimator	Bn	1.951767
		By	161.518616
	Direct with Tridecimator	By	0.051602
		Bn	0.051602

RESULTS FROM VISUAL EVALUATION

These results were generated by testing SimplFy and Tridecimator in the final version of the Mesh divider. The root node on the hierarchy was set to be simplified to the default 400 000 faces. The simplifier and enabled options are displayed below each image. For SimplFy –By enables the bounding box border preservation

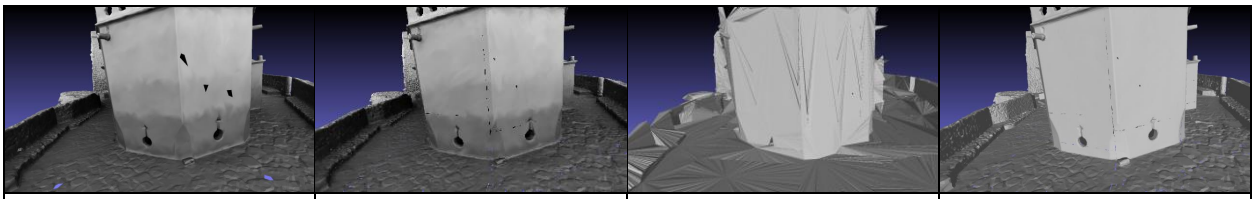
Visual inspections of extracted root node in Meshlab

Gede Palace



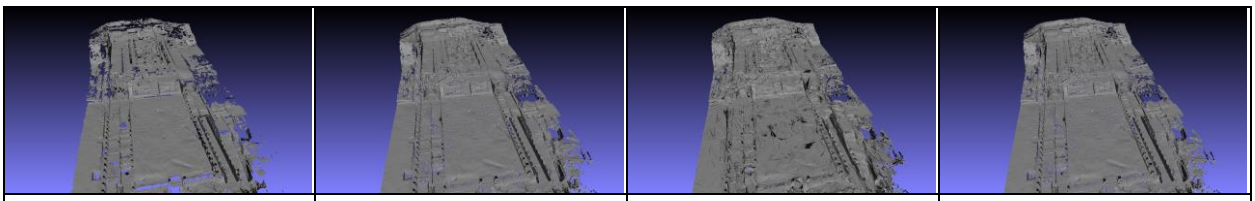
SimplFy –P -By	SimplFy –P -Bn	Tridecimator –By	Tridecimator –P -Bn

Chapel of Nossa



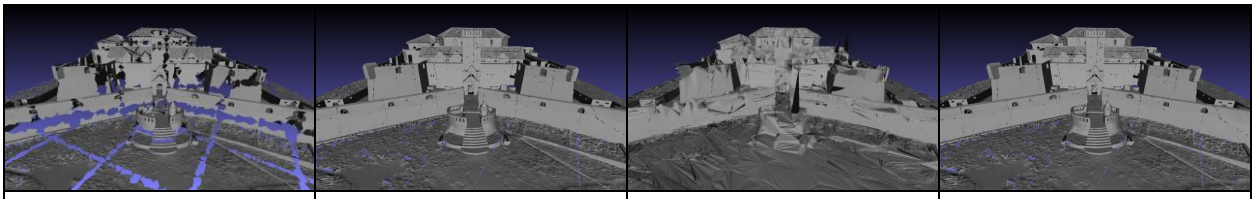
SimplFy –P -By	SimplFy –P -Bn	Tridecimator –By	Tridecimator –P -Bn

Great temple



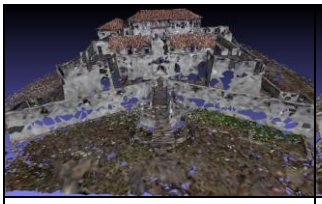
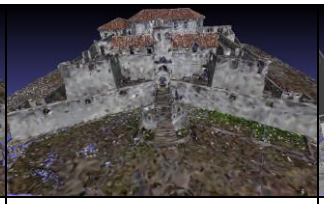
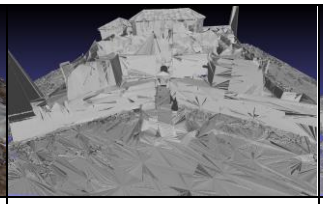
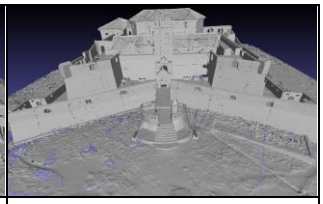
SimplFy –P -By	SimplFy –P -Bn	Tridecimator –By	Tridecimator –P -Bn

Jago 30 Million



SimplFy –P -By	SimplFy –P -Bn	Tridecimator –By	Tridecimator –P -Bn

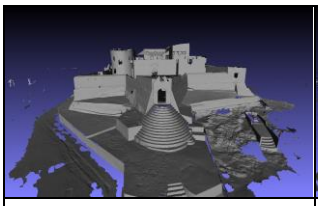
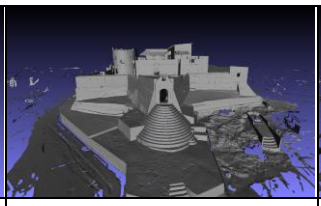
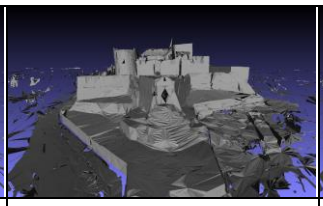
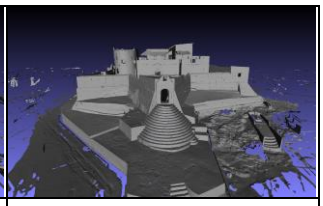
Jago Colour 30 Million

			
SimplFy –P -By	SimplFy –P -Bn	Tridecimator –By	Tridecimator –P -Bn

Jago 350 Million

			
SimplFy –P -By	SimplFy –P -Bn	Tridecimator –By	Tridecimator –P -Bn

Shama Fort 95 Million

			
SimplFy –P -By	SimplFy –P -Bn	Tridecimator –By	Tridecimator –P -Bn

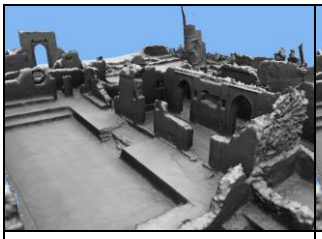
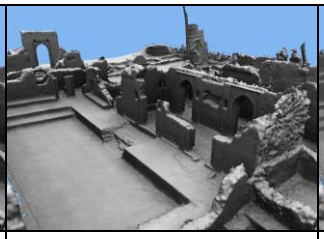
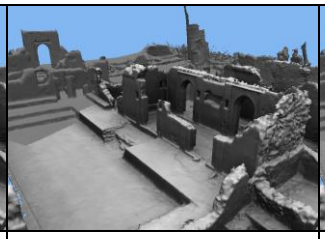
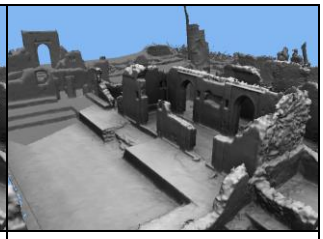
Visual inspection of Hierarchies from the renderer

These results were generated by testing SimplFy and Tridecimator in the final version of the Mesh divider. The root node on the hierarchy was set to be simplified to the default 400 000 faces. Screenshots were taken at 3 different distances, giving a close-up, medium and far (distant) view. The simplifier and enabled options are displayed below each image.

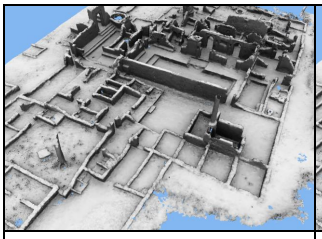
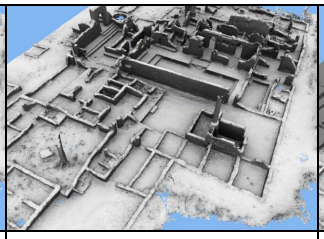
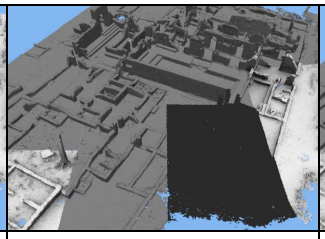
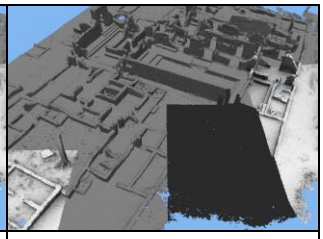
–P enables mesh cleaning, -By enables border preservation, -Bn disables border preservation.

Note SimplFy uses a bounding box border definition.

Gede close-up

			
SimplFy –P -By	SimplFy –P -Bn	Tridecimator –By	Tridecimator –P -Bn

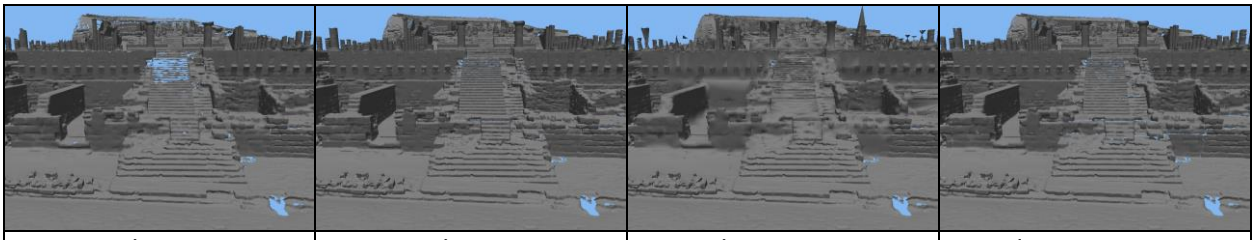
Gede medium

			
SimplFy –P -By	SimplFy –P -Bn	Tridecimator –By	Tridecimator –P -Bn

Gede Far

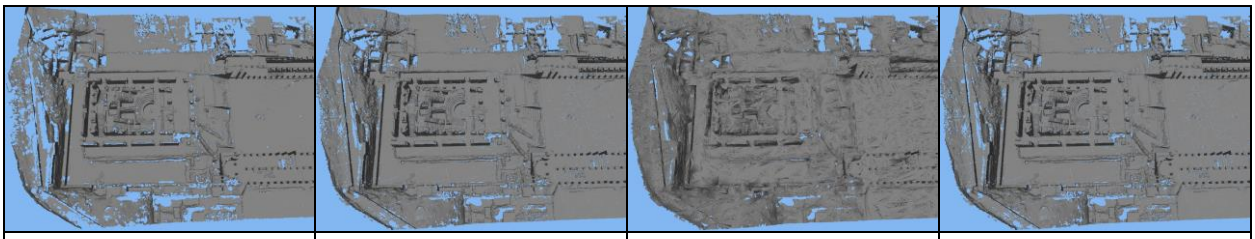
			
SimplFy –P -By	SimplFy –P -Bn	Tridecimator –By	Tridecimator –P -Bn

Great temple close up



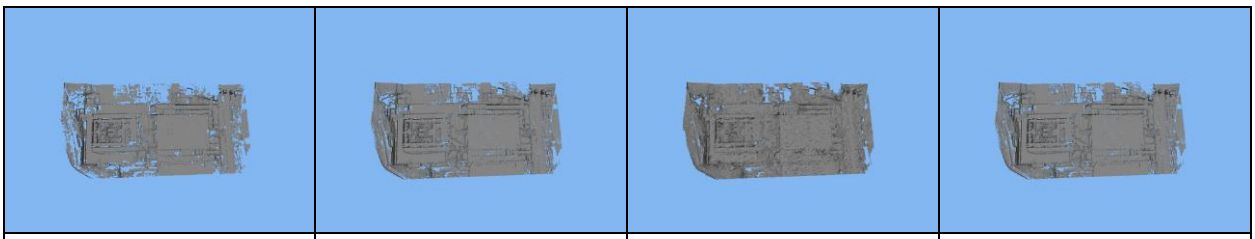
SimplFy –P -By	SimplFy –P -Bn	Tridecimator –By	Tridecimator –P -Bn

Great temple medium



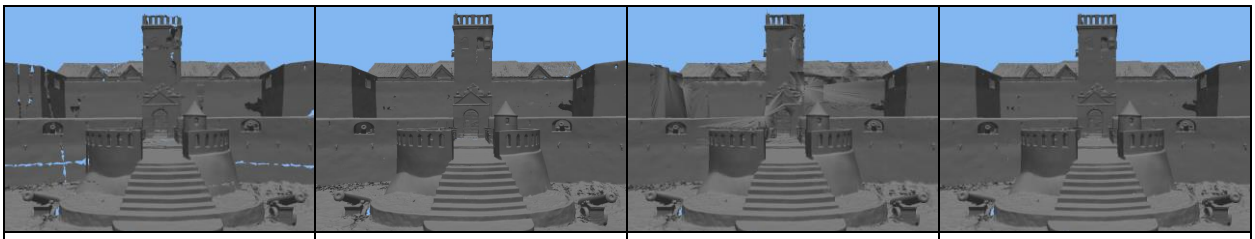
SimplFy –P –By	SimplFy –P -Bn	Tridecimator –By	Tridecimator –P -Bn

Great temple far



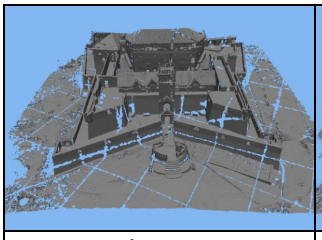
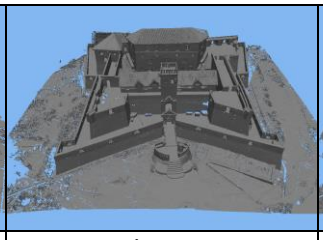
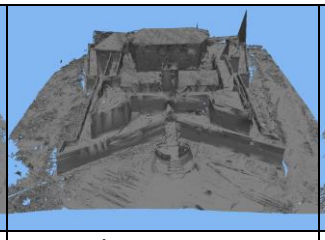
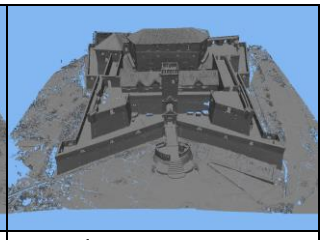
SimplFy –P -By	SimplFy –P -Bn	Tridecimator –By	Tridecimator –P -Bn

Jago 30 million close

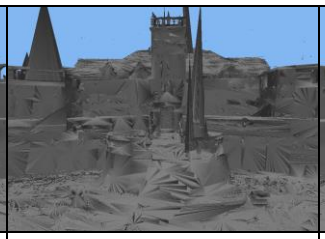
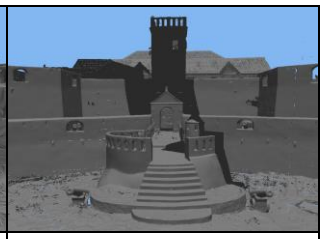


SimplFy –P –By	SimplFy –P -Bn	Tridecimator –By	Tridecimator –P -Bn

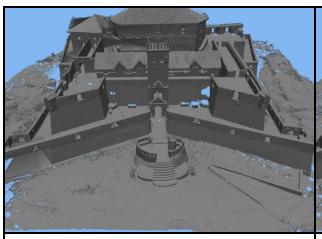
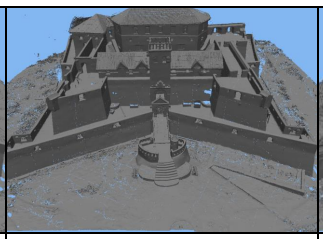
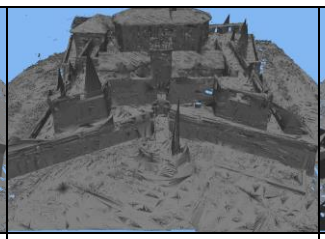
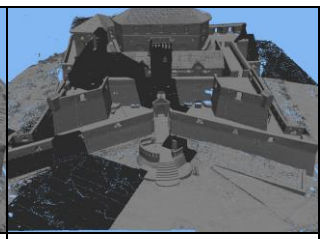
Jago 30 million far

			
SimplFy –P –By	SimplFy –P -Bn	Tridecimator –By	Tridecimator –P -Bn

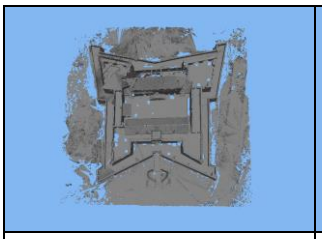
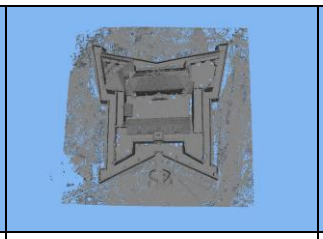
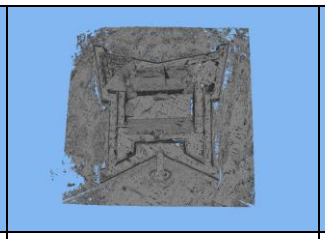
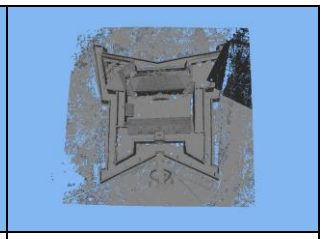
Jago 350 million close

			
SimplFy –P –By	SimplFy –P -Bn	Tridecimator –By	Tridecimator –P -Bn

Jago 350 million medium

			
SimplFy –P –By	SimplFy –P -Bn	Tridecimator –By	Tridecimator –P -Bn

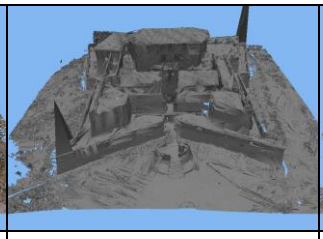
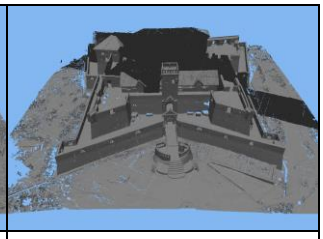
Jago 350 million far

			
SimplFy –P –By	SimplFy –P -Bn	Tridecimator –By	Tridecimator –P -Bn

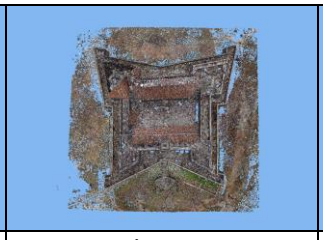
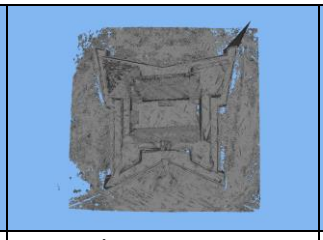
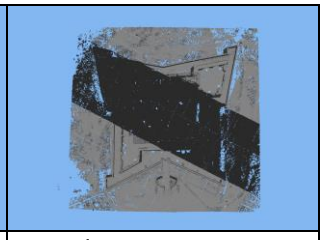
Jago 30 million colour close

			
SimplFy –P –By	SimplFy –P -Bn	Tridecimator –By	Tridecimator –P -Bn

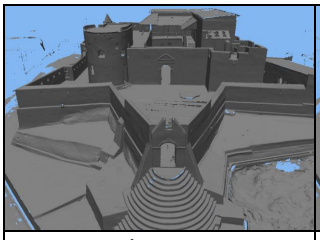
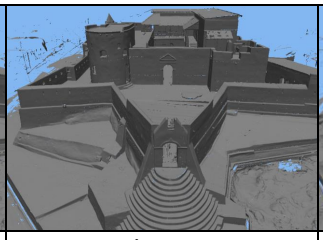
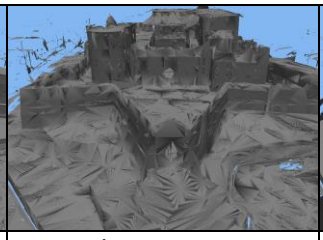
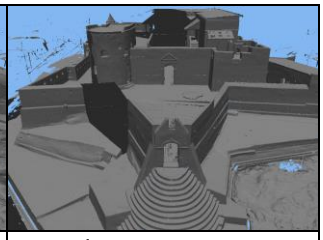
Jago 30 million colour medium

			
SimplFy –P –By	SimplFy –P -Bn	Tridecimator –By	Tridecimator –P -Bn

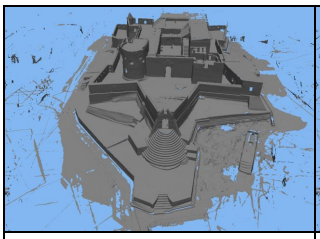
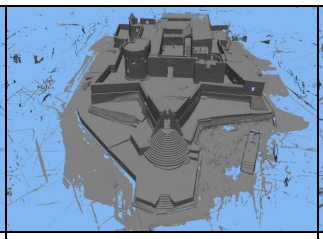
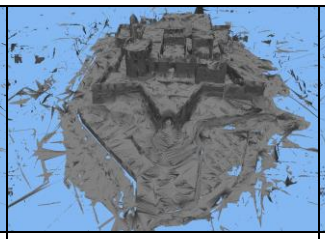
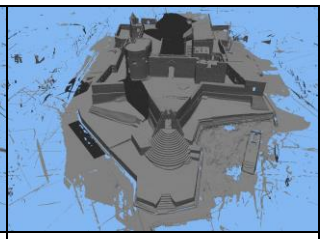
Jago 30 million colour far

			
SimplFy –P –By	SimplFy –P -Bn	Tridecimator –By	Tridecimator –P -Bn

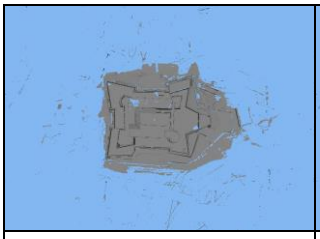
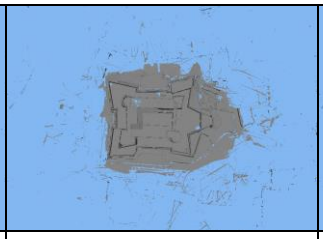
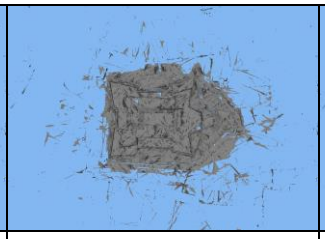
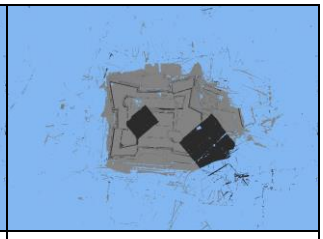
Shama fort close

			
SimplFy –P –By	SimplFy –P -Bn	Tridecimator –By	Tridecimator –P -Bn

Shama fort medium

			
SimplFy –P –By	SimplFy –P -Bn	Tridecimator –By	Tridecimator –P -Bn

Shama fort far

			
SimplFy –P –By	SimplFy –P -Bn	Tridecimator –By	Tridecimator –P -Bn

8 REFERENCES

[1] H. H. C. B. R. S. R. W. S. Rüther, “From Point Cloud to Textured Model, the Zamani Laser Scanning Pipeline in Heritage Documentation. SAJG.,” 2012. [Online]. Available: http://www.sajg.org.za/index.php/sajg/article/view/20.

[2] C. a. L. M. a. K. A. Stein, “Spatial Data Structures for Accelerated 3D Visibility Computation to Enable Large Model Visualization on the Web,” 2014. [Online]. Available: http://doi.acm.org/10.1145/2628588.2628600.

[3] H. Samet, Applications of Spatial Data Structures, Addison-Wesley, 1990.

[4] M. R. J. D. C. A. V. ,. B. W. R. H. David Luebke, Level of Detail for 3D Graphics, Morgan Kaufmann Publishers, 2002.

[5] C. a. M. D. a. B. W. V. Erikson, “HLODs for Faster Display of Large Static and Dynamic Environments,” in *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, ACM, 2001, pp. 111--120.

[6] S. a. L. M. Rusinkiewicz, “QSplat: A Multiresolution Point Rendering System for Large Meshes,” in *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, New York, NY, US, ACM Press/Addison-Wesley Publishing Co., 2000, pp. 343 - 352.

[7] H. Hoppe, “Progressive Meshes,” in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, New York, NY, USA, ACM, 1996, pp. 99 - 108.

[8] P. Lindstrom, “Out-of-core Simplification of Large Polygonal Models,” in *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, New York, NY, USA, ACM Press/Addison-Wesley Publishing Co., 2000, pp. 259-262.

[9] P. Cignoni, C. Montani, C. Rocchini and R. Scopigno, “External memory management and simplification of huge meshes," Visualization and Computer Graphics,” *IEEE Transactions on,* vol. 9, no. 4, pp. 525-537, 2003.

[10] C. M. R. S. P. Cignoni, “A comparison of mesh simplification algorithms,” *Computers & Graphics,* vol. 22, no. 1, pp. 37-54, 1998.

[11] M. a. H. P. S. Garland, “Surface Simplification Using Quadric Error Metrics,” in *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press/Addison-Wesley Publishing Co., 1997, pp. 209-216.

[12] “CGAL,” [Online]. Available: https://www.cgal.org/.

[13] [Online]. Available: http://meshlab.sourceforge.net/.

[14] “VCG,” [Online]. Available: http://vcg.isti.cnr.it/~cignoni/newvcglib/html/.

[15] P. R. C. a. S. R. Cignoni, “Metro: Measuring Error on Simplified Surfaces,” *Computer Graphics Forum,* no. 17, pp. 167-174, 1998.