Massive Mesh Simplification and Visualisation

Research Proposal

Daniel Burnham-King, Justin Cossuti and Benjamin Meier

University of Cape Town
Department of Computer Science

June 17, 2014

1 Project Description

1.1 Background

The Zamani Project, 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 or the building or terrain being scanned.

The point cloud is cleaned by hand of 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 [1]. The final surface can be very detailed and can contain billions of triangles. The Zamani Project researchers use existing mesh processing software, such as MeshLab and ArcGIS, to navigate and interact with the resulting models, but the models often contain too

[1] Mesh - The structure of a model in 3 dimensional space. Consists of a list of vertex positions and a list of triangles connecting these vertices

many triangles to render at interactive frame rates, making manipulation difficult. Dense point clouds can be decimated to reduce the amount of geometry and increase rendering speed however this has harmful effects on details such as crevices, engravings and other surface detail which are particularly important to researchers.

This leads to the need for a more effective method for displaying the highly detailed meshes while maintaining interactive frame rates [2]. The solution will also allow the models to be displayed for tourism and educational purposes.

1.2 Problem Statement

The meshes generated by the Zamani Project contain between 3 million and 8 billion vertices. In order to interact with such large meshes, they are currently split into multiple smaller meshes so that each can be viewed individually using software tools such as MeshLab. This is a time consuming task that yields no additional benefit and is inconvenient as the team cannot quickly navigate an entire mesh, but must rather continuously swap between different mesh files.

In our project we propose a solution for viewing the large models created by the Zamani project, without the need to break the meshes down into smaller parts. We will investigate whether the proposed solution will be able to improve the Zamani team's workflow and reduce the amount of administrative overhead currently required to process and interact with the models. This will allow the Zamani team to view the entire model as a whole as well as small details such as cracks, crevices, and surface detail. The benefits of such a suite include reduced administrative processing time required for each site scan, the ability to interact with entire meshes and can provide walk- and fly-throughs of large models for tourism and educational purposes.

Thus the research question can be stated as: "Can arbitrarily large architectural models be simplified [3] [4] and rendered in real-time with little noticeable loss of detail?". We further break this down into research questions for the individual sections.

[2] Interactive frame rate - defined within the scope of our project to be 15 frames per second.

[3] Mesh simplification - The automatic removal of vertices and triangles from a model's mesh, thereby resulting in a loss of detail, but also decreasing its size on disk and the time taken to render the mesh. In the context of this project, it will be done during the preprocessing stage.

[4] Level-of-detail - Refers to the decreasing complexity of a 3D model as it moves further away from the viewer. A model's mesh is incrementally simplified (nodes in the mesh are removed, resulting in less triangles) during a preprocessing stage. During rendering, the low-detailed version is initially displayed to the viewer and is iteratively swapped out for the next incrementally complex version as the distance between the model and the viewer becomes smaller.

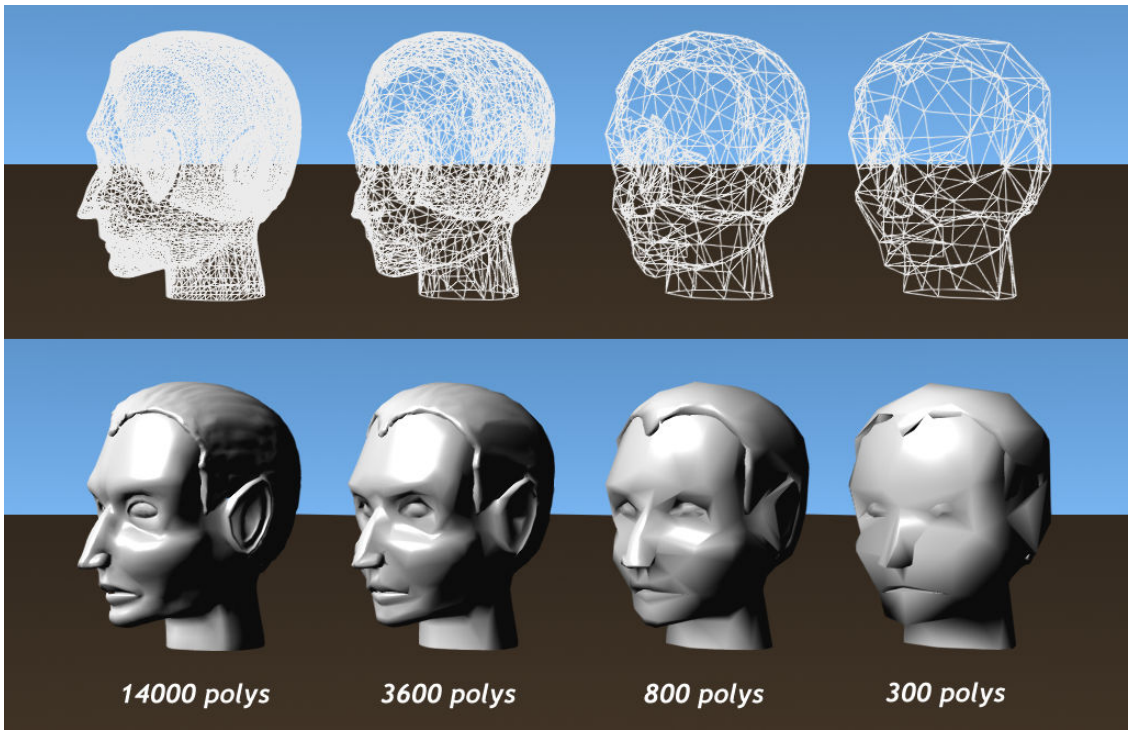

Figure 1: Image displays the different levels-of-detail of a 3D Mesh

We have agreed upon the following component breakdown and allocation:

Feature-preserving simplification - to be done by Daniel Burnham-king
Simplify mesh geometry. To be used by the mesh divider to generate the different levels of detail, approximating the original mesh with fewer surfaces.

Mesh divider - to be done by Benjamin Meier
Subdivide a model into a hierarchical level of detail [5] structure to be viewed by the renderer.

Detail restoration using normal-mapping - to be done by Benjamin Meier
Simulates additional model detail by applying a normal-map texture to the surface of the mesh. A normal-map is generated from the high-resolution version of the mesh and simulates additional mesh geometry by modifying the lighting of a low-resolution surface.

[5]Hierarchical Level of Detail (HLOD)A sequence of meshes, each one incrementally more refined than the previous. Often represented using a tree structure,where child nodes represent the same region as the parent node, but in more detail. The tree often contains bounding volume information too for fast rejection culling tests

View-dependant renderer - to be done by Justin Cossutti.
The renderer's name suggests its function. It is the piece of software that the client will directly interact with. It will need to avoid the memory and disk read speed bottlenecks that plague the software currently used by the Zamani team in order to achieve interactive frame rates.

This proposed breakdown reduces the inter-component dependencies as far as possible. While not completely independent, sub-projects can be developed in parallel if there is consensus on the layout of the intermediate file formats. These file formats will enable the components to load and save modified mesh data in a manner that the next component in the pipeline can use, without knowing how the previous component modified the data. An example of this scenario can be anticipated between the sub-meshes output by the mesh divider for the mesh simplification component to use as input.

2 Procedures and Methods

Since the goal is to produce a usable and effective software application, this project is primarily a case of method experimentation and implementation.

2.1 Feature-preserving simplification

A level-of-detail scheme will be needed to reduce the memory requirements of rendering the large mesh. Simplified versions of the mesh (and/or segments of the mesh) will be used to reduce the memory footprint of rendering the model, when lower detail is acceptable (for example, when navigating the mesh).

The algorithms will need to be tested for their ability to preserve features relevant to the architectural nature of the Zamani data, as well as their new technical limitations (on modern hardware). The meshes will need to be subdivided into manageable chunks if in-core [6] algorithms are used.

Software libraries such as the Visualization and Computer Graphics Library (VCG) and the Computational Geometry Algorithms Library (CGAL) will be used both for their built-in simplification and to aid our own implementations. This package will ultimately be used to simplify individual mesh divisions created by the mesh splitting algorithm.

[6] In-Core processing- Some models or datasets are small enough that they can beloaded fully into main memory (RAM). This allows the whole model to be processed at once.

Lower-detail versions of the model will be most apparent to users when they are interacting with the mesh. Metro [CRS98] and other similar tools will be used to measure the accuracy of the simplified meshes, potentially allowing the detection of excessive feature loss. Further research may be directed into measuring accuracy if metro fails due to the size of the dataset.

2.2 Mesh division

In order to develop a hierarchical level of detail scheme, such as an octree, a mesh must be split into multiple subsections as demonstrated by Ali Lakhia [Lak04]. The primary goal of mesh division is to be able to split a given mesh along a specific splitting plane oriented on the X, Y, or Z axis. In order to create an octree, this process is performed multiple times, recursively.

Mesh division can be relatively expensive when meshes are very large and cannot fit into memory or when vertices are unordered. This can be problematic when a triangle intersects the splitting plane as the triangle can either be split, or duplicated into the spanning cells, both of which introduce new geometry into the mesh.

The goal of this part of the project is to develop a mesh division algorithm that will work efficiently for the large meshes used in the Zamani project. We will evaluate this algorithm using two methods, a small scale deterministic test, and a large scale efficiency goal. The small scale test will determine whether the algorithm splits meshes accurately by using an existing 3D renderer, such as Blender or Meshlab, to visually inspect the resulting submeshes.

The large scale test will test the efficiency of the algorithm by measuring the elapsed time when performing mesh division on a standard mesh. Test hardware will be chosen that is close to that of the desktop computers used by the Zamani project researchers.

2.3 Normal-mapping

Simplification of a mesh can be used to reduce the amount of geometry being rendered on the screen but does have the effect of reducing the accuracy of the mesh as small details are lost when triangles are removed. To recover lost detail, a normal map texture can be applied to the geometry to simulate perturbations in surface lighting on a pixel level without requiring extra geometry. Normal mapping maps the normals of the high resolution mesh onto a low resolution mesh as a texture. Since a normal map is a texture, it requires the 3D mesh to be parameterised onto a 2D texture so that texture values can be correctly interpolated across the triangles of the mesh.

We will also be investigating whether the memory usage and performance impact of normal maps degrades performance to a point where it is better to render the original non-simplified and non-textured geometry.

In order to evaluate this component, and compare performance, we will measure the frames-per-second achieved on a standard test hardware setup.

We will also be investigating whether the memory usage and performance impact of normal maps degrades performance to a point where it is better to render the original non-simplified and non-textured geometry.

2.4 View-dependant renderer

The renderer's name suggests its function. It is the piece of software that the client will directly interact with.

The renderer will take in as input the simplified and sliced mesh output from the preprocessing stages. The preprocessing will generate a hierarchical level-of-detail (HLOD) as proposed by Lakhia [Lak04] , with child nodes of a tree containing more refined detail of the content contained in the parent node. The cells will contain hierarchical bounding volume information to perform fast frustum culling. Lakhia also found HLODs are better suited for use with VBOs to increase performance. A distance metric will be used to classify cells as near or far from the view point. Cells nearer to the viewpoint will have the higher level-of-detail versions rendered, while cells further from the viewpoint will have coarser versions rendered.

Based on an approach proposed by Lindstrom [Lin03], the rendered will make use of two threads, one for rendering and another used as the "refinement" thread. The refinement thread is responsible for loading in new, required cell data from the octree-based HLOD scheme. This increases the length of time available to the refinement thread during the rendering stage.

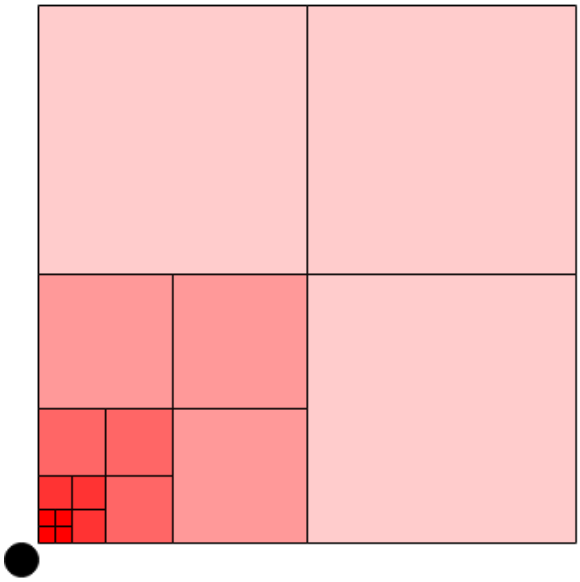

Figure 2: Image illustrates the size of nodes in the hierarchical level of detail tree. Nodes closer to the viewpoint, the black dot, are more refined, while nodes further away are larger and contain coarser detail. In this example, each square contains the same number of triangles..

While the requirement of performant[7] code to obtain interactive frame rates suggests the use of C++ and OpenGL, a first attempt will done in Java using the JOGL libraries for OpenGL. The primary reason behind this is none of the developers have much C++ experience and it is thought that Java's JIT compiler applied to good Java code will enjoy more performance than average C++ code.

The goal of the renderer is to achieve interactive frame rates when using the example models provided by the Zamani team. To set a more discrete target frame rate, the renderer should at least provide a better frame rate (FPS) than the current tool the Zamani project uses to interact with their models, Meshlab. The benchmark is as follows:

Model Name	**Faces**	**Frames per second**
Gede Palace	3 000 000	1.7
Chapel of Nossa	9 452 066	0.9

Table 1: Current performance benchmarks on an Intel Core i7 - 3630QM, 8GB RAM, nVidia GT650M

[7]Fast and efficient enough to accomplish the goal in a suitable amount of time.

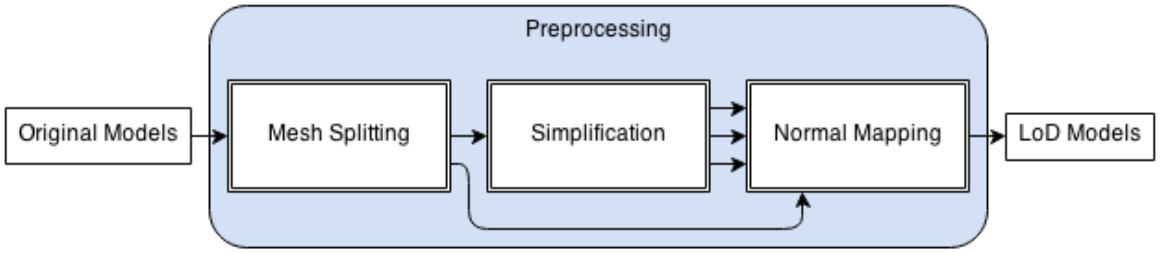

Figure 3: Image illustrates where each component fits in during the processing stage. The produced LOD models are saved in files as input for the viewer.

3 Related work

Progressive meshes
Hoppe [Hop96] proposed a solution for loss-less, continuous-representation of massive meshes called Progressive Meshes (PMs). It is a technique for morphing between meshes of increasing complexity and detail, ultimately until the original mesh, by streaming a sequence of vertex transforms. The streaming nature of the solution allows a viewing client to selectively refine the mesh in specific regions, providing the capability to display and initially low detailed mesh and refining it over time as a user alters the zoom level and navigates around the model.

This technique was considered the best alternative solution to our problem, however, due to the complexity of the algorithms for selectively refining vertices, the technique is simply too large and complex to fit within our scope and time constraints.

Point based rendering of massive meshes
Qsplat is a out-of-core [8] multi-resolution rendering system [RL00]. Qsplat focuses on rendering points instead of faces as opposed to mesh simplification and uses a hierarchical bounding sphere rendering scheme. While point-based rendering will not suit the project requirements, it does suggest some techniques for optimisation, such as saving the hierarchy in a breadth-first manner for quick, continuous file reading and only requiring pointers to the first child of a sibling set.

Surface based rendering of massive meshes

[8] Out-of-core processing - When In-Core processing is not possible, out-of-core techniques need to be used to process sections or parts of the data at a time. Out-of-core methods can often be easily parallelised.

Previous proposed solutions related to our project were focused around rendering massive meshes with the aid of some or other form of a hierarchical bounding volume. Adaptive Tetra Puzzles[CGG+04], Out-of-Core Construction and Visualization of Multi-resolution Surfaces[Lin03], Octree-based External Memory Management and Simplification of Huge Meshes[CMRS03]. However, their effective use with large architectural meshes has yet to be tested (to the best of our knowledge).

4 Anticipated outcomes

System
The complete system will be a cross platform software suite. It will be composed of two components, the preprocessor and the renderer.

The preprocessor will generate a mesh that will allow the renderer to balance performance and detail. The renderer will load sections of the preprocessed mesh depending on the level of detail required. This will allow the model to be displayed with an interactive frame rate.

Key Success Factors
The success of this project lies in a few key factors that will be determined by the client during user testing. They include:

- The levels in the level-of-detail hierarchy should maintain important visual artifacts and aspects of the heritage sites. It is important that at least the most detailed level should not eliminate artifacts such as engravings, doorways and portholes.
- Viewing of the models should be able to be done at interactive frame rates and at least be better than the performance achieved in the current software tool the Zamani team uses, Meshlab.
- The preprocessing of the models (including simplification and mesh division) should ultimately reduce the amount of administrative overhead currently incurred by the Zamani team.

Expected impact of project
If the system meets the factors of success defined above, the system will reduce a significant portion of the administrative overhead incurred by the Zamani team. They will no longer have to slice large site scans into multiple smaller meshes in order to fit into memory when viewing them. It will provide a smoother and faster viewing

of models, allowing fly-through videos for educational and tourism purposes created in a shorter period of time.

The usefulness of such a tool could also be welcomed in areas other than UCT's Geomatics Department. A lightweight viewer capable of displaying large models could be enjoyed by teams facing similar issues, such as artwork creators and architects, provided the university grants the appropriate redistribution permissions.

5 Ethical, Professional and Legal Issues

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 final project hand-in, to prevent accidental public distribution. Access to the supplied models will be provided via a password-protected FTP account from UCT's Geomatics Department. The list of authorized parties is limited to the client, ourselves and the project supervisor.

User testing will be conducted in conjunction with the client, the Zamani team, to acquire feedback on the usability of the system and whether it meets their requirements . As no public testing will be done, we will not require ethical clearance from the university.

As such a set of tools could be found useful in other areas for application, permission for redistribution would have to be acquired from the university.

6 Project Plan

6.1 Deliverables

The main goal of the project is to deliver a set of tools that will allow the researchers from the Zamani group to interactively display large models. The set of tools will be composed of three individually developed components:

- **Mesh division and normal mapping tool.**

  This software tool will be used to recursively divide the models into cubes that can be simplified to allow for different levels of detail to be loaded by the renderer in real time.

Initial prototype for feasibility demonstration due: 28 July 2014

Final implementation due: 25 September 2014

- **Mesh Simplifier.**

  This software tool will be used to simplify the blocks created by the mesh divisor. Its input will be a .ply file which will contain either an entire mesh or a subset (chunk) of a larger mesh. The tool will be written in C++ to allow for the use of existing geometry processing libraries (VCGlib, CGAL and Boost).

  Initial prototype for feasibility demonstration due: 28 July 2014

  Final implementation due: 25 September 2014

- **Viewer (renderer)**

  The viewer will render the large mesh by loading segments (divisions of the mesh) containing different levels of detail. Allowing the renderer switch between high performance and high detail.

  Initial prototype for feasibility demonstration due: 28 July 2014

  Final implementation due: 25 September 2014

- **Project Poster**

  Due: 3 October 2014

- **Individual Project reports**

  Each team member will produce a project report, detailing previous works in the field, design of the system, implementation, testing and results.

  Due: 29 October 2014

- **Project Web Page**

  A website detailing the different components of the project.

  Due: 12 November 2014

6.2 Resources Required

1. **Hardware**
   We require access to one or more workstations equipped with suitable hardware. Due to the large memory requirements of massive meshes, the workstations will need to be equipped with at least 8 GB of RAM and at least 200GB of hard drive space. A modern quad core CPU such as an Intel i5 or suitable AMD

alternative is recommended. The viewer will require a dedicated Graphics Card but since the software is designed to run on consumer hardware a modern card such as the Geforce GTX 560 Ti, currently available in the honours lab, is suitable.

2. **Software**
   - Windows 7/8 operating system
   - Meshlab
   - C++ and Java development environments.

3. **Human Resources**
   Our supervisor, Patrick Marais, is an experienced researcher in the field of computer graphics. Patrick will provide guidance and supervision during development of the project.

   Professor Heinz Rüther of the UCT Geomatics department and the Zamani project is the client of the product. He and his team will provide input to guide the project and answer any questions we might have.

4. **Data**
   The Zamani Project has given us access to sample models of 67 MB and 207 MB and will provide original meshes when required as the project progresses.

6.3 Risks and Mitigation

Risk	Likelihood	Impact	Mitigation
Dependency between components.	5	6	Regular communication between the parties of the related works needed to stay on the right course.
Implementation complexity too high.	7	6	Sample lots of related work and seek review and advice from supervisor. Try use simple data structures wherever possible.
Project fails to perform at an interactive frame-rate.	6	4	Introduce more levels-of-detail to reduce the amount of time spent rendering the most detailed versions of the models.
Source code becomes unwieldy with multiple develops on the same codebase.	2	6	Use a version control system such as Git and branch out different components.

Table 2: Risks and Mitigation Table

6.4 Milestones

Milestone	**Date**
Literature review	15 May 2014
Project (Research) proposal + Project Plan	26 May 2014
Project Proposal Presentation	30 May 2014
Feasibility Demonstration	28 July 2014
Basic Web Presence	17 June 2014
Background/Theory Chapter	29 July 2014
Design Chapter	27 August 2014
First Prototype, Experiments, Performance Test Chapters	17 September 2014
Final Prototype, Experiments, Performance Test Chapters	26 September 2014
Chapters on Implementation and Testing. Software Complete.	1 October 2014
Draft of Report	22 October 2014
Final Report	29 October 2014
Poster	3 November 2014
Web presence up to date (complete)	5 November 2014
Reflection paper	9 November 2014

Table 3: Table of Project Milestones

6.5 Gantt chart

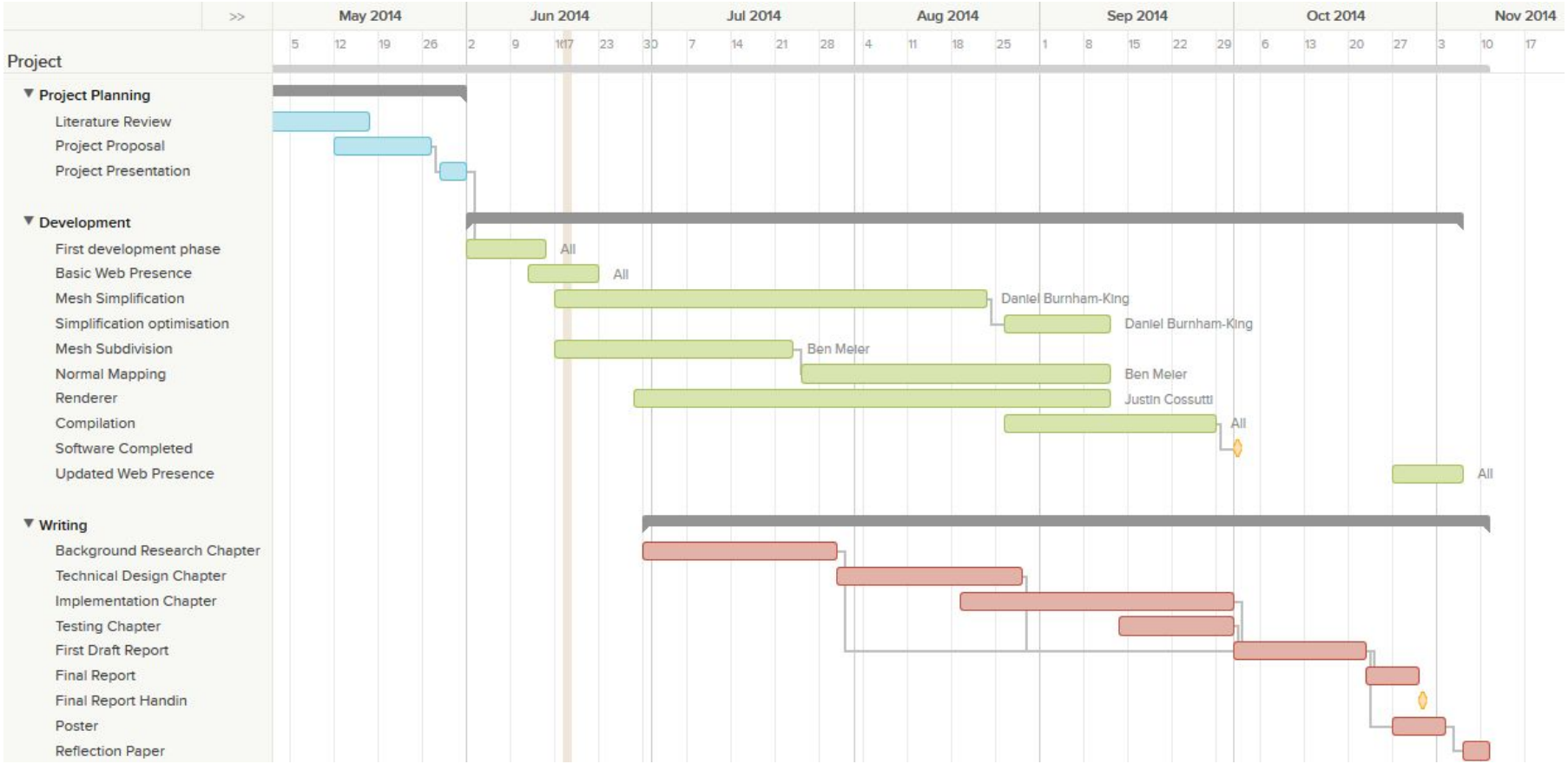

Figure 4: Gannt chart showing project time line and major hand-in's.

List of Figures

List of Tables

References

[CGG+04] Paolo Cignoni, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, Federico Ponchio, and Roberto Scopigno. Adaptive tetrapuzzles: Efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. *ACM Trans. Graph.*, 23(3):796–803, August 2004.

[CMRS03] Paolo Cignoni, Claudio Montani, Claudio Rocchini, and Roberto Scopigno. External memory management and simplification of huge meshes. *IEEE Trans. Vis. Comput. Graph.*, 9(4):525–537, 2003.

[CRS98] Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno. Metro: Measuring error on simplified surfaces. *Comput. Graph. Forum*, 17(2):167–174, 1998.

[Hop96] Hugues Hoppe. Progressive meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 99–108, New York, NY, USA, 1996. ACM.

[Lak04] Ali Lakhia. Efficient interactive rendering of detailed models with hierarchical levels of detail. In *3DPVT*, pages 275–282. IEEE Computer Society, 2004.

[Lin03] Peter Lindstrom. Out-of-core construction and visualization of multiresolution surfaces. In Michael Zyda, Michael V. Capps, Randy F. Pausch, and Gary Bishop, editors, *SI3D*, pages 93–102. ACM, 2003.

[RL00] Szymon Rusinkiewicz and Marc Levoy. Qsplat: A multiresolution point rendering system for large meshes. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 343–352, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.