Performance Impact of Java3D Binary Serialized Files on Multiresolution Terrains

Master Thesis

Morten Gustavsen

March 13, 2009
Halden, Norway

Østfold University College

Abstract

This thesis describes an exploratory study that measures the overall performance impact on a textured and non-textured multiresolution terrain rendering system when using pre-processed Java3D binary files. Much of the work has focussed on studying fundamental concepts and algorithms for processing large virtual terrains. Based on the output from this work, a software testbed capable of collecting quantitative measures from the rendering output was implemented. The testbed was implemented in order to be able to study, and understand, the performance impact of Java3D binary serialized files by looking at the entirety of a virtual terrain rendering system.

The testbed implements a novel approach to LOD control using the high level graphics API Java3D, and we will argue that this approach is viable for generating large virtual terrains.

The performance comparisons are made by running a set of tests that simulate typical user operations performed in virtual terrain software. Based on the output from the testing, the thesis gives guidance on when it is most appropriate to use Java3D binary serialized files.

Keywords: Multiresolution Virtual Terrains, Java, Java3D, Serialization

Acknowledgements

I wish to thank my thesis supervisors Gunnar Misund at Østfold University College and Grete Rindahl at the Institute for Energy Technology (IFE), they have shown patience and interest in my studies beyond what could be expected.

I am also grateful to Michael Louka for the technical discussions and the proof reading of this thesis.

I also wish to thank my employer IFE which gave me the necessary time and flexibility to work with my thesis during the work hours.

Table of Contents

Chapter 1

Introduction

The use of maps dates back to the Stone Age, and appears to exist several millenniums earlier than written language. Even though cartography has developed significantly from hand-drawings to extensive use of computers today, old maps are surprisingly similar to the maps we use today.

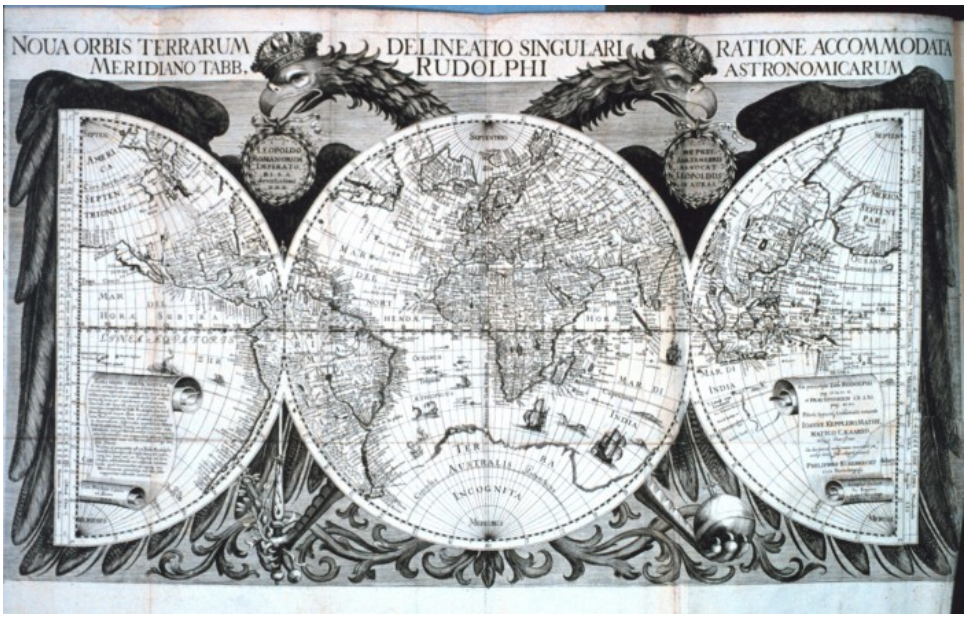


Figure 1.1: A map of the world from Tabulae Rudolphinae 1627 by Johannes Kepler

While we tend to think of maps as reduced, simplified 2D images of the world, the rapid development of computer hardware has made it possible to draw large 3D representations of terrains on a normal personal computer. In the 1980s and 1990s such visualisation would have required a

mainframe or supercomputer. Technology and especially the more widespread use of computers, have not only changed the way we create maps, but has also changed the way we view and work with maps.

Even though researchers, and the geographic information system (GIS) business, has worked with virtual terrain visualisation since the late seventies and still do, the interest for geo-data systems amongst the general population has not increased before the latter years. The increased interest is popularised by Virtual Globe applications such as Google Earth[1] and NASA Worldwind[2]. Virtual Globes are 3D spherical representations of the Earth, in which a user can move freely around.

The virtual terrain project[3] lists some common usages and hypes concerning virtual globes. Some of the interesting areas they describe includes virtual tourism, education, urban planning, visualisation of weather and video games.

Figure 1.2: A map of the world in 2009 using the Virtual Globe application - Google Earth

Virtual Globe applications have not only resulted in a renewed interest in research in the area,

[1] http://earth.google.com/
[2] http://worldwind.arc.nasa.gov/
[3] http://www.vterrain.org

but also in the availability of public-access terrain databases.

1.1 Motivation

In virtual terrain or virtual globe software, a large number of activities take place behind the scenes; downloading files from servers, reading and parsing files, creating geometry, building data structures, paging in and out of memory, mapping textures and so on. In order to optimise and reduce the risk of re-processing the same data several times, some of these activities store information to disc when the process is finished. For instance, the Nasa Worldwind software is storing the image and height data downloaded via the internet to local disc, to avoid doing this time-consuming process several times. It is more optimal to have them availiable for rapid access on local disc.

The rapid advance of computer hardware opens up for possibilites of using less specialised and optimised methods for building virtual terrains, and yet having a high rendering throughput. Using a more high-level API, such as Java3D[4], can provide abstractions from the low level details, and can facilitate and speed up the development process speed by making more complex programming simpler.

Java3D has the ability to store parts or entire virtual universes as binary objects on disc. This opens up for the possibility to store completely processed geometry to disc. One could assume that by storing completely processed geometry to disc, and by having the ability to avoid the entire recalculation process by rapidly reading the already processed geometry directly into software will have an effect on performance.

This work will also be an contribution to an already existing software platform developed at the Institute for Energy Technology (IFE)[5] that is capable of scenario management and simulation combined with hazard visualisation. A number of interesting applications are made possible by the combination of geo-data and terrain handling, especially in combination with scenario management and simulation functionality.

If we consider the area around a nuclear site, a wide-area model can i.e be used to plan new construction and other operations around the existing plant, checking access routes for personnel and vehicles, or in extreme cases, like Chernobyl, to aid in the site clean-up and remediation. In particular, emergency plans can be visualised and simulated, and used to brief emergency services,

[4] http://java.sun.com/javase/technologies/desktop/java3d/
[5] www.ife.no

such as fire fighters (e.g. [10]) and other personnel that are not as familiar with the plant than the permanent staff. Simulation features for planning [14][12] and training [23] [10] [24] could include risk visualisation, contamination and radiation distribution (e.g. in an accident scenario), fire and smoke, evacuation of personnel, and so forth. For planning, training, or even for emergency support, 3D software could be used to rapidly identify optimal routes for vehicles and personnel to achieve specific goals.

The ability to combine techniques demonstrated by virtual terrain software, with the VR simulation and scenario planning [12], collaboration [2], radiation visualisation [9] [13], and other capabilities, e.g. [11] [10] in the software platform at IFE, will enable us to create more dynamic simulations than is possible using tools like Google Earth directly.

1.2 Research Objectives

The work in this thesis focusses on measuring the overall software performance on a textured and non-textured multi-resolution terrain rendering system when using serialised Java3D binary files compared to more traditional file-formats. In this context the traditional file-formats are the formats that are available from the Nasa Worldwind Server.

The software performance will be measured by looking at quantitative measures from the rendering output. In order to achieve this, a software testbed of a virtual terrain system has been implemented using Java[6] and Java3D[7]. The thesis also gives a description on how we can build a virtual terrain rendering system by undertaking a novel approach using the high-level Java and Java3D API (Application Programmers Interface).

The design and implementation of the developed testbed has favored simplicity, and have utilised many of the inbuilt high-level mechanisms in the APIs in order to build a complete virtual terrain rendering system. The testbed is using satellite imagery from a Nasa Worldwind geodetic server, and is coupled with parts of the Nasa Worldwind Java implementation in order to retrieve the data necessary to build realistic virtual terrains. The software has been implemented with the capability to store the rendered virtual terrain to serialised Java3D binary formats and reading them again.

Through research and prototyping the following research question will be focussed on, In visualisation of a simulated real world with virtual terrains.

- What is the overall impact on performance on multiresolution virtual terrains when utilising

[6] http://java.sun.com/

[7] http://java.sun.com/javase/technologies/desktop/java3d/

Java3D serialised binary files compared to utilising files from a Nasa Worldwind Server?

- Will serialising different portions and levels of a Java3D scenegraph have impact on performance?
- What differences will serialising a textured terrain compared to a non-textured terrain have on performance?

The results and findings are to be backed up with the following deliverables:

- A software testbed capable of rendering virtual terrains using Java3D with automated test data collection.
- The results from the automated test data collection.

A secondary objective will be to see if a novel approach, using a high level graphics API, can be undertaken in order to build virtual terrain rendering software with high rendering throughput.

1.3 Method

1.3.1 Literature and Domain Study

In the initiation and preparation phase of this work, the most fundamental concepts and algorithms for generating large terrain models were studied, and in some cases tested. A brief description on virtual terrain rendering techniques, and how some systems have successfully utilised them are described in Chapter 2. The chapter also describes how we can go from a limited data set, to a distributed technique that in theory can visualise gigabytes of data.

1.3.2 Testbed design and implementation

The design and implementation of the testbed is presented in Chapter 3 and 4, and is based on the concepts outlined in Chapter 2. The design has favoured simplicity, and is utilising many inbuilt features of the high-level graphics API Java3D.

A virtual terrain system consists of many different parts that need to work together in order to create a high rendering throughput. To predict the consequences of using different file-formats as the basis for building a terrain is not a trivial task. The work with this thesis has for this reason prioritised to build a working testbed of a virtual terrain rendering system in order to study the

overall software performance effects of different file input formats. By having many of the vital parts in place, one can extract more data from the tests and hence get a better understanding of the eventual advantages or bottlenecks for the different file-formats.

Methodology

The implementation of the testbed is considered to be a research effort, and will need a software methodology that is flexible and can handle late changes during the implementation. This leaves out predictive methodologies like i.e. the Waterfall methodology, and opens up for Agile development methodologies.

Many consider Agile methodologies as a modern practice but its applications dates back to the mid-1950s [19]. There are a number of agile software development methods, but all of them have some common denominators and are written down in the agile manifesto.[8] The following points is taken from this manifesto, and is the most important ones for this project.

1. Software is developed using many iterations, and not only one.
2. Emphasise realtime communication, preferably face-to-face communication over written documentation
3. Working software is the measure for success.

The biggest difference from predictive methodologies is the amount of time used on a iteration. It emphasises that one time period is measured in weeks and not months, this also fits the scope of this project better. Each iteration is handled as a small project, hence having it s own life cycle containing analysis, design, implementation and testing. There is still a need for analysis, design, implementation and testing activities, but these activities are carried out in a more flexible way than in the waterfall process.

An agile software software methodology fits the requirements of a research software development project, because we are not able to predict and document all requirements before the implementation period starts.

There are a number of agile development methods and perhaps the most popular is Extreme Programming[9], I will use a method called Iterative and Incremental programming, which many of the other agile methodologies is based on.

[8] http://www.agilemanifesto.org/
[9] www.extremeprogramming.org

1.3.3 Data collection

The test data is collected either by the implemented testbed or by third party applications. The implemented testbed have inbuilt mechanisms for counting the number of tiles rendered in a scene, the number of attempts to render tiles and the accumulated time to render a tile. This data, collected from the actual rendered data, is used to evaluate the file-formats ability to deliver high resolution data. By having quantitative measures collected from the rendering output, one has a better basis for discussion, because the data has been through all the vital parts of a virtual terrain rendering system. The data is collected when running simulated user test from predefined flight-paths that simulates normal user operations in order to approach the problem as realistically as possible.

1.3.4 Data Analysis

Human modellers are often using visual criteria to decide the visual quality of a 3D-model. This is in many ways an excellent way of evaluating the effectiveness of a 3D-models, because it uses the human visual system. However, in many applications this method is not desirable because it is too work-intensive, and can not be measured in a quantitative way.

In response to this, there are implemented several methods for automatic quantitative measures for the quality of virtual terrain models. These are often referred to as *error measurement* . Error measurement is important in order to know the quality of the results, and to be able to compare them to other algorithms. There are published several algorithms concerning error measuring, and is by many considered to be a work field in its own. The LOD book [17] gives a good introduction to the field, and is explaining the key elements of the error metrics and explains some of the published algorithms. This approach is considered to be out of scope for this study, and we will undertake an approach where we are looking at the entirety of a virtual terrain application.

Many applications uses the Frame Per Second (FPS) tests in order to measure the performance in 3D applications. In this application it would not be fair to use this as primary test data input for discussion. Due to the unknown fact that different file-formats have the ability to deliver the equal amount of rendered data during the simulated tests, one would not be sure if the the comparisons would be reliable due to the different amount of rendered data. The amount of rendered data can have a huge impact on rendering performance. For this reason, the fps tests are used as a measure to tell if a test has passed or fail, and to give indications concerning the rendering performance.

1.4 Scope

The implemented software is intended to be used as a testbed to answer the research objectives, and will not be tested as thoroughly as a commercial software product. Important aspects in the field of virtual terrain rendering such as spherical textures, cracks and t-junctions between adjacent tiles and supporting of different data-sources and formats is not implemented nor considered. This is both due to a limited timeframe for the project, and the fact that it is not considered important in order to answer the research objective.

The performance investigation and findings are undertaken by an exploratory study, rather than using formal statistical methods.

1.5 Outline

Background

Chapter 2 gives a brief introduction to the most fundamental concepts when generating large terrain models, and describes the most commonly used optimalisation techniques for terrain rendering. It also describes the most popular virtual globe systems used today, and presents some of it s advantages and disadvantages. Having this in mind, it describes the motivation of implementing this thesis.

Testbed Design

Chapter 3 gives a detailed explanation of the design and the development as well as the design process. It uses pseudo-code, figures, UML (Unified Modeling Language) sequence and class diagrams to support the explanation of the design.

Testbed Implementation

Chapter 4 gives a detailed description of the motivation and implementation of the design described in Chapter 3.

Performance Investigation

Chapter 5 explains the designing of the tests used to answer the research objective together with the test results.

Discussion and clarifying tests

Chapter 6 presents the findings from the execution of the tests defined in Chapter 5, and performs further testing to clarify the indications from the performance investigation.

Conclusion and Future Research

Chapter 7 Discusses the suitability and the viability of using Java3D binary as file format for virtual terrains. Finally some recommendations for further research are offered.

Chapter 2

Background

From the last quarter of the 20th century it have been developed several methods for visualising large terrains on a computer. Visualising large terrain models requires powerful graphics hardware and even high-end modern graphical cards are not capable of generating a high resolution virtual terrain covering a large geographical area. To visualise 3D-models as large as the entire planet we need techniques to be able to render data-sets that is larger than the available main memory on the computer we are running on, or that can be handled in real-time by the 3D graphics hardware. This capability is referred to as operating out-of-core. To give an example of the amount of information required to render the entire planet in 1-cm resolution, the color imagery alone requires 1 Petabyte of storage [5].

2.1 Related Applications

Many tools have been developed to solve the problems of discovering, modeling, and visualising global geospatial data and associated data. In the section below a number of related applications are oulined shortly, however it should be noted that the applications that had the biggest impact on the implemented testbed is Google Earth and Nasa Worldwind.

2.1.1 Google Earth

Google Earth is a well known virtual globe geospatial browser available for Macintosh, Windows, and Linux. Google Earth is commercial product and is available in different versions at different

cost. Google is also distributing a free version which has a limited feature set and slower performance compared to the non-free versions[1]. The free version is licensed for home/personal use, and can not be used in a work environment. In a work setting the Pro version that costs 400 dollars per year[2] is necessary.

The release of Google Earth in mid 2006 increased the number of references to virtual globes in online news media between 2006 and 2007 with about ten times[3]. Based on a lexical analysis of crawled Web data, Google Earth had about 83 percent of this coverage, so it is fair to say that Google Earth has been the primary driver behind the observable increase in popularity of geospatial platforms.

One of the key factors for Google Earth s success is that they have opened up some of their API s for public access. This has resulted in several interesting applications and mash-ups, which again has resulted in even more interest in Google Earth.

2.1.2 SINTEF - Virtual Globe

Virtual Globe [4] is an Internet based client-server application for visualisation, out of core, global scale terrain models. From the homepage, many different versions can be downloaded, and the application is capable of visualising several different datasets, such as SRTM[5], SRTM 30+, GLOBE [6]and GTOP030 [7], as well as using VRML[8] models as additional 3D content.

The application uses highly optimized datastructures both on the client and the server-side in order to achieve a high rendering throughput. The mutual computations increases the complexity and computational cost of the software. The main bottleneck seems to be the client, and the authors [3] suggests to replace the Java platform with C++, due to the limitations of optimising the algorithm furher, especially regarding memory handling.

In 2006 the Globe project was obtained by Norkart[9] for further development.

[1] http://earth.google.com/enterprise/earth_pro.html
[2] http://www.bullsworld.net/2006/01/18/google-earth-is-not-free/
[3] http://www.geospatialweb.com/figure-4
[4] http://www.virtual-globe.info
[5] http://www2.jpl.nasa.gov/srtm/
[6] http://www.ngdc.noaa.gov/mgg/topo/globe.html
[7] http://edc.usgs.gov/products/elevation/gtopo30/gtopo30.html
[8] http://www.web3d.org/x3d/specifications/vrml/VRML1.0/index.html
[9] http://www.norkart.no/

2.1.3 Microsoft Virtual Earth

Microsoft Virtual Earth 3D is built as an extension to Internet Microsoft Internet Explorer or Mozilla Firefox, and enables the end user to view a Virtual Globe within any of the two Internet browsers. It is a 3D user interface for Live Search Maps that can only be run on the Windows platform. The advantage of using a software extension, is that the user does not need to download an entire application, but can use the Virtual Globe inside an already installed browser.

Microsoft Virtual Earth has an extensive, free Software Development Kit(SDK) which enables developers to integrate their own content or modifying the user interface or interaction.

2.1.4 Nasa Worldwind

Nasa Worldwind is another popular virtual globe geospatial browser, and differs from Google Earth in that it is open source and only available for Windows. A free, cross-platform Java version is under development, and was planned to be released in spring 2007, but a the beginning of 2009 it is still in version 0.5.0.

Nasa Worldwind differs from Google Earth in philosophy as all parts of the software are distributed for free. While Google Earth has copyrighted all maps under United States Copyright law, one may freely modify, re-distribute and use maps used in Nasa Worldwind . NASA have collected their own geospatial data from satellites, and are distributing terabytes of this geodetic-data for free. They are distributing the most complete high-resolution digital topographic database of Earth, covering approx. 80 percent of the world. The satellite imagery used for this testbed, is fetched from a Nasa Worldwind server, but is however restricted to Landsat7[10] imagery set, which is the most high resolution dataset freely available today.

2.1.5 Other important technologies and applications

On the sections above, one could also list GeoVRML and Terravision[11], but these are handled separately in own sections later in this chapter.

[10] http://landsat.gsfc.nasa.gov/
[11] http://www.ai.sri.com/tvgeo/

2.2 Terrain Rendering Simpli cation Concepts

Because of the amount of data needed to render a large virtual terrain, or even the entire globe, it would be an impossible task to build this off-line or by hand. Virtual Terrains are therefore often built up by satellite imagery combined with DEMs (Digital Elevation Model). A DEM is a high-quality picture that is used to represent the earth s ground surface topology, and is often used to retrieve the elevation of the ground. By combining information from these images, we have enough information to build a virtual terrain from a mesh of triangles or quads.

A DEM can be represented as a Regular Grid (a grid of squares) or as a triangular irregular network (TIN).

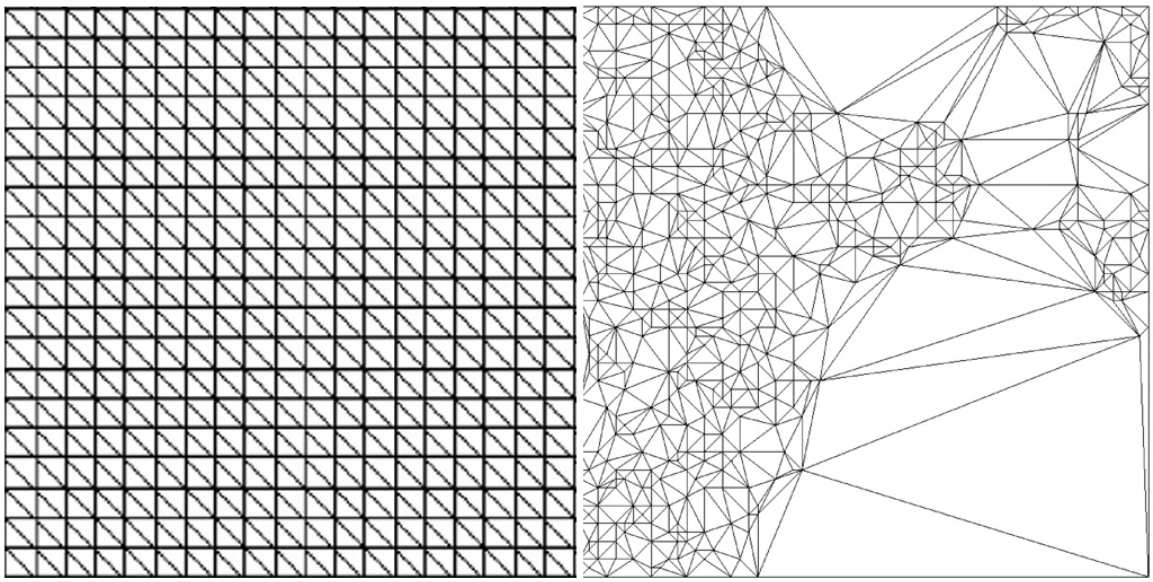


Figure 2.1: To the left we can see a regular grid, to the right we see a Tin representation

As we can see from Figure 2.1 from [25], regular grids are usually less optimal because the method uses the same resolution for the entire region. However, the advantage of this is that regular grids are simpler to map to datasets, as the mesh is represented in the same way as the data. This makes it easier to implement and to create real-time solutions for generating these meshes, as we do not need as much computer power when generating geometry. They are also easier to divide into smaller pieces because of the inherent regularity.

The regular grid makes it easier to implement features like terrain following and collision detection because we can easily retrieve the height value directly from the dataset. By evaluating the

height value we can adjust the viewpoint based on this value.

However, TINs are more flexible and can result in fewer polygons because the method uses fewer polygons for flat terrain areas, while using higher resolution height fields for areas that require more detail. The difficulty with TINs is that they require more processing because the meshes are represented very differently from the typical dataset. This make TINs less suitable for real-time and online solutions. They also require a more complex dataset, as they require x, y and z data during mesh creation.

2.3 Level of Detail(LOD)

The first applications to visualize and use terrain data was flight-simulators and Geographic Information Systems (GIS). Because of the large amount of data and that was needed to render a large virtual terrain, Schachter [27] discusses the necessity of decreasing the number of polygons in the scene, and states that a common way of solving this problem is to render objects with lower resolution as they appear far way. He is here describing main purpose behind a LOD, simplify complex objects to achieve better rendering performance.

2.3.1 Discrete LOD

The main concept behind LOD is here basically summed up in the Figure 2.2. Use a less detailed model for small, distant or unimportant parts of the scene when rendering [7]. With the use of a LOD we typically have several versions of the same object in the scene, each duplicate version with less polygons than the previous, and hence quicker to render on screen. With an example like the one illustrated, we create the different versions of 3D model by hand, and utilise a static or discrete LOD technique. With a discrete LOD, we choose which version we will render based on the distance from the viewpoint to the model. When viewing an object from a far distance, we will choose a less detailed version, and switch to more detailed ones when we navigate closer to the model. These types can be created offline at fixed resolutions. One of the advantages with discrete LODs is that they are easy to implement, in fact many modern 3D Application Programmer Interface(API) has this mechanism already implemented and ready to use. Another advantage with a discrete LOD, is that they are very cost effective for the run time system. It only has to pick the LOD, no continuous calculation is necessary.

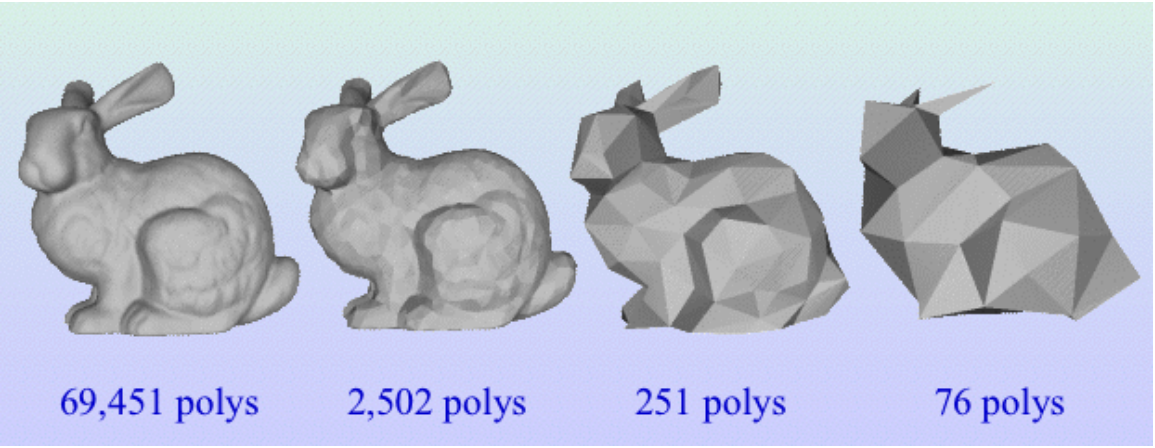


Figure 2.2: Traditional LOD In a nutshell, taken from D. Luebke, Introduction Presentation for course, Level Of Detail Management for Games

2.3.2 Hierarchical LOD

To perform simplification on a virtual terrain, we must be able to represent parts of the grid at different resolutions. Lindstrom created techniques that enabled us to add extra detail gradually at runtime rather than choosing between predetermined discrete detail levels. This technique was a huge improvement, and made it possible to view larger terrains at a higher resolution. Lindstrom used a hierarchical datastructure for his algorithm. Most of the more recent algorithms developed for virtual terrains have adopted a similar concept, with hierarchical data structures, but supplemented it with other enhancements.

With a hierarchical LOD structure, we can represent different parts of a grid at different resolutions. In that way a large object is represented from subdivided, gradually refined smaller objects, this is shown in 2.3. The structure is reffered to as a multiresolution representation, this results in better scalability for large models, because we can exclude parts of the scene that cannot be seen, or use larger polygons at greater distances. This is visually illustrated in figure 2.4, taken from [6]. These meshes are referred to as continuous level of detail meshes or LOD meshes. This approach is considered the most suitable for terrain applications [20] [18], however it is not without problems.

This LOD technique is optimised for traversing a virtual terrain. Because the optimalisations are based on reducing the detail of distant parts of the scene, we may have multiple resolutions of a relatively large area when viewing a terrain from above.

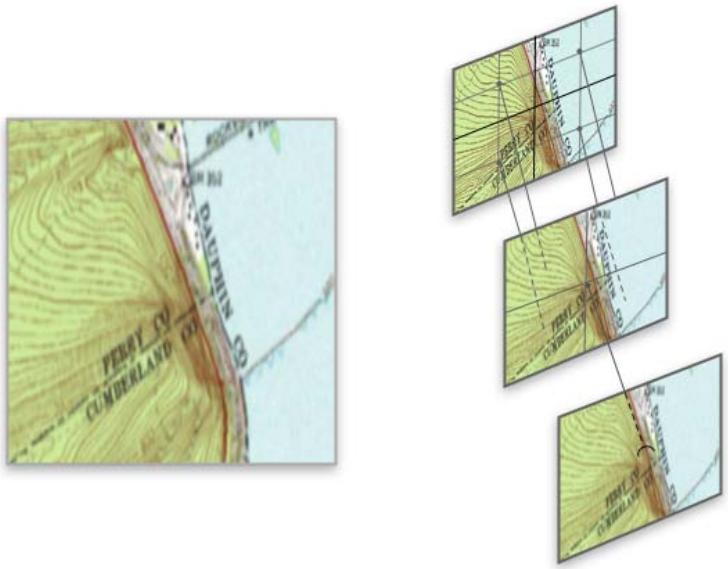


Figure 2.3: Hierarchical LOD structure

2.3.3 Quadtree

A quadtree is a tree data structure where each internal node has up to four children. When used to implement terrain LODs, a rectangular area is typically divided into four uniform parts, and those parts are then recursively divided into to sub-parts until the desired resolution is achieved. The term quadtree refers to the representation and not the geometric primitive, so triangles can still be used as primitives.

Quadtrees are simple and efficient, and are therefore appropriate for rendering terrain data that exceeds the size of the available main memory on the computer we are running on. This capability is referred to as operating out-of-core. To achieve this, we can load datasets dynamically and page terrain in and out of the main memory. While most terrain algorithms page terrain data from a local disc, a desirable feature of a terrain rendering system is loading data over a network such as the Internet. Due to the typical lag when loading data over a network connection, when compared to loading from local persistent storage, we are dependent on rapid processing on the client so that the user does not experience lag. Because of the simplicity of the data representation, quadtrees are preferred over other tree structures for out of core operations, as they do not require much processing on the client and are therefore relatively insensitive to network lag.

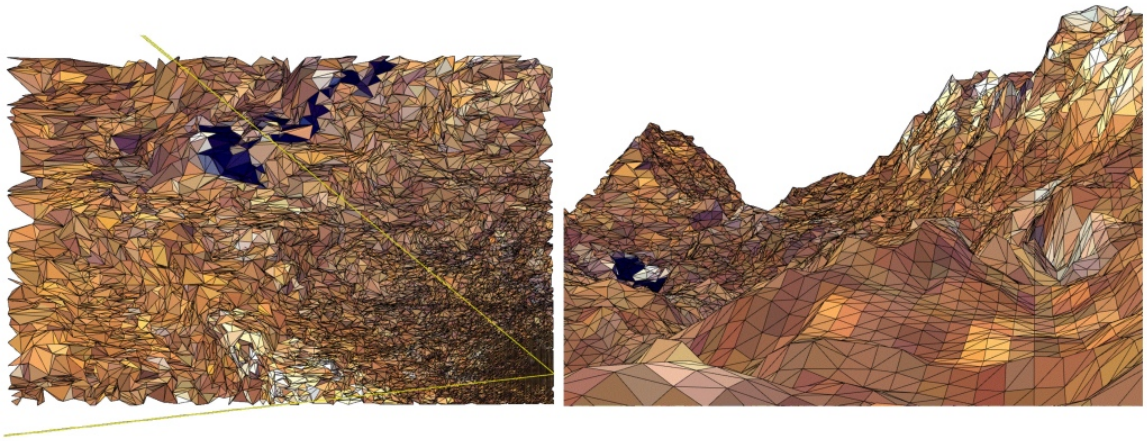


Figure 2.4: A Quadtree representation

2.4 From local to distributed data - TerraVision

TerraVision is an open source visualisation system which was developed in response to the limitiations from other software that can only browse terrain data from local persistent storage. The amount of data needed to represent the entire globe is to large to store on persistent storage on one single machine, it was necessary to load datasets distributed across multiple servers on the web. In addition to the ability to visualise huge datasets at an impressive framerate, Terravision has an extensive feature set,

TerraVision is utilising a quadtree structure, where they split large data into a grid of 128 x 128 pixels. In addition images at different resolutions are split into power of 2 sizes (e.g. 2048x2048, 1024x1024, 512x512, 128x128). By using power of 2 images, we can reduce the amount of memory used on the graphics card heavily. By having the images at different resolution represented in a hierarchy, makes it possible to directly access any part of the image at any desired resolution, in addition to the ability to only streaming the parts of the images they need for the current view.

Another advantage with a distributed approach, is that it enables the distribution of load on a machine when fetching a data tile, to several machines. This can result in better performance if many users are getting data from the same area. Having distributed sets is also facilitating the updating of data, because it enables for local or national updates of parts of the terrain data. It is usually easier for Norwegians to update data from Norway, than someone in the USA.

This software and the vision that everyone should have access, and be able to interactively browsing the entire earth, lead up to the GeoVRML standard, which is an added as an amendment

to the VRML97 specification.

2.5 VRML

VRML is a standard text file format used to represent 3D worlds, designed specifically for the use on Internet. The first version of VRML was designed in November 1994, and resembled the API and file format of the Open Inventor software component designed by SGI. The current and functionally complete version is called VRML97 and has become an ISO standard.

Although VRML is often used as an geometry interchange format for 3D models, it is also possible to create interactive 3D content. Animations, sounds, lighting, and other aspects of the virtual world can interact with the user or may be triggered by external events such as timers. A special Script Node allows to add program code to a VRML file. This code is often written in Java or JavaScript (ECMAScript).

2.5.1 Using VRML97 for terrains

One could imagine using standard VRML97 for terrain rendering, but we would face some challenges. M. Reddy is presenting all these problems with solutions in his publication Modelling the Digital Earth in VRML[19]. The biggest challenges he mentions are:

- Lack of precision; VRML97 have only floating points support making it possible to model the entire earth with better precision than one meter
- VRML97 loads entire models during startup, visualising huge models would require loading tiles dynamically from different locations and organise them in a multiresolution terrain
- The default navigation in VRML does not consider the viewpoints position above the terrain. We need faster movement when viewing the model 10.000km from above, compared to 100m
- VRML97 uses a right-handed Cartesian coordinate system, most georeferenced data are in or geodetic (geographic) coordinate system.

Martin Reddy s work on these issues led up to the GeoVRML extension to VRML.

2.6 GeoVRML

GeoVRML is an extension to VRML, and is used as a building block on top of VRML. This means that GeoVRML has to be used in combination with VRML and cannot operate on its own. GeoVRML is suffering from one of the same problems as VRML. X3D have included it in the specification as one of its extensions to VRML. This has resulted in that GeoVRML is not widely supported by VRML renderers, because there was anticipation that X3D would take over for VRML.

Morten Granlund [15] have tested eight different VRML plugins to check for GeoVRML support. The plugins he tested was Blaxxun Contact 5.104, BS Contact, Cortona VRML Client 4.2, Cosmo Player, Flux, Octagon, Free Player, Open Worlds and Vcom 3D Venues. Unfortunately he reported that the only plugin that supports GeoVRML is the Cortona client. He also reports that this plugin loads an entire model into memory before displaying it, making it unsuitable for viewing large 3d models.

The latest implemented version is 1.1, which was finished in 2002. There has not been any developments on the GeoVRML standard or implementation since the 1.1 version, but it should be mentioned that a 2.0 version is listed on the official GeoVRML site, but that is more a wishlist than an an actual specification.

The work of the GeoVRML consortium has been taken over by X3D-earth working group, which shares many of the same visions and goals as the GeoVRML working group.

2.6.1 GeoLOD

The node that controls the terrain resolution in GeoVRML is called GeoLOD. This LOD algorithm is based on Martin Reddy s ambitious work on TerraVision. TerraVision had an ultimate goal of visualising huge datasets real time, in the order of terabytes. This amount of data must be distributed, making it necessary for GeoVRML to load datasets from multiple servers across the web. The GeoLOD node can specify inline nodes, which it can load runtime. The main focus on distributed datasets is making GeoVRML differ from the other algorithms described.

The GeoLOD node is a quadtree LOD where up to four different scenes are dynamically loaded. Images or elevation bitmaps are recursively divided to produce a multiresolution pyramid. As we can see from Figure 2.5, each level of this pyramid is divided into a grid with equally sized regular grids e.g. 128x128 pixels. This is done to reduce the amount of texture memory used. A tile in the pyramid maps directly onto 4 tiles on the higher resolution level. In this way we can display areas around the viewpoint at a higher resolution whilst other not so interesting areas will be displayed at

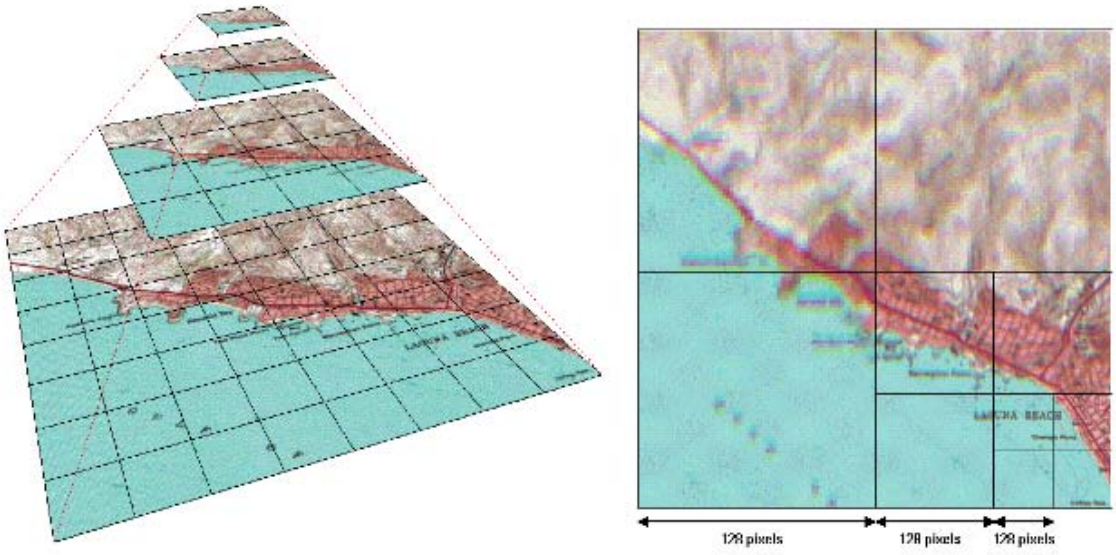


Figure 2.5: Image illustrating GeoLOD pyramid representation. The right picture shows how we can have different resolutions in a model at different regions (from [4])

a higher resolution.

Using a discrete LOD to select different levels in the LOD, is very difficult, and may produce visual popping when switching from one resolution level to another. Selecting the optimal distance is difficult as it may vary due to terrain complexity or the users field of view. The specification does not say anything on how we should calculate this value.

The specification does not either specify any mechanism to smooth LOD between adjacent tiles, producing a high risk to create cracks in the terrain. In fact it does not even specify if we should use triangles or quads as geometry in our mesh. The other algorithms described are much more precise concerning these issues.

2.6.2 Elevation and GeoElevationGrid

The GeoLod node is often used together with GeoElevationGrids. VRML97 provides an Elevation Grid node which all values are offset from a single flat plane. Elevation Grids can define the geometry by using an array of height values that specify the height above the surface above each point. This property makes elevation grids suitable for distributed solutions as this limits the amount of data needed to be sent over the network and thus limits the download time. Elevation Grids will be more thoroughly outlined in 4.0.2.

The GeoElevationGrid node, which is a part of the GeoVRML specification, provides the capability to define a grid of height values offset from the ellipsoid or geoid used to model the planet. It supports the specification of height fields in latitude/longitude or UTM coordinate systems.

2.7 Java3D

Java3D is a high level object-oriented 3D scenegraph graphics programming API for the java language. It is built on top of either DirectX or OpenGL to achieve hardware acceleration, and to ensure proper multi platform independency.

Developing with Java3D is done using a scenegraph structure that is independent of the underlying hardware implementation. When working with the scene-graph, the rendering process is left to Java and not the developer. The API provides scenegraph compilation and other optimization techniques towards the requirements of realtime 3D rendering. The scene-graph structure and the event-model in Java3D is heavily inspired by VRML97. However Java3D offer greater flexibility due to being a programming API.

By developoing with the Java platform together with the Java3D extension, one also have the possibility to write applets that can be included in an HTML page. This means that by writing an application once, one could rapidly create a virtual terrain testbed that runs a standalone application (such as Google Earth, Nasa Worldwind) or that resides in a browser (such as Microsoft Virtual Earth).

2.7.1 Serializing

Serialization is the process of saving an object s state to a sequence of bytes in a structured way, as well as the process of rebuilding those bytes into a live object at some future time. Even if serialized objects can be stored to persistent storage, perhaps the most common approach is to transmit the current state of an object using a stream, and restoring an equivalent object from that stream. This approach would fit the client server client mode, that is similiar to Google Earth or Nasa Worldwind. As we will se described in chapter 3 and 4, this is not the way the implemented testbed will be using the serializing capabilities of Java. It will store parts of a Java3D scene to persistent storage, and will be using it as a local caching mechanism.

There are several mechanisms for achieving object persistence such as object databases and disk files. Similarly, there are many ways to share objects such as writing data to a socket or implementing CORBA- or SOM-compliant models. The advantage with using the inbuilt java serializing API

is that it requires no additional library files, it provides a standard mechanism for developers to handle object serialization. The API is small and easy to use and it is well documented and tested.

2.8 Texture Mapping

Texture mapping is the process of adding an image onto a surface in a 3D scene. In the context of virtual terrain rendering, it will often be used for applying ground textures(satellite/aerial texture) onto 3D geometry. Supporting paging of texture imagery can be considered equally important as paging of geometry, due to the large memory print of an image in a graphics card memory, and the limited texture memory available.

One solution to this problem is to cut the image into tiles with sizes that are power of 2 (i.e 128x128 pixels, 256 x 256, 512 x 512), due to the fact that the texture memory is always power of 2 (i.e 2,048 by 2,048 pixels). The cutting of images into smaller equal sizes also enables the graphics API to perform viewpoint culling on parts of the virtual terrain, and hence paging the texture out of memory. One problem with this approach is that it can create visible seams between adjacent tiles. Much of this can be repaired by utilising texture- clamping feature in the underlying graphics API.

As described in Chapter 1, the implemented testbed will connect to Nasa Worldwind geodetic server, and utilise the Landsat7 satellite imagery. A big advantage with this approach is that it enables the testbed to fetch pre-cut, ready-to-use, texture files in .png or .jpg format at the size of 512 x 512 pixels. By utilising already processed imagery, we also reduce the processing on the client side, as it can be utilised directly after it is fetched from the server.

2.9 Summary

This chapter has outlined related applications and datastructures which has had impact on the design of the testbed. Based on the outlined success stories, it seems like a viable approach to utilise the flexibility of Java3D and combine it with many of the same ideas Terravision and GeoVRML use for LOD simplification. The high level Java3D scenegraph, which is inspired by VRML97, can ease the building of datastructures similiar to TerraVision or GeoLOD, because one could utilise inbuilt natural features of the API.

Chapter 3

Testbed Design

This chapter will present the design of important functionality of the testbed software. To provide the necessary understanding of the testbed, important parts of the system are both described at a high level with pseudo code as well as UML software diagrams.

The design of the testbed has favoured simplicity, and is utilising many of the inbuilt features of the Java3D API in order to rapidly create a virtual terrain rendering testbed with many of the same features known in commercial virtual terrain applications. This is done in order to be able to answer the research questions, through collecting of empirical data, by using ”real world” examples.

In the initiation and analysis phase of the project, both user and software requirements, and a scenario for the testbed was written. This was done to increase the awareness of the requirements of the software. This chapter starts off by describing the user scenario which covers the typical usage of the testbed.

In section 3.2 the main thoughts and motivation behind the resulting design is explained.

In section 3.3 a more thorough description of important parts of the the design is given, before the entirety of the design is explained at the end of the chapter.

3.1 Scenario

The user must download a Java Network Launch Protocol (JNLP) file specific for this application using an ordinary web-browser. After double-clicking the downloaded file, the installation process begins. If the user accepts the security warnings, the installation continues. The user is prompted on whether she wants a shortcut on the desktop to ease application start-up at a later time.

When the application has started up, a frame with a black area and a toolbar at the bottom of the

frame becomes visible for the user. After a short while a flat view of a part of the world is visualised.

From the bottom toolbar the user can see a drop-down box and three different buttons. Using these GUI elements different settings for the benchmark test can be set. From the left drop-down the list, the user can choose which caching strategy the application should use. The strategies the user can choose from is

- no cache
- original file formats
- binary tile
- binary quad tree

The three different buttons can be used to run three different benchmark tests. They are labelled ”test1”, ”test2” and ”test3”. By clicking one of these buttons the application will automatically move the viewpoint at various speeds to various places. During these tests, the application software collects test data in the background, when the test-run is finished, the collected data is shown to the user.

3.2 Design

As we can read from the previous chapter, an adequate terrain application consist of many parts. An important aspect of the design of the testbed has been to modularise these parts into software packages in order to reduce the complexity and support change of requirements. Considering that the testbed will be used as a research framework, it is important to have the possibility to replace parts of the system, without rewriting the entire application.

The resulting LOD simplification criteria are based on previous research outlined in chapter 2. The implemented testbed will use a Quad-tree simplification algorithm. Because of similarities in the VRML97 specification and Java3D specification, the design of the testbed is influenced by the GeoLOD node in the GeoVRML specification. The Java3D scene-graph is based on VRML97, meaning that by implementing a GeoLOD node we can utilise several inbuilt mechanisms in the Java3D language, making the implementation task easier.

Listed below are some aspects that were all considered important in order to successfully implement a working testbed within the time-limit and scope for this thesis.

- The GeoLOD node is already well-documented
- The algorithm is proved to function well in both TerraVision and GeoVRML.
- The algorithm and data-structure fit well to Nasa Worldwinds Landsat 7 geo-database

However some modifications and changes are done to adjust the implementation to fit Nasa Worldwinds geo-database. Care has also been taken to handle loading and deletion of tiles. This is done to avoid the memory problems that the different GeoVRML implementations struggle with [15].

3.3 Algorithm and data-structure outline

This section will explain the outline of the implemented algorithm used to render virtual terrains with Java3D. In order to be able to create an adequate virtual terrain rendering system within the scope of the project, the software utilises many of the high-level inbuilt features of the Java3D language to reduce the amount of programming.

3.3.1 Base- and sub-tiles

To clarify and explain the concept behind the algorithm and data-structure of the testbed better, we have introduced two definitions, base- and sub-tiles. They are created solely for the purpose of the pseudo code explanations in this chapter. The definitions are created in order to better explain the adding and removing of data to the data-structure.

The definitions are illustrated in Figure 3.1

- A base-tile is the topmost level of a sub-tile and always has two levels.
  - A base-tile has only 1 sub-level with 4 children
- A sub-tile can be any level below a base tile, and can be a terminating (the last level).
  - A sub-tile can have 1 level with 0 or 4 children

As already stated in chapter 2, we can use proximity to control the resolution of the terrain by using a discrete LOD. We can also imagine to use proximity to decide how many tiles we have loaded into the scene at a time.

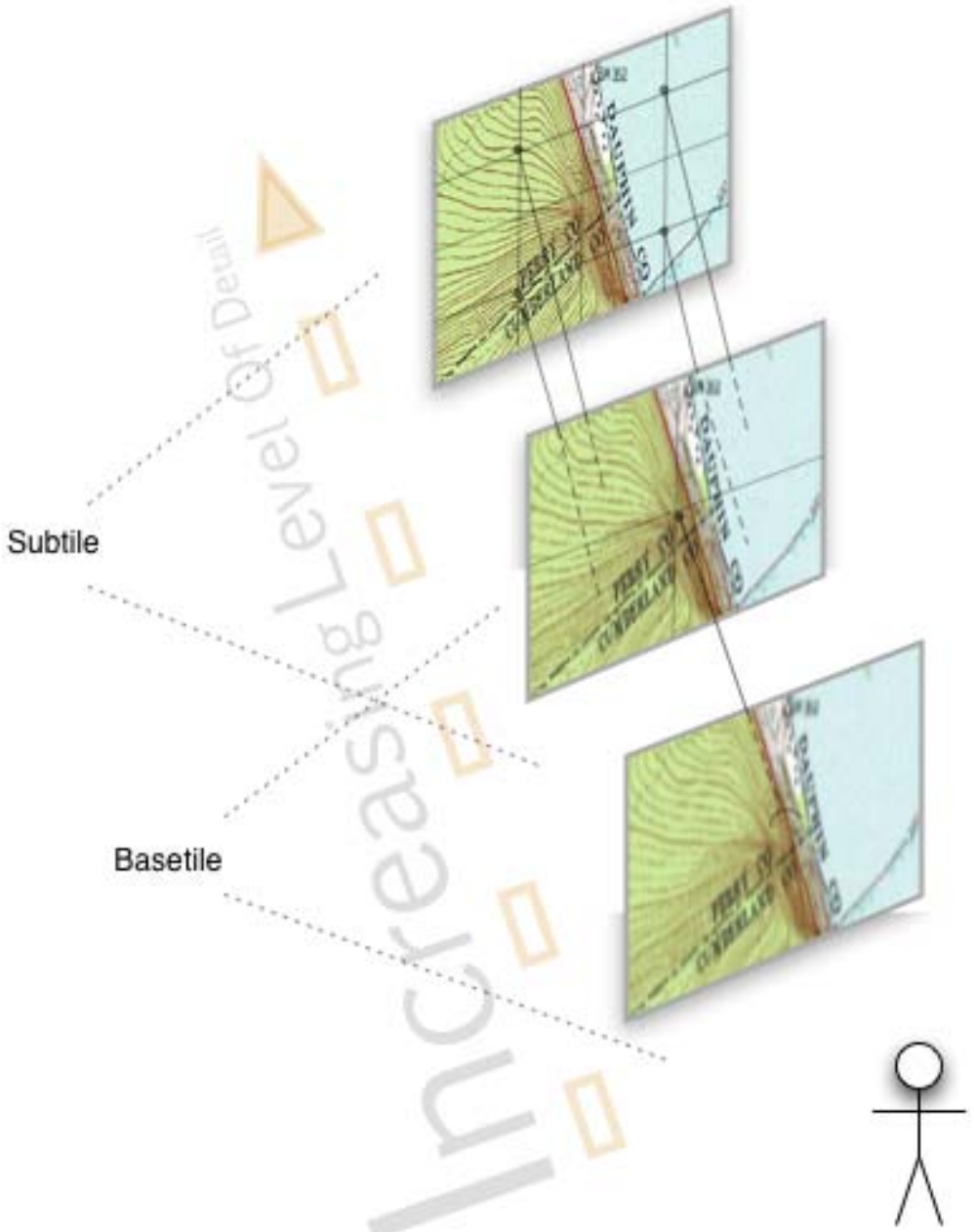


Figure 3.1: Visual explanation of base- and sub-tiles

Controlling Loading and unloading of tiles

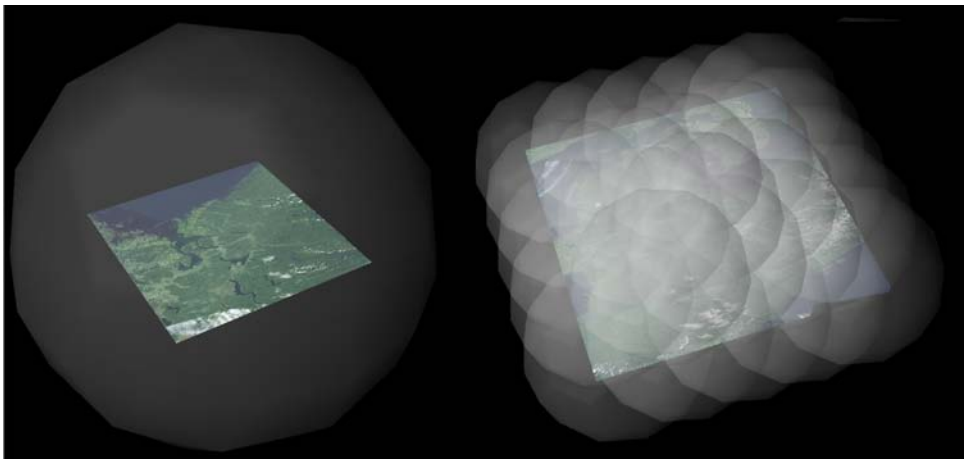


Figure 3.2: To the left a base-tile with scheduling bounds visualised from a far distance. To the right a base-tile with scheduling bounds for all levels visualised

The loading and unloading of tiles in the testbed are controlled by using the distance from a tiles centre to the users viewpoint. To illustrate this we can imagine having a visual bounding sphere around each tile, as illustrated in Figure 3.2. The visual bounding sphere is defined as a tiles scheduling bounds.

Whenever the user s viewpoint is intersecting with this bounding sphere we need to take some decisions. The testbed distinguishes between when the user is entering and exiting a tile s scheduling bounds.

```
when ( viewpoint enter a tiles scheduling bounds )
        if ( base-tile scheduling bounds intersected entry )
                a. check if neighbouring tiles are missing
                        if ( missing )
                                i. load missing neighbouring tiles
                                ii. add missing neighbouring tiles
```

Each time the user is exiting a tile s scheduling bounds, the following check is performed.

```
when ( viewpoint exit a tiles scheduling bounds )
        if ( base-tile scheduling bounds intersected exit )
                a. Delete exited base-tile with all sub-tiles
```

The deletion of tiles is necessary in order to reduce the amount of rendered tiles in scene, and to reduce the amount of used memory. By removing the tiles tiles furthest away from the user s viewpoint, we will increase the rendering performance, as we reduce the amount of rendered geometry. This can be also considered as a very simple approach to out of core rendering, where we page the not so important tiles out of memory.

Increasing the level of detail

To load new LOD levels, and to enable the testbed to increase the level of detail around the users viewpoint, the same technique is used. When the users viewpoint is intersecting with a sub-tile s scheduling bounds, some decisions need to be taken.

```
if ( sub-tile scheduling bounds entry )
                a. check if intersected tile is terminating
                        i.load next tile
                        ii.add new tile
```

Switching between LOD levels

An important feature of the testbed, is that sub-tiles are never deleted and paged out of memory. The algorithm uses Discrete LODS to switch between LOD levels. Sub-tiles are never paged out of memory before the topmost base-tile is deleted from the scene. To switch between between different LOD levels, each base and sub-tile are connected with a Discrete LOD. This makes sure that the tile with the highest resolution is always closest to the users viewpoint. This feature is shown in Figure 3.3.

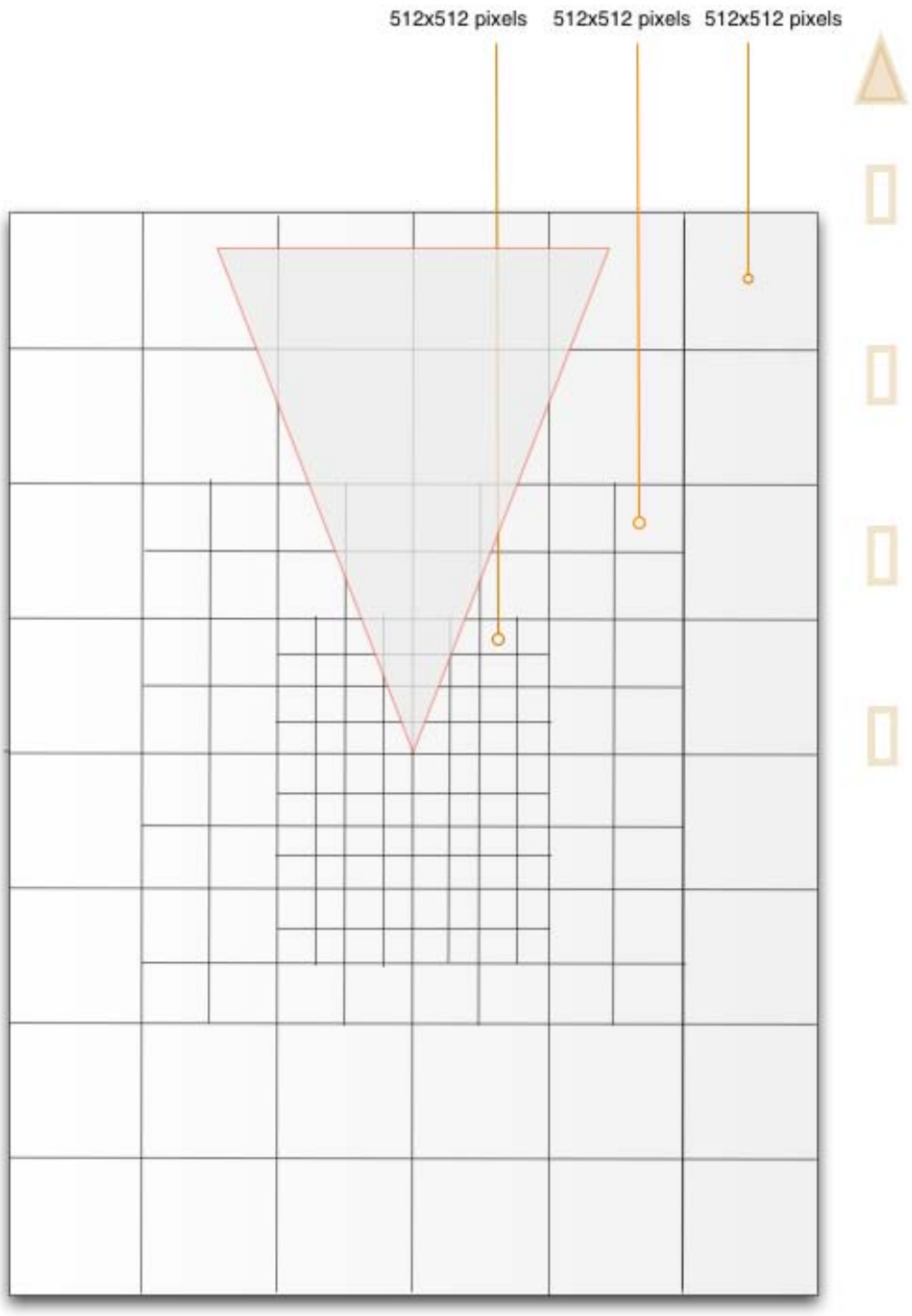


Figure 3.3: Resolution controlled by discrete LODs

3.4 Coupling with Nasa Worldwind Server

For the testbed to be of any practical use, and to be able to answer the Research objective : ”What is the overall impact on performance on multiresolution virtual terrains when utilising Java3D serialised binary files compared to utilising files from a Nasa Worldwind Server”. The implemented testbed must be able to use the data from Nasa Worldwind servers.

3.4.1 Imagery

The satellite imagery used in the testbed is the same as used in Nasa Worldwind, but is restricted to Landsat7 imagery set. Nasa Worldwind stores all imagery on a server using Plate Carrèe or Equirectangular projection. The Plate Carre is a simple projection method, which converts the globe into a grid of perfect squares. The horizontal coordinate is longitude, and the vertical coordinate is latitude, with no transformation or scaling applied. To request a specific image on a Nasa Worldwind server, we need to send a request to the NWW server with the Plate Carrèe coordinates and which detail level that is desired.

As described earlier the detail level is dependent on the viewers viewpoint above the ground, this is also the case with the imagery data.

3.4.2 Elevation Data

To give a realistic impression of the virtual terrain we need to combine the satellite imagery with elevation data. There are several approaches to retrieve elevation data from a NWW server. The approach that will be used in the testbed is to utilise the NWW Java open source code-base and couple this with the implemented testbed. The NWW codebase can give an elevation value for any given latitude and longitude value. This will enable the testbed to look up and use the same elevation values as NWW. To combine these two properties the testbed will need to able to convert from any point on a tile to a correct Latitude and Longitude value, in order to visualise the terrain naturally. How this is solved, is outlined in Chapter 4.

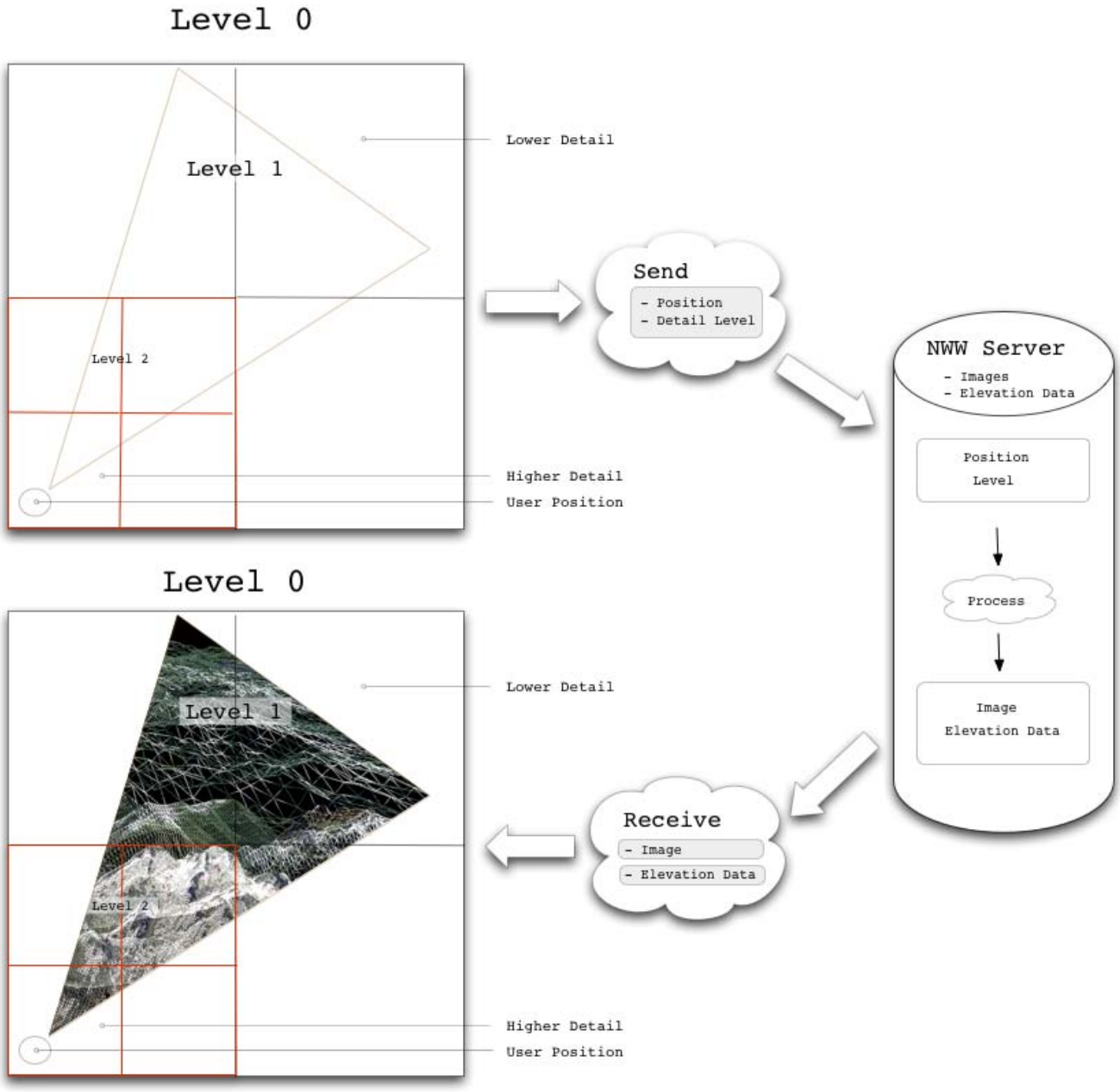


Figure 3.4: fetching data from Nasa Worldwind Server

3.5 Design Limitations

A limititation in the current design is the handling of the movement of the user s viewpoint at high altitudes. When the viewpoint is moved at altitudes above a base-tiles scheduling bounds, the outermost base-tiles from the viewpoint does not get the notificiation that the viewpoint has exited their scheduling bounds. This is because the implementetation utilise predictive loading when loading new base-tiles. In practise this may result in adding base-tiles with scheduling bounds outside the user s viewpoint.

This limitation can avoided in several ways. A simple improvement to the implemented testbed would be to check the users altitude and add more base-tiles when the user is moving upwards, and delete them again when she is moving closerto the ground.

The simplest solution would be to turn off the predictive loading, and make sure that all base-tiles scheduling bounds can be intersected. This is however not very satisfactory in implementations that only support one dataset, as this would decrease the performance of the systems. In systems that supports other datasets at lower resolutions (i.e. Blue Marble), one could avoid this limitation by having tiles at lower resolutions loaded instead of the outermost base-tiles.

3.5.1 Handling high travel speeds

The predictive loading of base-tiles occurs when the viewpoint intersects a base-tile. If the user is traversing at a higher speed than the implementation manages to load the neighbouring tiles, a situation can arise, where no further base-tiles are loaded. The implemented testbed avoids this shortcoming by not letting the user move at very high speeds.

3.6 Design Strengths

One of the design strengths in the testbed is the progressive adding of meshes as we move closer to the ground. A dynamic top-down building of the quad tree scene-graph structure has the advantage that it maps very well to the image file-structure on the NWW server. All images in the NWW Landsat7 database are pre-loded and can placed directly in a Quad-tree algorithm after they are downloaded.

Another advantage compared to a bottom-up approach is that the testbed does not need to have the highest resolution data available before start-up. This property gives us several advantages.

- Progressive memory usage, do not load tiles into memory before they are needed.

- Uses less initial time, because it is not necessary to process the highest resolution data during start-up.
- Requires less processing on the client side.

The solution with progressive memory usage solves the problem many GeoVRML implementations struggle with [22], because they load the entire scene into memory during startup.
However, top down approaches is not without problems. A common problem with top-down approaches is that they often produce cracks between adjacent tiles. This problem is not addressed in the work with this thesis, as it is considered out of scope for a project of this size.

3.7 Model Description

As a part of the initial design of the testbed, both user and software requirements where created(See appendix A,B). This was done to augment the understanding of the implementation. Further logical models were created to depict the software requirements. The models are based on object-oriented UML diagrams. All requirements and the UML-models were created before the actual implementation started in order to increase the awareness of the different problems that could arise, and to create an initial overview of the development-priorities of the different parts of the software. This documentation was considered to be used only as initial input to the actual development process, this means that they are not synchronised with the resulting software. The UML-diagrams are not updated continuously with the development of the software, as working software is the measure for success and not documentation.

The class diagram shows a simplified description of the system, it identifies the main components and the classes of the system. Class diagrams also show the inter-relationships and the operations between the classes. The main components are represented as packages. The objects residing in the packages are related to each other and solve a common task. Please note that the classes in the diagrams are just representations of objects, hence a class diagram is implementation independent, and can represent any object-oriented programming language.

All classes that are related in functionality are put into packages for better maintenance and organising of files. Some packages also have a subpackage named j3d, all functionality related to Java3D and hence 3D graphics are put here. This is done to separate the 3D graphics functionality from other functionality. This results in better flexibility in the design because we will not be as tied up to one 3D graphics library. By using this structure, and as long as we have clearly defined interfaces between packages, we will be able to replace parts of the system easier, and i.e. replace mesh geometry without affecting other parts of the system.

To supplement and clarify the static class diagrams, we have used a sequence diagram to represent the dynamic behaviour of the system. A sequence diagram depicts the sequence of actions that occur in a system. The diagram is made up of objects and messages; the objects are represented on the same way as in the class diagram. The sequence diagram shows the the execution of the implementation as described in the scenario.

3.7.1 Component Description

In order to be able to support rapid changes of requirements, the software is divided into software modules. The components were identified during the logical model description phase. The component description was written in order to get further insight of each component.The initial component description is added in appendix C.

The component description is used to describe the main function of each component and the relations between each component. This is all valuable input before the actual development starts.

Chapter 4

Testbed Implementation

This chapter presents the most important and challenging parts of the implementation of the designed testbed.

4.1 Elevation Grids

The implemented module uses regular elevation grids as building blocks for the geometry. Elevation Grids are well known from both VRML97 and the ISO GeoVRML standard, and have proven to be successful for virtual terrain rendering [20] [18]

As we can see from Figure 4.1, an Elevation Grid can easily be modified in both X and Z-directions, which simplifies the creation of rectangular grids at different resolutions. The geometry is described by an array of height values that specify the height above the surface for each point in the grid. This property makes elevation grids suitable for distributed solutions as this limits the amount of data needed to be sent over the network and thus limits the download time. The example in Figure 4.1 needs only an array of 4x4 = 16 elevation values in order to describe a terrain mesh.

4.1.1 Separate height values

Because of the inherent structure of virtual terrains, where the elevation data (DEM files) are separated from the imagery, it would be an advantage to reflect this property in the implementation of the testbed. As we can see from Figure 4.1, the use of elevation grids fulfils this task, as elevation grids make it possible to separate the elevation values from the resolution of the data. This adds flexibility to the design and implementation of the virtual terrain application as we can easily switch

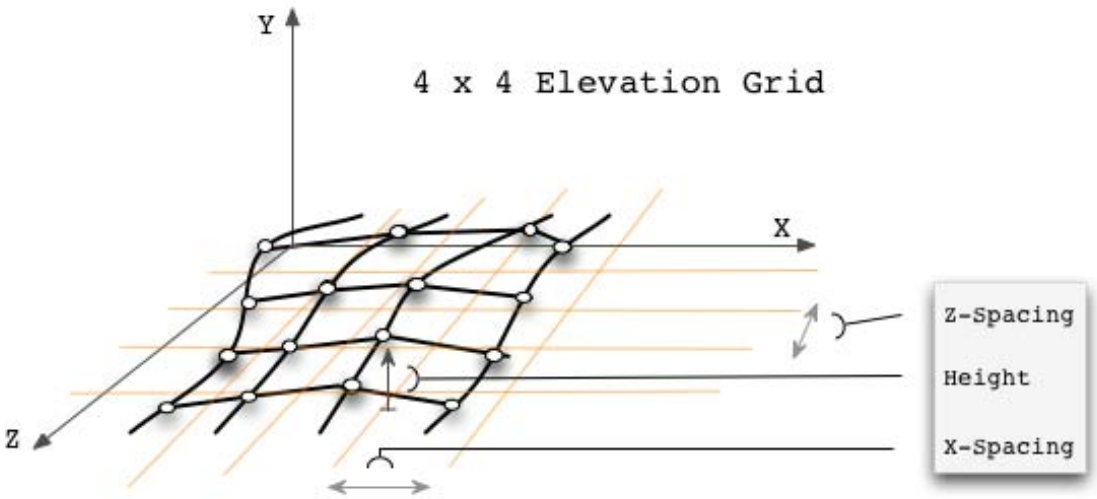


Figure 4.1: Basic Elements of a Elevation Grid

sources for retrieving elevation data. This is an important property for applications that should be capable of visualising geo-referenced data, as the data sources can be diverse.

4.2 Getting height data - using the NASA Worldwind Server

The geodetic data residing on NASA Servers are organised using a Plate Carrèe projection or equirectangular projection. This projection handles the spherical Earth as a flat map, and is often used in virtual terrain applications because of the direct mapping of X and Y axis to latitude and longitude.

For the module to be able to render a specific geographic region, it requires a valid latitude and longitude pair of values. When these are supplied, the module can connect to the NASA Worldwind server and retrieve the necessary data in order to handle the visualisation of the requested geographic area. A conceptual figure explaining the data-flow and connection between the testbed and the Nasa Worldwind server is illustrated in the figure 4.2.

To get the correct geodetic data from the NASA worldwind server, the module must be able to understand how NASA has organised the geodetic data. The following explanation is taken from[28]

> *"World Wind uses what is defined as Level Zero Tile Size to determine how large (in decimal degrees) each tile is in width and height (all tiles are square). A standardized level zero tile size is under consideration but is not yet implemented, but it must divide*

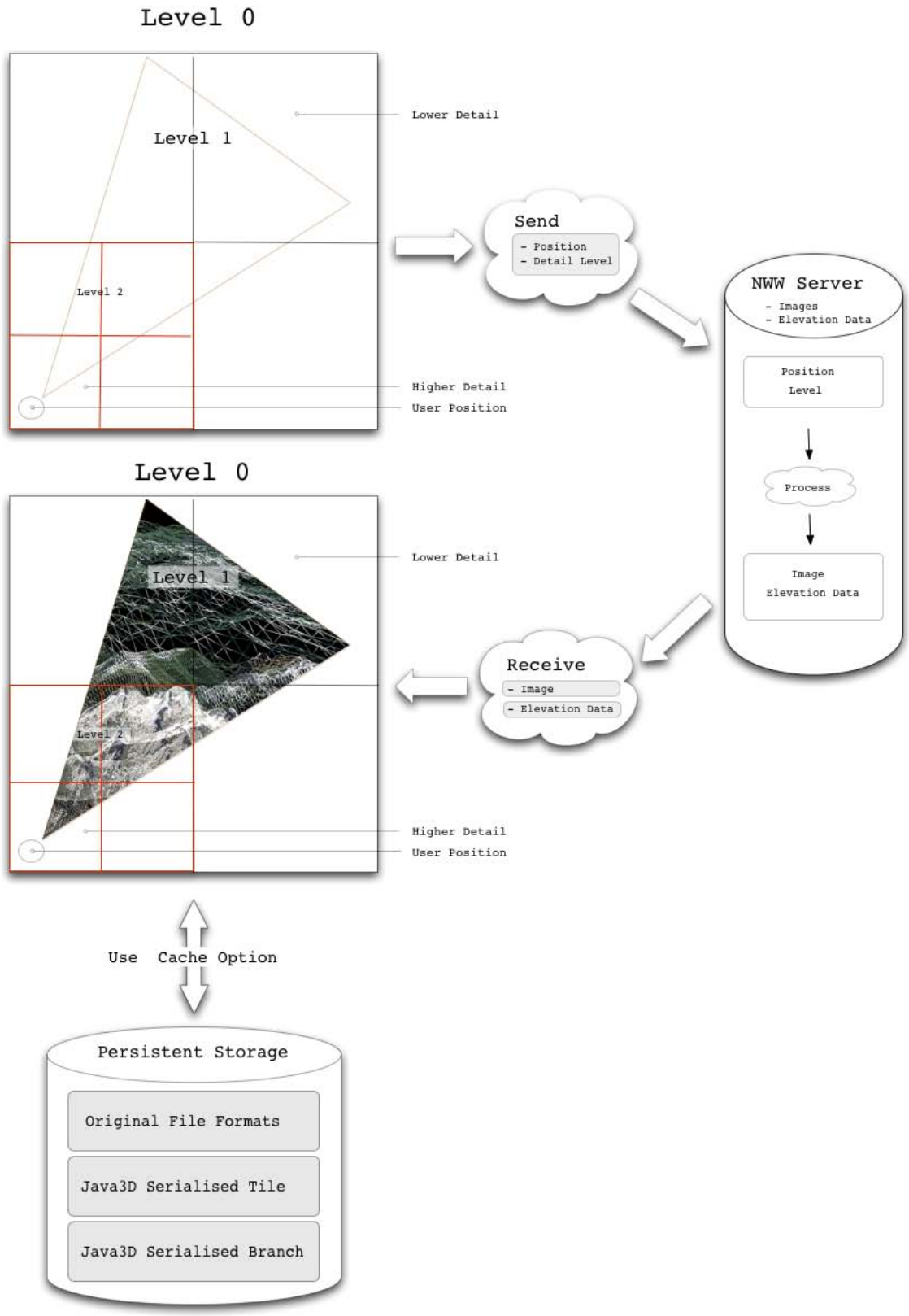


Figure 4.2: Getting data from NWW server and store processed data to Java3D Serialize

into 180 evenly. The level zero tile size (herein referred to as lzts) is simply the distance traveled in degrees from one side of a tile to the next side. In the NLT Landsat 7 datasets, the lzts is normally set as 2.25 degrees. For each increase in level, the lzts calculated using this formula:

$$lzts * 0.5^{level}$$

What this does effectively is decreases the tile size by one half for each increase in level. Where their was one tile, there is now four"

This projection mapping scheme fits very well to our module s design, as we can directly fetch data from a NWW server, and place it progressively into a multi-resolution quadtree structure.

To find the geographic area in Plate Care projection covered by any valid latitude and longitude values, the following equations must be performed. The first equation converts from latitude and longitude values to Plate Carrè projection, where n = level or amount of detail.

$$x = 180 + \frac{longitudeMOD(360) \times 2^n}{2.25}$$

$$x = 90 + \frac{latitudeMOD(180) \times 2^n}{2.25}$$

The result of this equation gives us the Plate Carrèe x, and y values, describing a geographic area around any given latitude and longitude. By using these x and y values, we can extract elevation values for a geographic area by using any given latitude and longitude using the following equations. To do this we first need to find the lowest (minimum) latitude and longitude for a tile in Plate Carrèe.

The minimum longitude is given by:

$$2.25 + \frac{x}{2^n} - 180$$

The minimum latitude is given by:

$$2.25 + \frac{y}{2^n} - 90$$

When having calculated the minimum values, we can easily find the maximum latitude and longitude for this tile, by adding the Level Zero Tile Size(lzts) for this tile s level.

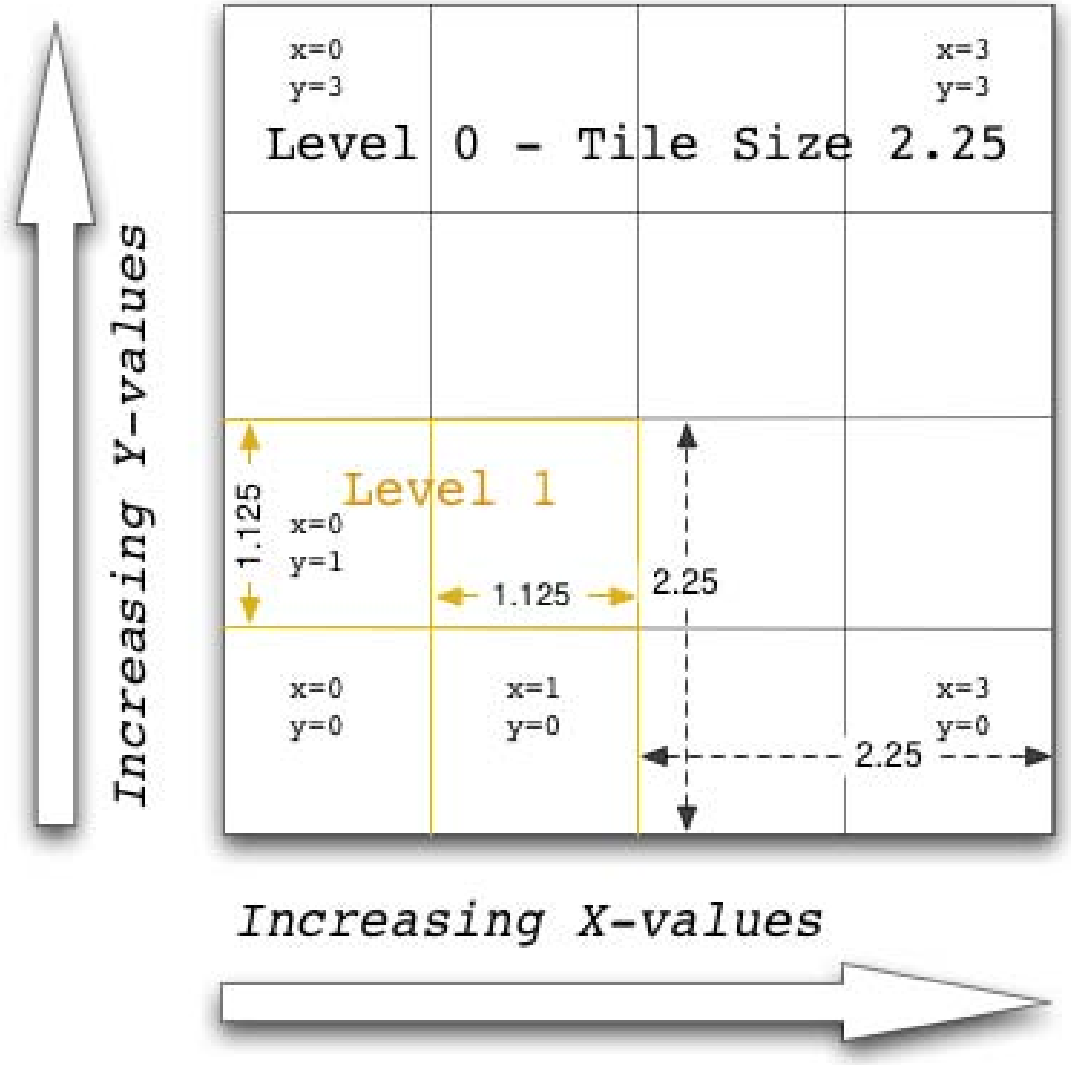


Figure 4.3: Visual Explanation of Tile Structure on NWW server

4.3 Resolution

The implemented testbed is adjusted to utilise the highest resolution data available from NASAs geodetic database, producing elevation grids with an x- and z-spacing of 32m at the highest resolution level in the LOD structure. The x- and z-spacing for the following lower resolution levels are calculated using the 2 logarithm of 32, giving X and Z spacing values of 64m for the next level, 128m for the next level after that and so on.

NASA Worldwind uses DEM raster files that contain 150x150 elevation samples per tile. To fetch elevation values on a tile, we need to sample 150 times in the x and y direction between the minimum and maximum latitude and longitude.

4.4 Programmatic LOD in implemented testbed

One of the advantages by utilising a regular elevation grids as building blocks in a quad-tree structure, is that we can easily adjust the resolution to different tiles at different levels. This can be done by increasing the x and z-spacing, and reducing the x and z-dimensions. The example in table 4.1 would produce two different tiles at different resolution but would represent the same geographic area.

Table 4.1: Example that show how to represent same geographic area at different resolutions

x-spacing (m)	z-spacing (m)	x-dimension (m)	z-dimension (m)	area in x,z-dimension
32	32	150	150	32 * 150 = 4800m
64	64	75	75	64 * 75 = 4800m

As described in chapter 2, a desirable feature in a terrain rendering system, is to be able to reduce detail of distant parts of the scene. In the implemented testbed a simple detail reducing system is implemented to reduce detail on far distant parts of the virtual terrain. The reducing system in the implemented testbed is a simple table where it is possible to get a reducing factor number for any given level in the hierarchical LOD.

The decision of using elevation grids with a size that corresponds to the NWW SRTM30+ bil-files in the implemented testbed, which stores data in a raster of 150x150, can in this context be considered as a design flaw. By not using tile-sizes that is power of two, we have only a limited set of integers to reduce the tile resolution. In this case it is possible to reduce the tile spacing with 4 (150 / 4 = 37.5), because we are not returning an integer. Considering the very flexible NWW

elevation data retrieving system the implemented testbed is utilising, it would be a fairly easy task to fix this design flaw, but it was not considered important in order to get the necessary test data.

The integers used for each level in the testbed are as indicated in Table 4.2

Table 4.2: Reducing integer factors at different quad-levels

Level	Reducing Factor
0	6
1	5
2	3
3	3
4	2
5	1

By using a reducing factor of 1, we assure that we are always showing the highest available data at the lowest level in the quad-tree.

One can imagine utilising this feature more dynamically, as these values can be adjusted real-time in software. If i.e. a user is travelling at high speeds, the same detail level is not required as slower speeds. This could dynamically be adjusted real-time in software by increasing the reducing factor in the programmatic LOD algorithm. The same feature can also be taken advantage of to control the frame-rates, by increasing or decreasing the frame-rate runtime, we can adjust the detail in scene dependent on the available computing power.

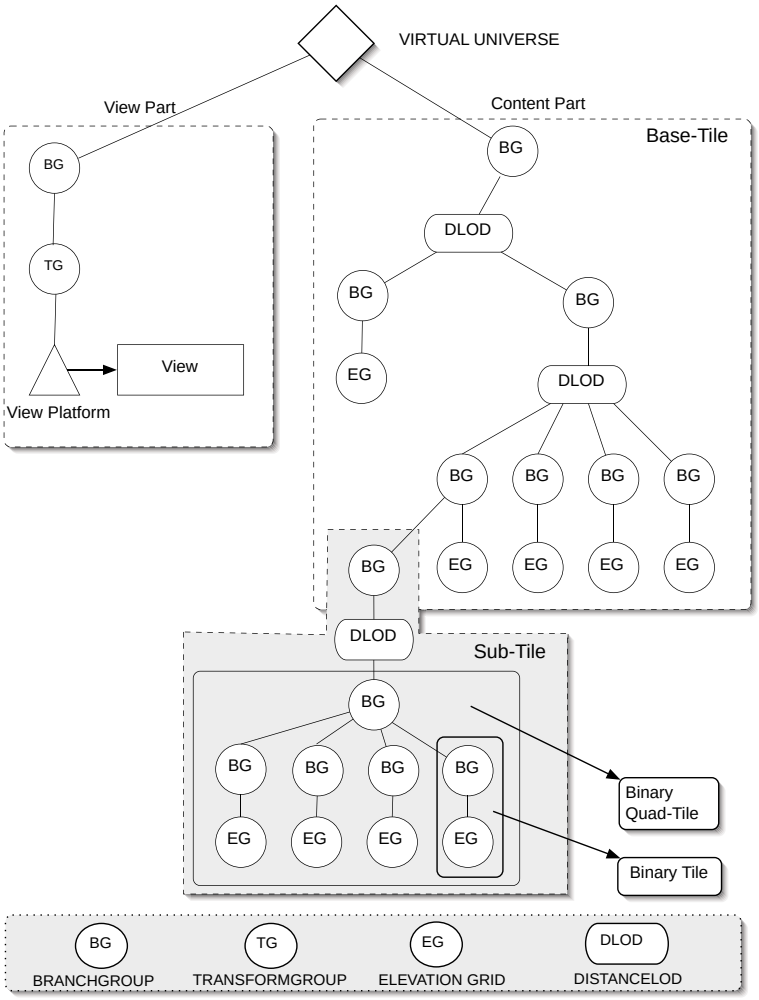


Figure 4.4: The Java3D quadtree scenegraph representation in the implemented testbed

4.5 Quadtree Java3D Scene-Graph representation

The scene-graph is divided into to main parts, the content part and the view part. This is done to cleanly separate the geometry from the part that controls the viewpoint.

To better understand the structure of the scene-graph, and the Figure 4.4, it is necessary to very briefly explain the differences between basic-group types in Java3D.

TransformGroup (TG): Group type that is used to set rotation and translation

BranchGroup (BG): Connector node, the only node in Java3D that can be connected to a live scenegraph.

ElevationGrid (EG): A Java3D Shape that can render textured height maps

Distance LOD (DLOD): A discrete LOD that uses distance as selection criteria for different tiles at different resolution

4.5.1 View Part

The view part is visualised in the left side on the Figure 4.1. This part is contains all necessary parts needed in rendering a three dimensional scene from one viewpoint.

The viewpoint is controlled by a TransformGroup node, that is used to control the translation and rotation of the view. This node is for running flight-path animations and for the mouse-navigation that is implemented.

4.5.2 Content Part

The content part is visualised in the right side on the figure below. It is utilised a Quad-Tree structure, where Java3D nodes is used to construct the correct hierarchy.

The upper rectangle shows the Java3D representation of a base-tile. One of the most important feature to notice is that higher resolution Elevation Grids at are added to the scenegraph using Java3D distance LOD nodes. This feature assures that the highest available resolution tile always is displayed closest to the users viewpoint.

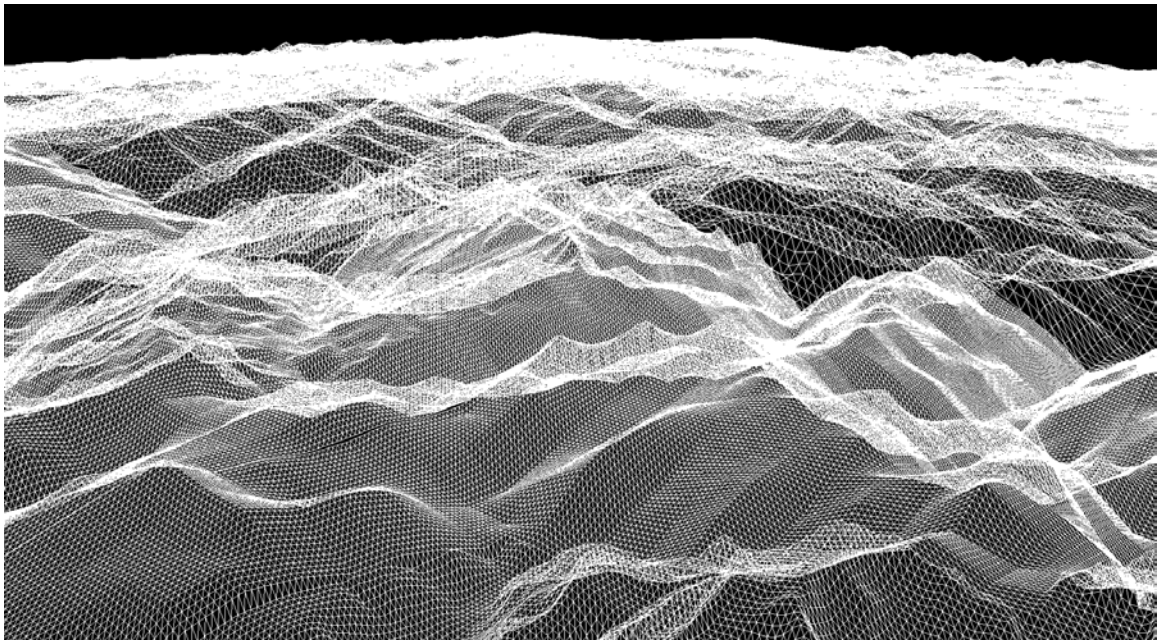


Figure 4.5: Screenshot from implemented module visualising a high resolution terrain illustrating the multiresolution capabilities, note that the higher resolution tiles are closest to the viewers position

4.6 Java3D Binary Serialisation

The implemented testbed is developed with support of storing parts of the virtual terrain by using serialisation. The process of serializing java objects is done by converting an object into a sequence of bits so that it can be stored to persistent storage or transmitted across a network with a stream.

The serialized java object can be re-read according to a serialization format, and can be used to build up an identical clone of the original java object.

The Java3D API has inbuilt serializing mechanisms that supports this method of storing Java3D scene objects. Java3D provides an interface for saving and restoring scene-graphs both from a Java Stream and/or for storing in Random Access Files. The implemented testbed uses the provided feature-set to read and write branch-graphs to persistent storage in order to rebuild the already processed virtual terrain

By reading already processed tiles, the chain of processes that build up the quad-tree structure change significantly. The table below lists the differences of “main-events” when using the different file-formats. The listed events are the ones that are assumed to have an potential effect on software performance.

For textured virtual terrains:

Original File Format	Binary Tile	Binary Quad-tile
-	Read Binary File	Read Binary File
Read Texture File	-	-
Generate Texture	-	-
Generate Texture MipMapping	-	-
Read Height Data File	-	-
Parse and process elevation values	-	-
Render	Render	Render

For non-textured virtual terrains:

Original File Format	Binary Tile	Binary Quad-tile
-	Read Binary File	Read Binary File
Read Height data file	-	-
Parse and process elevation values	-	-
Render	Render	Render

4.6.1 Binary Tiles, Binary Quad-Tiles

When looking at the tables above that lists the differences in processes, one can see that the computational cost for the binary formats are smaller. This is due to the fact that we can load already processed tiles directly into the correct position in the quad-tree structure. Because of this, one can assume that this will have an impact on performance when rendering large virtual terrains.

To investigate and try to understand the performance effect of binary Java3D serialising further, the testbed stores different quantities of the virtual terrain. The testbed fulfils this task by storing either one tile in a quadtree-level or all tiles in a quadtree-level to persistent storage. In order to be able to differentiate when one of the two are used, we have introduced two new definitions; Java3D Binary Tile and Java3d Binary Quad-Tiles. These definitions are used throughout the rest of the document, it is however important to note that these definitions are only used within this work, and are not a general terms in the field of virtual terrain rendering,

Original File-formats The tile information is the same as retrieved from a NWW Server.

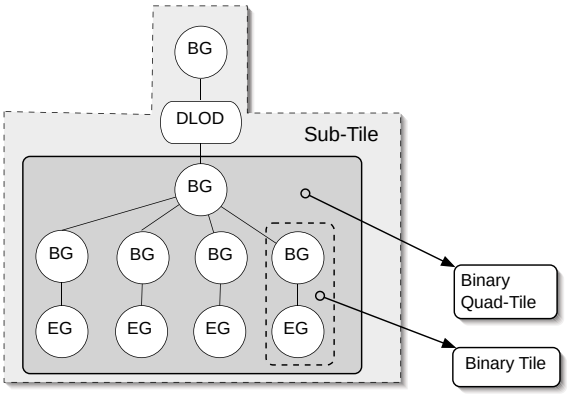


Figure 4.6: Visual Explanation of Binary Tiles and Binary Quad-Tiles

Java3D Binary tile One binary serialised Java3D elevation grid with parent BranchGroup for any given LOD level

Java3D Binary Quad-Tile All binary serialised Java3D elevation grids for this LOD level with parent BranchGroup

As we can see from the Figure 4.6, the Java3D Binary-Quad-Tile stores 4 times the amount of ready processed data as Java3D Binary Tile. This is done in order to understand if there can be a performance gain by reading larger chunks of ready-processed data compared to smaller chunks of ready processed data. However, one should have in mind that these files will grow to be approximately 4 times larger in size.

4.6.2 File IO

The different file formats are written to disc by using elements from the Plate Carrèe projection information. This guarantees that each tile is described by an unique identifier, and we can easily read/write tiles because we are already using this information to deploy tiles in the software. To assure that the software has read and write access to the disc, all application data are stored on the users home directory.

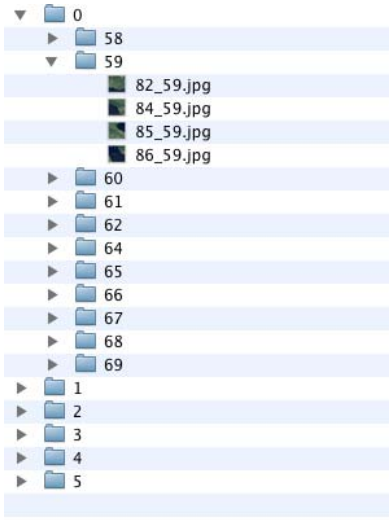


Figure 4.7: The files are stored using the Plate Carrèe projection information

4.7 Threading

The software is developed using the iterative and incremental development method, this method is suitable where it is difficult to know all requirements before the development starts. This was also the case with threading.

When developing software with so many parts that should work together, it is very hard, if not impossible to predict which parts should be threaded or not. For this reason the software was initially developed single-threaded, and then analysed which parts would benefit the most of becoming threaded.

The initial testing proved that reading much data from disc sequentially and process data afterwards was a bottleneck for the software. This task was then threaded asynchronously, which gave the software a boost in performance.

This approach could not be kept because of Java3D issue 4340607. (see appendix H for description) Java3D has issues with setting the geometry component on two Shape3D nodes under same branch concurrently. This issue makes development of Multithreaded applications with Java3D more difficult.

This forced the application to load an entire sub-tile simultaneously, and synchronise the setting of geometry components. Making this change avoided the problem with setting the geometry component concurrently.

Each subtile is loaded and set using the core java ExecutorService threadmanager. Using a

thread manager has several advantages in a multiresolution terrain rendering system, perhaps the largest advantage is that it is possible to terminate ongoing asynchrounous tasks and reuse them quickly. Using threads can produce much memory overhead, and having the oppurtunity to reuse threads can reduce the total amount of memory used.

4.8 Summary

The implementation of the testbed is using a hieararchial quadtree structure, which is heavily inspired by the design of the GeoLOD node, for LOD simplification. It is integrated with the NWW Java codebase, to be able to render a specic geographic region.

For caching of the rendered virtual terrain, the testbed is capable of storing different quantity of a Java3D scenegraph as binary serialized files. These files are very important for the performance investigation of the testbed.

Chapter 5

Performance Investigation

5.1 Test Plan Overview

The main focus during the performance investigation is measuring the overall software performance on a textured and non-textured multi-resolution terrain rendering system when using pre-processed Java3D binary files compared to original file formats. In this context the original file formats are the formats that are available from the Nasa Worldwind Server.

As explained in Chapter 4, a terrain rendering system consists of many different independent parts. To predict how the different parts affect each other when using pre-prossessed or unprocessed files is not a trivial task, and for that reason, the test plan will focus on the overall performance of the software when using different file formats, and not on specific parts of the implemented testbed. The testing will favour the file format s ability to deliver high resolution data

The software performance effect will be measured by comparing the quantity of data that is processed when running predefined viewpoint animations when using different file formats. The accumulated number of tile levels displayed on screen during a flighpath animation will be used as one of the measures for effectiveness. To be able to differentiate the results, the implemented testbed uses timing mechanisms to measure how long time a LOD level takes to load, until it is displayed on screen.

In addition Frame Per Second (FPS) tests are run to monitor the rendering performance. The FPS will not be used as primary measure for software performance effects, but will be used as a measure to determine if a test has passed or failed, and to give indications concerning rendering performance.

5.2 Features to be tested

The tests are run using predefined flight-paths running in the 3D-view that simulate normal user operations in virtual terrain software. As we can see from Figure 5.1, several testing tasks are performed simultaneously during the entire data collecting process. This is done to precisely measure the software performance effect of using different file formats.

During execution of the flight-path, the following data are collected.

- The accumulated number of quadtree levels loaded during the entire flight-path animation
- Time to load an entire quadtree-LOD level
- The number of loading attempts to load an entire quadtree LOD level
- The Frame Per Second

The rendering framework of the implemented testbed takes a naive approach to rendering, and is always trying to render as many quadtree LOD-levels as possible, without considering aspects as such FPS, CPU load, memory usage etc. Rendering high resolution data affects the rendering performance and hence the FPS. If a test situation should occur where we are trying to load too much data, the rendering frame-rate will drop to an unacceptable level. This limitation must be assigned to the rendering framework and not to the file format used in the test. For this reason FPS-tests are not used alone as a measure for software performance, however FPS tests will be used as a test pass/failed criteria.

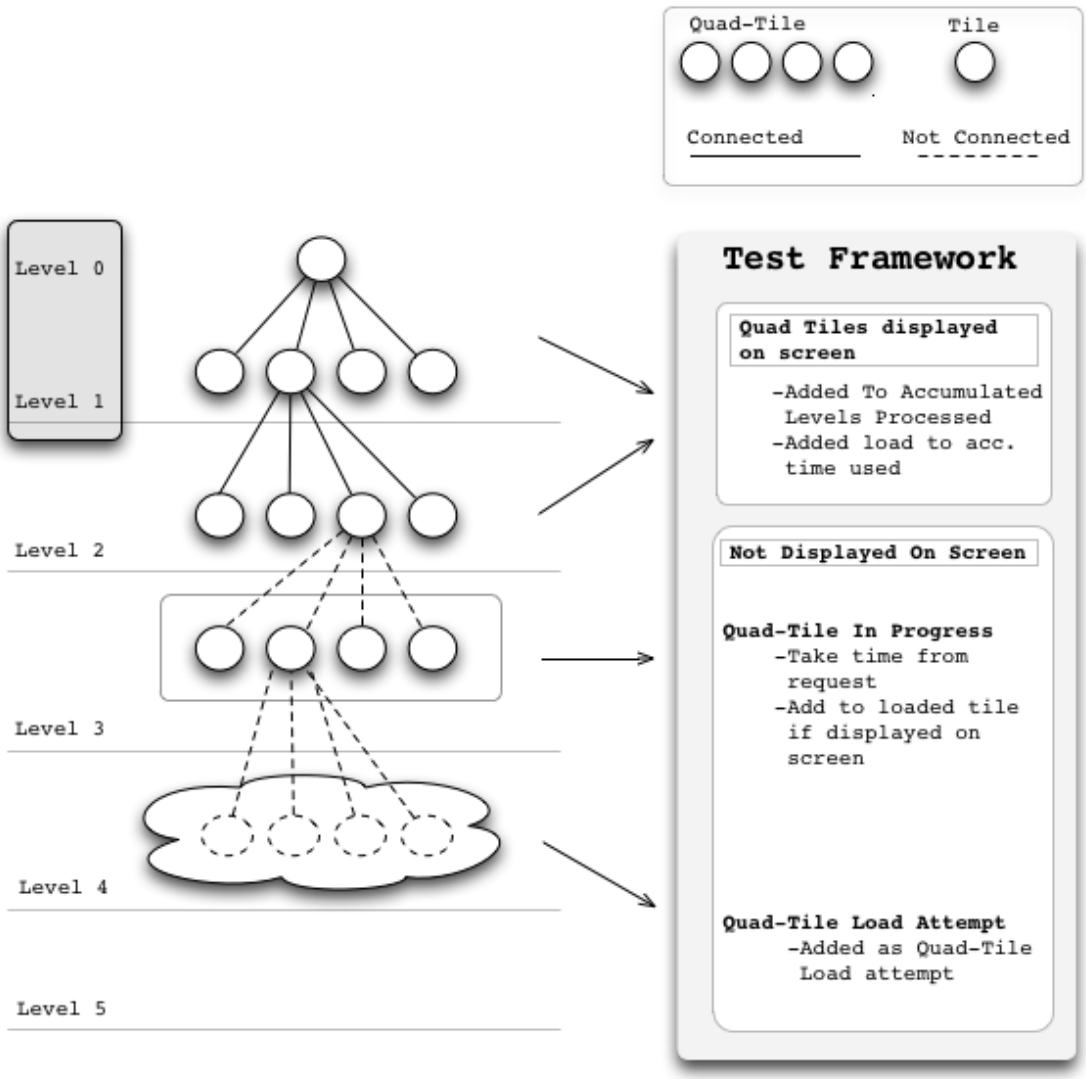


Figure 5.1: Test data collected during an execution of a test

The implemented testbed measures the accumulated number of quad-tiles loaded into the rendering framework, and uses it as one of the measures for effectiveness when loading new data. It also collects the number of loading attempts to load an entire LOD level. This measurement can be used to track the number of missed attempts of loading tiles, giving an indication of the file format s ability to provide the rendering frameworks with data. A high number of missed attempts indicates that the loading process may be too time-consuming.

In addition to this, the software uses Java timing to measure the time elapsed from when a new

level is requested for loading until it is displayed on screen. This measurement is performed to be able to differentiate the measurements better.

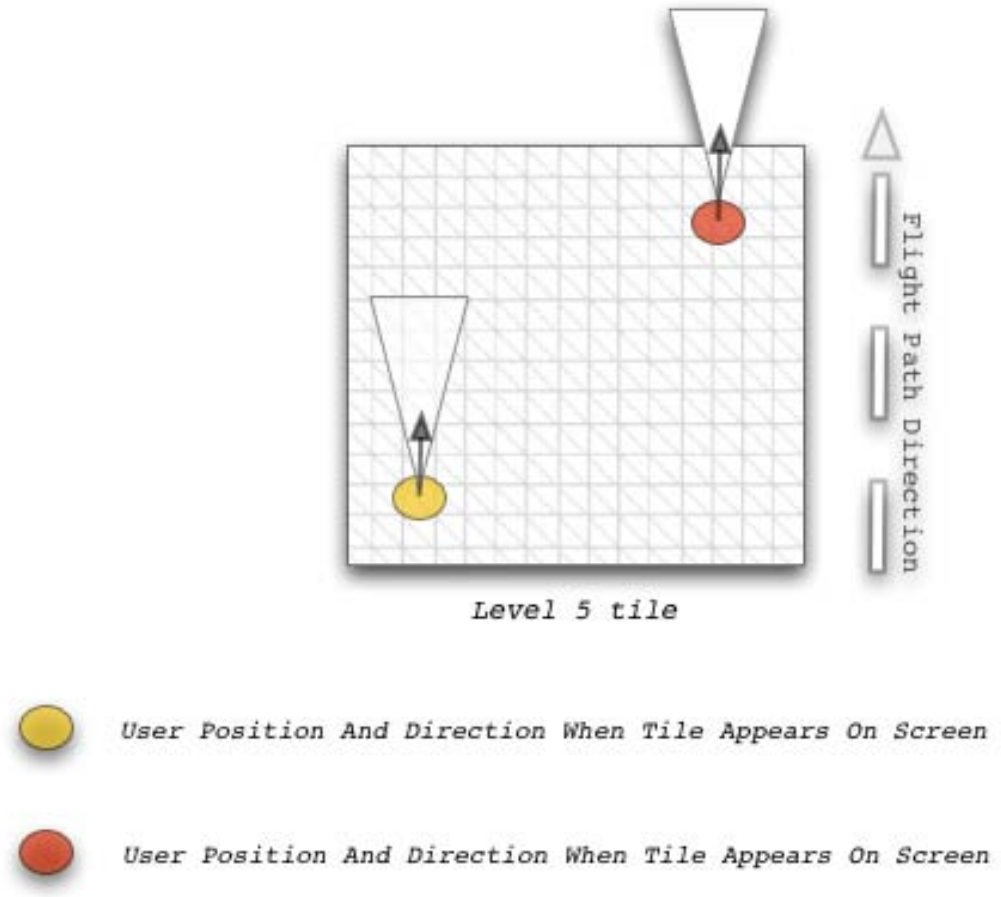


Figure 5.2: Two users at different positions at the moment when a tile is displayed on screen

The rendering framework can produce results as shown in Figure 5.2. The Figure illustrates the moment a tile is displayed on screen, but with two different user positions. If only accumulated number of tiles were used, both situations would have been added to the accumulated number of tiles, and would have counted the same. As we can see, the “yellow-user” has a longer remaining traversal time on the same LOD level compared to the “red-user” To assess this weakness, the software calculates the time from when the LOD-level starts loading until it has finished. With this data available, one can also easily calculate which file format enables the rendering framework to display a quad-lod level quickest on the screen.

5.3 Test Data

The implemented testbed is capable of reading and writing all necessary data from several different file formats. All file formats are based on, and contain, data that can be retrieved from the Nasa World Wind server. The test data is collected by running the predefined flight paths and writing the test data to disk. All test data is collected and stored to disk before the actual test is run.

The implemented testbed measures and compares 3 different strategies for rendering a LOD-tile.

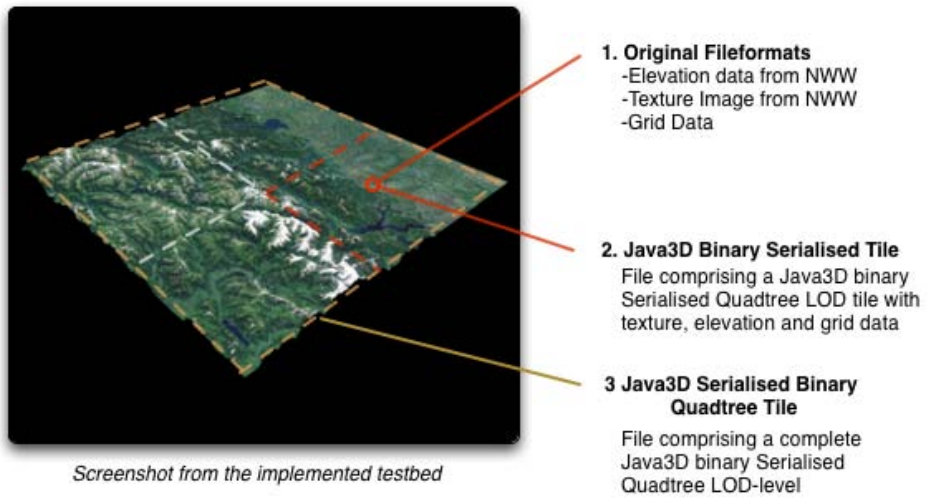


Figure 5.3: Visual Explanation of differences in file formats

1. Based on Nasa Worldwind image and elevation data

2. A Java3D binary serialised elevation tile which is created from Nasa Worldwind image and elevation data

3. Four Java3D binary serialised elevation tiles, which are created from Nasa Worldwind image and elevation data, with LOD capabilities

The serialised Java3D file formats are created by using the original data from Nasa Worldwind and rendering them in the implemented testbed. When all data for a tile is collected and ready to be rendered on screen, they are stored to disk using the Java3D serialising mechanisms. The binary file formats used in the implemented testbed are stored using the inbuilt Java3D serialising mechanisms.

The texture data is limited to the Landsat7 image data set from the Nasa Worldwind server, and all texture data is downloaded from the NWW server in 512x512 pixels.

The data collection is an automated process that starts immediately as a test run is started. There are implemented mechanisms within the testbed to automatically start and stop the frame per second test data collection from the third party application Fraps.

5.3.1 File Sizes

The binary Java3D serialising stores all data uncompressed. Considering the amount of video texture memory an image can consume, the filesizes can grow large compared to the NWW original data. The expression

$$(512 * 512 * 3) * 1.333 = 1023.7$$

shows roughly in kilobytes how much graphics memory one texture in the implemented testbed will consume on disk with pixel size of 512x512 in the graphics card memory.

Since there is no need for transparency. The virtual terrain generates textures without alpha channels, making it necessary to only multiply with the Red, Green and Blue (RGB) colors, and not include the alpha channel (RGBA) when generating a texture. Being able only to multiplying with 3 (RGB) and not 4 (RGBA) decreases the use of video texture memory by approximately 1/4 in this case.

To increase the rendering speed and the level of detail, the software generates mip-map textures for each texture. This results a memory overhead factor of about 1.333. This shows that storing only one texture to disk, takes about 1 megabyte of space, in addition we are storing the geometry and group structure information.

It is important to note that the size of binary tiles will vary depending on which level that is used. This is because of the programmatic geometry Level of Detail simplification algorithm that is used within the software, the simplification algorithm is explained in section 4.1. Tiles with more detail, will take up more space on disk compared to tiles with less detail, however this is not the case for original file formats which have a constant size for height data of 44 kilobytes.

To assess the problem with large binary files, Java3D has added the possibility to compress textures to JPEG format by setting a property “j3d.io.ImageCompression” to JPEG. Compressing textures with lossy compression formats, such as JPEG, would not be a fair comparison, because we would not compare identical data when running the tests. By using lossy compression formats in terrain rendering software one would also run the risk of generation loss; repeatedly compressing

and decompressing files will cause files to progressively lose quality, however this would not be a problem in the implemented testbed since it is only storing data once.

Due to the large differences in file-size, and to create reliable comparisons of performance. The test are run both with and without textures in order to better understand the impact of large file sizes.

5.4 Environmental Needs

To create reliable comparisons of performance, a test sequence is executed on a single computer representing a typical end-user hardware configuration.

All necessary data needs to be pre-processed and loaded onto the machines hard-drive, so that during testing no connection to the Internet is required.

Although the implemented testbed runs on all platforms that supports the Java 5 platform with Java3D 1.6, the test is performed on a Windows XP computer with Service Pack 2 installed. Figure 5.4 shows the hardware configuration used for the tests.

Test tools

In order to accurately benchmark the applications frame per second performance, third party applications is used. The Fraps software is used to test FPS graphics performance under Windows.

Test pass/fail criteria

The Fraps software is used to monitor the frame per second during a test. An average frame rate above 15 fps on all tests is necessary to pass the test. Research has shown that users working in a low frame rate virtual environment substantially degrade their ability to perform tasks both in terms of accuracy and speed. M. Reddy [26] found in his research that with frame rate above 15 the rate of improvement in user performance is much less dramatic, and recommends in his work that a frame rate around 15 Hz should be taken as a minimum requirements for VR applications. It should be noted that higher frame rates will continue to improve performance and should be strived for in performance critical applications.

In addition a test-run that results in a complete set of valid output results is considered to pass the test. A successful completion of all test-runs is a requirement to pass a test session.

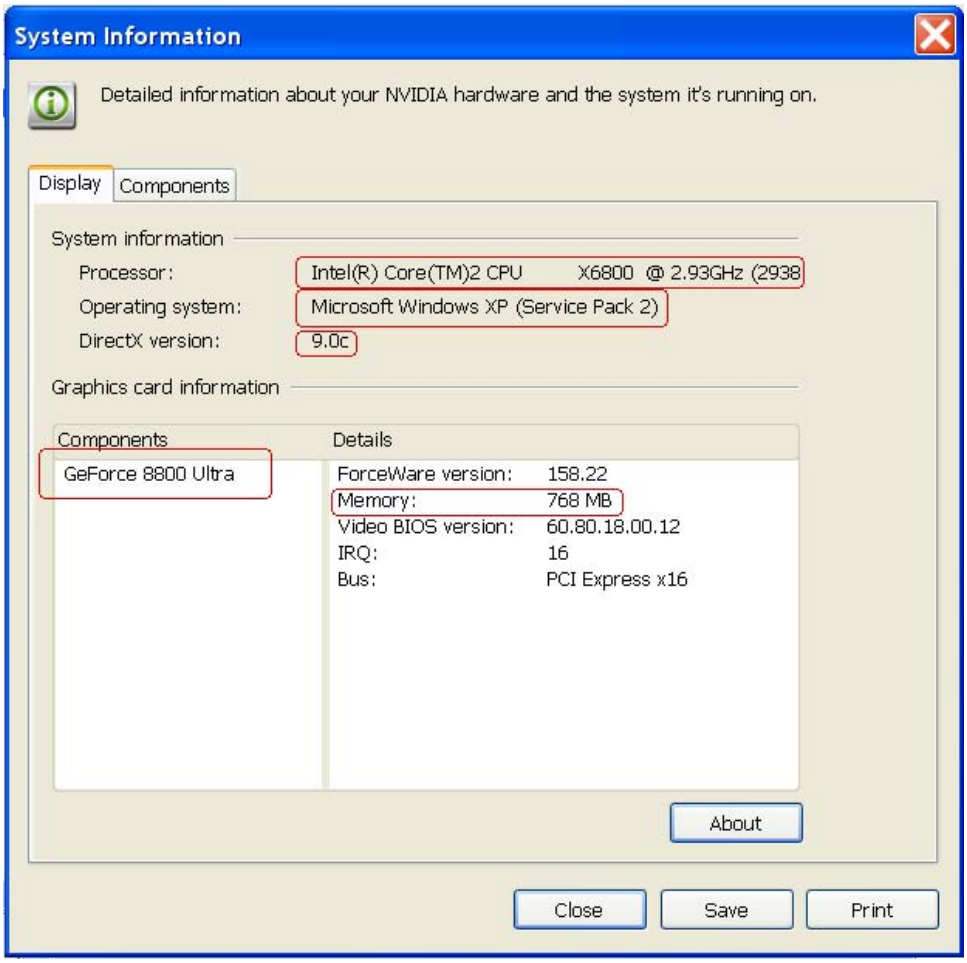


Figure 5.4: Hardware and software configuration used when running tests

5.5 Test Procedures

The testing tasks consist of 3 different tests that simulate different typical user operations performed in virtual terrain software. Each test is repeated 5 times for each file format. The test data are logged for each successful completion of a test and put in a table.

The application is shut down after a test is finished for a file format. Between each test of a file format, a reboot of the computer is performed to make sure that all memory is cleaned and that the tests are started with the computer in a equivalent state.

5.5.1 Testbed User Interface

The following test procedures are performed when running the different tests

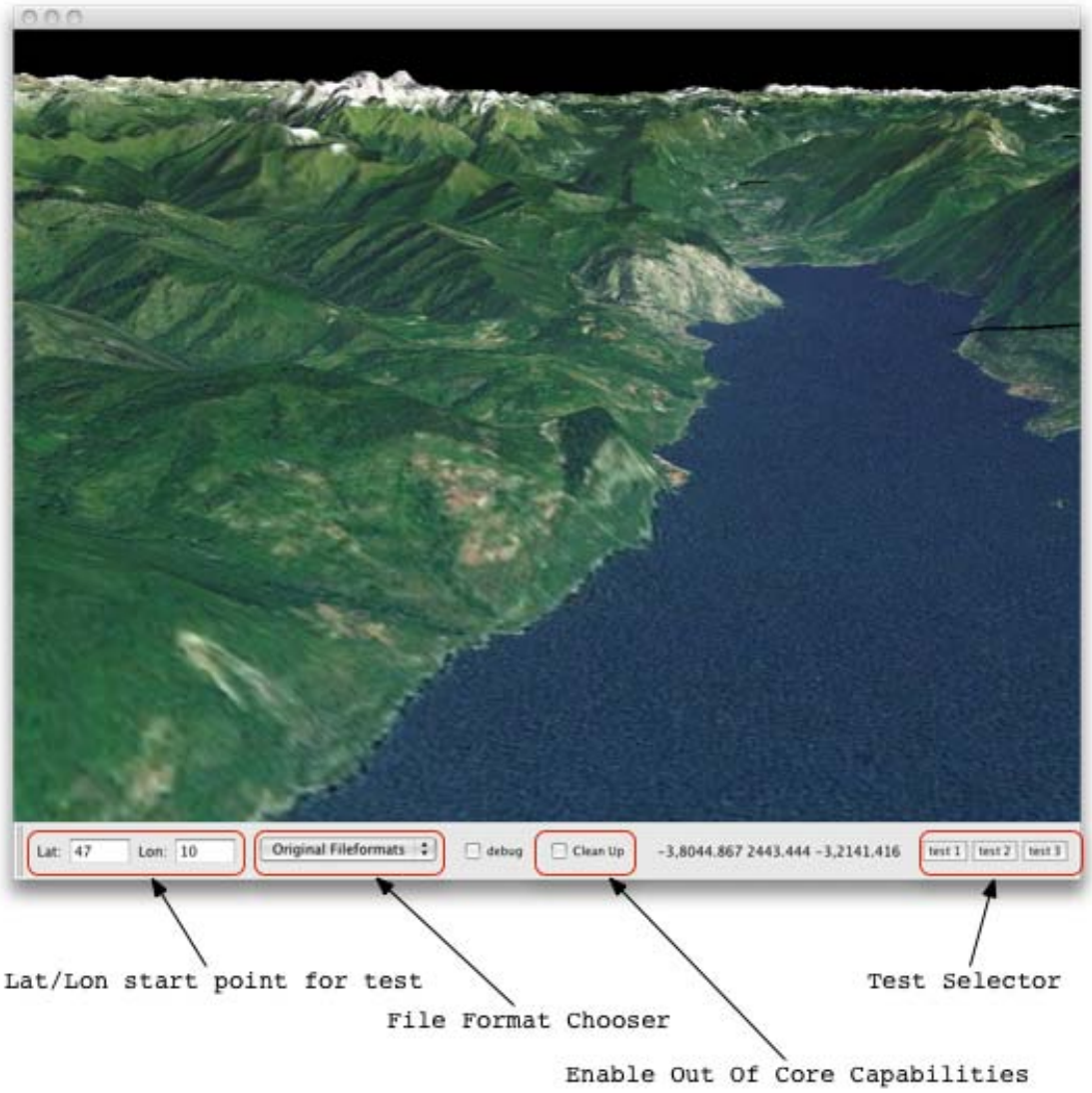


Figure 5.5: Overview of test-features on implemented testbed

1. Start the command prompt
2. Start the implemented testbed.
3. Start up FRAPS and make sure it has enabled the benchmarking capabilities.

4. Choose the desired file format to test
5. Type in the Latitude and Longitude used as geographic start-point for test, the latitude 60 and the longitude 10 is used consistently in all tests.
6. Press one of the test-buttons to perform the desired test (all tests are described in the next section.)
7. The test session starts, the implemented testbed collects all data during the test and the results are output when the test run is finished.

5.6 Test Descriptions

There are created 3 different test-cases with the purpose of simulating different typical user operations. All tests are defined using a metric, right-handed Cartesian coordinate system.

5.6.1 Test 1 - Quick Traversal

The purpose of this test is to simulate a user that zooms closer to the ground, then navigates quickly over a large area and back again. The purpose of the quick traversal test is to check the file format s ability to provide the rendering framework with high resolution tile-data when a user traverses quickly over a geographic area.

This predefined flightpath has defined 4 different destination points.

	Position	Travel Length	Time
Travel point 1	-1000.0 10000.0 -5000.0	190068.40m	2000 ms
Travel point 2	-500000. 10000.0 99	499026m	20000 ms
Travel point 3	50000.0 1000.0 -990	550074.70m	17000 ms
Travel point 3	-5000.0 30000.0 99	62186.70m	13000 ms

5.6.2 Test 2 - Zoom In and Out (and in again)

The purpose of this test is to simulate a user that zooms from a point high above the ground, closer to the ground and back again. This test has defined 3 predefined flightpath-positions that differ mostly in the y-direction.

	Position	Travel Length	Time
Travel point 1	-1000 800, 99	1284.4 m	7000 ms
Travel point 2	-1000 40000 -99	39200.5m	3000 ms
Travel point 3	-1000 400 99	39600.5m	1500 ms

5.6.3 Test 3 - Large Geographic Area Traversal

The purpose of this test is to simulate a user that traverses a terrain over a very large area. An important purpose for this test is to test differences in the out of core functionality between the different file formats. For this test, it is necessary to enable the out of core functionality in the implemented testbed.

The start travel-point moves the viewpoint fairly close to the ground to make sure we load high resolution data. The viewpoint is then moved fairly close to the ground over a large geographic area.

	Position	Travel Length	Time
Travel point 1	-99.0 5000.0 50000.0	201308.25 m	8000 ms
Travel point 2	99, 10000 -4000000.0	4050003.10 m	640000 ms

5.7 Test Results

All results that are shown in this section are averaged from all 5 test-runs. The average load times are calculated by the *average load times for all test runs on a specific load level divided by the average number of tiles loaded*. Output from all tests are added in Appendix E and F.

Note that level O and 1 are always loaded simultaneously, this can be considered as a very simple pre-fetching technique. If it is necessary to load a new tile at level 0, we are adding level 1 automatically, because there is a good chance that you are moving in that tiles direction. This is also implemented as a time-buffer, to give the framework a small amount of time until the next level is loaded. For these reasons LOD level 0 and 1 are merged in the test result data.

5.8 Test 1, Quick Traversal with textures

The table 5.1 shows that the average frame-rate during the test-run conforms. All test-runs are above the limit of 15 fps, and are fulfilling the test pass criteria.

Table 5.1: Average Frame Per Seconds

Original file formats	Java3D Binary tiles	Java3D Binary Quad-Tiles
258.2	1463.3	1171.2

We can see from Figure 5.6 that the average load times vary significantly, and that the original file formats load much faster, at all LOD-levels, compared to both Java3D binary file formats. The faster load time can be explained by the large differences in file size.

Figure 5.7 shows that the number of attempts correlates to the total number of quad-tiles loaded, the more attempts, the more quad-tiles are loaded. We can also see that the original file format has managed to create several more attempts compared to the other file formats. This indicates that the tiles are appearing quicker on the screen, and that the user is experiencing longer traversal times on each tile.

An interesting observation is the number of load attempts on level 5. The binary file formats have over 90 attempts to load level 5 tiles, but manage to render only 5. Because of the shortage of time, and hence the failing of loading, this implies that there can be a lag at all levels in the quad-tree making it hard to manage to load level 5 in time. This reason, together with the much higher load times, implies that the user has traversed for a smaller amount of time on each quad-tree tile.

These indications become even stronger when looking at the total number of tiles loaded in

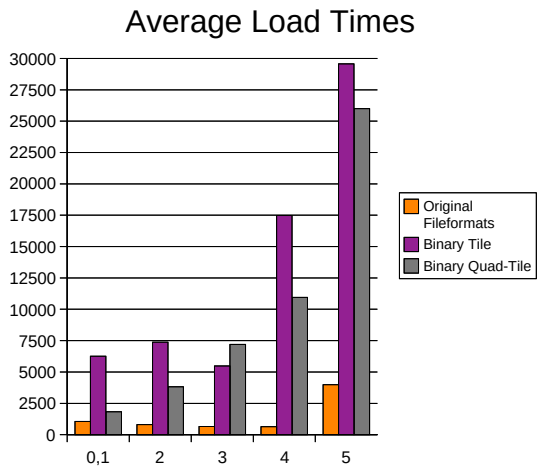


Figure 5.6: The Average Quad-Tile Load Time on Different file formats

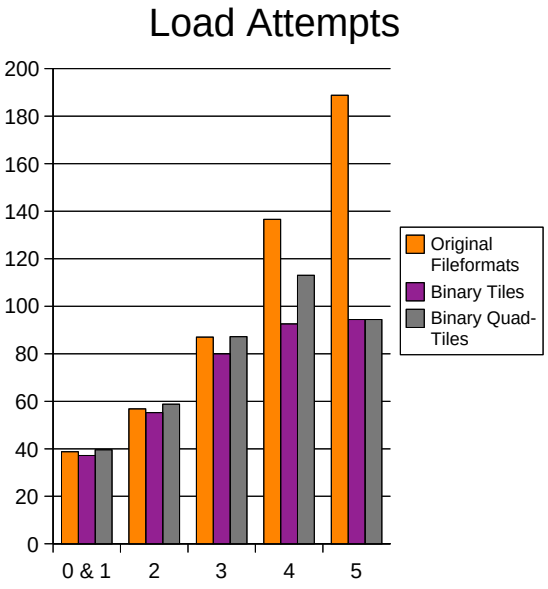


Figure 5.7: Quad Tile Load Attempts Test 1

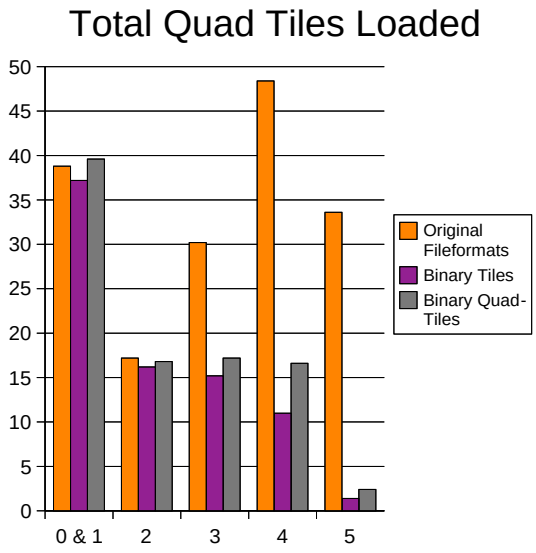


Figure 5.8: Total Loaded Quad Tiles Loaded Test 1

Figure 5.8. The total amount of tiles loaded on all levels correlates directly to the loading time. A smaller loading time enables the implemented testbed to load more tiles.

5.8.1 Test 1, Quick Traversal without textures

When running the same test without textures we can see that the average frame rate increases. This is probably due to less computational cost needed to render a texture. The biggest increase in frame rate is with the original file format. This can be due to the fact that the implemented testbed only need to access the disk once when only visualising geometry, compared to twice when using textures in addition.

Table 5.2: Average Frame Per Seconds

Original file formats	Java3D Binary tiles	Java3D Binary Quad-Tiles
777.3	1512.0	1445.1

When not rendering the textures, we can also see that the average load times for the different formats decreases considerably for all levels. The reason for this can be due to the smaller file-sizes, because we are not including the texture information in the Java3D binary serialised files.

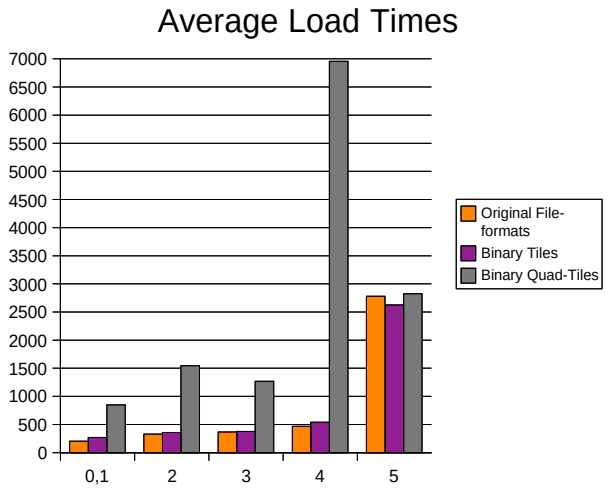


Figure 5.9: The average load time on geometry quad tiles during geometry test 1

When running the tests without the use of textures we can see that there is less difference in the test results between the original file formats and the Java3D binary serialised tiles, compared to running the test with textures. An interesting observation is that the Java3D serialised binary tile format performs slightly better than original file formats on level 5 on both total processed tiles(Figure 5.10) and average load time (Figure 5.9). This indicates that pre-proccesed tiles can

have performance effect on higher resolution data due to the lesser computational cost. Since there are lesser computation cost on lower quadtree LOD-levels, this indicates that it is more effective to use Java3D serialised tiles for higher resolution data, because that we are able to put pre-prossed data directly into the Java3D scene-graph.

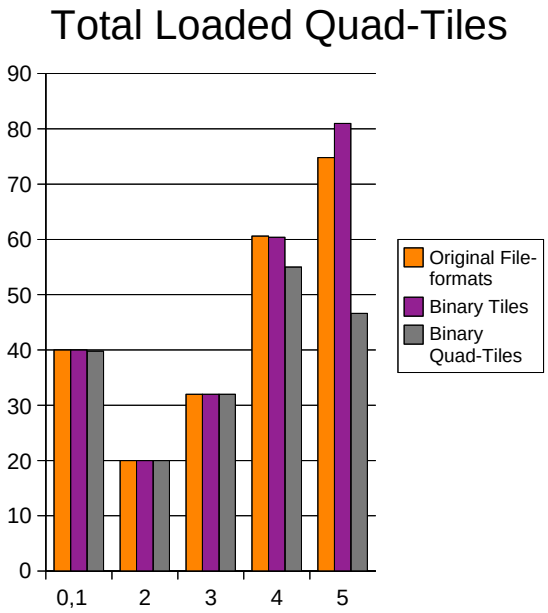


Figure 5.10: The total processed geometry quad tiles during geometry test 1

In addition to be able to load more tiles, we can also see an almost doubling of frame-rate on both Java3D binary formats. This indicate that these formats require less processing on the client side.

The Binary Quad-tiles format is performing inferior to the other formats even though it should require less processing on the the client side compared to the binary tile format. From figure 5.9, we can see that the load times are much higher on level 0-4 when using the Java3D binary Quadtree-tiles. This has probably reduced this file format s ability to feed the testbed with as much data as the other file formats. The average load time on level 5 are fairly equal, which shows that it is possible to load geometry with higher density with higher frame-rate when using Java3D binary file formats.

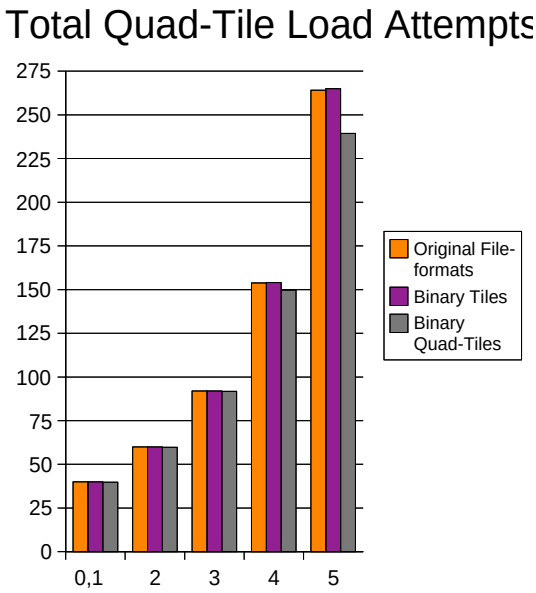


Figure 5.11: The total geometry quad tiles load attempts during geometry test 1

5.9 Zoom in and out test - Test 2 with textures

The table below shows that this test has generally higher or equal frame-rates compared to Test 1. This is probably due to less tile data and LOD levels are processed for this test.

Table 5.3: Average Frame Per Seconds

Original file formats	Java3D Binary tiles	Java3D Binary Quad-Tiles
1598.1	1623.9	1648.8

The average load times are surprisingly showing a different pattern than test 1, quick traversal. The highest average times are spent on tiles with least data. A partial explanation for this, is that its can be due to a performance hit on Java3D binary tiles on first run after reboot. This feature will be further outlined in chapter 6.

One interesting observation is that Java3D tiles are performing slightly better on high-resolution tiles, and inferior on level 0-4. This undermines the findings in test 1, that Java3D binary files, can have a positive performance effect on higher resolution data.

As we can see on figure 5.13 the amount of processed quad-tiles does not vary greatly, but the slight difference in total number of processed tiles could also partly be explained by the performance

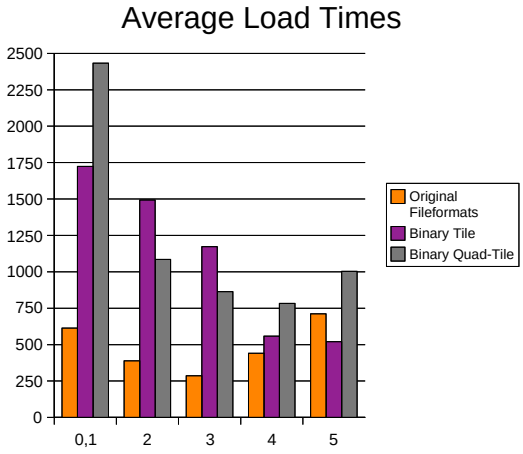


Figure 5.12: The average load times on quad-tiles during test 2

hit on first run after reboot, which was the only run on both binary file formats that did not manage to load the equal amount of data on the highest resolution tiles.

The total number of load attempts are equal for all the different file format. This can be due to the small amount of data loaded into the testbed, and that the testbed is able to load all data that is asked for.

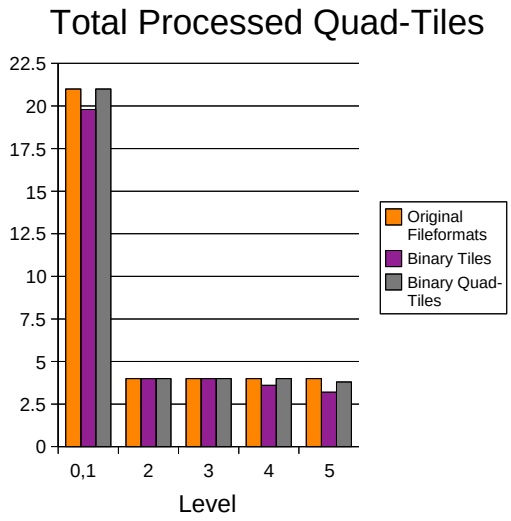


Figure 5.13: The total processed geometry quad tiles during test 2

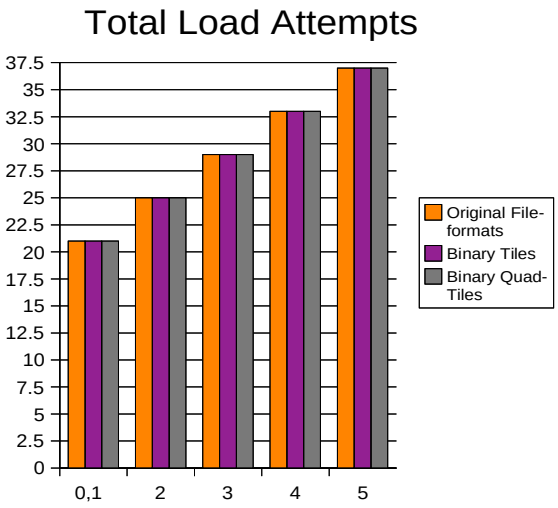


Figure 5.14: The total load attempts

5.9.1 Zoom in and out test - geometry testing

Table 5.4: Average Frame Per Seconds

Original file formats	Java3D Binary tiles	Java3D Binary Quad-Tiles
1923.0	1957.0	1929.7

The total number of loaded quad-tiles are the same for all formats, shown in Figure 5.18, meaning that the testbed is able to deliver an equal amount of data for each file format for this particular test.

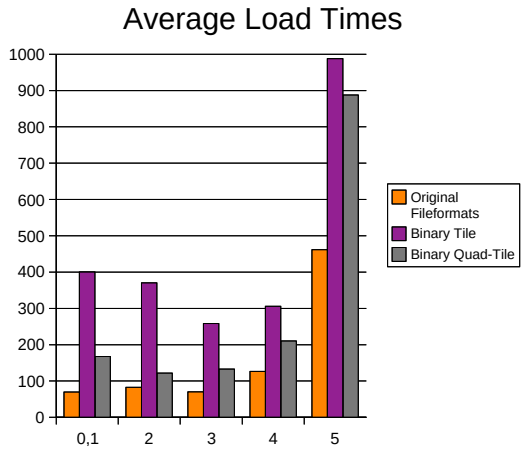


Figure 5.15: The average load times on geometry loading - test 2

As we can see from figure 5.17, the number of load attempts are also equal for all file formats, indicating that all attempted geometry for the viewpoint animation was loaded, and that there is no significant difference in the quality of the output to the user. Having in mind that this test moves the viewpoint over the same geographic area twice, there is a chance that high resolution tiles were added the second time the viewpoint moved over the same area.

Load times are longer for the binary formats on high resolution tiles, compared to test 1. This can be due to the smaller amount of processed data, and hence less computational cost in general for this test. The total amount of loaded tiles on level 2-5 is only 4.

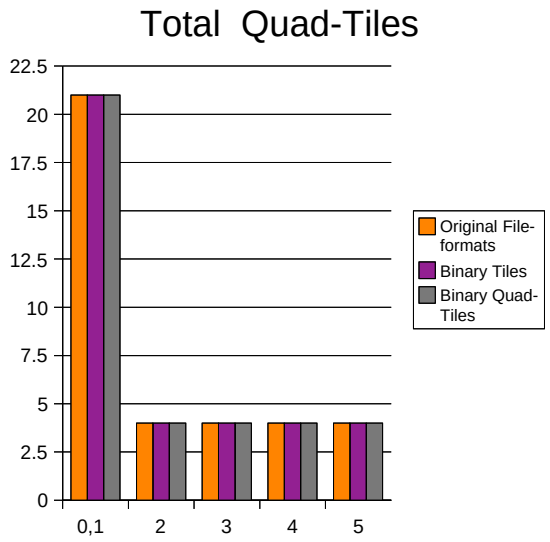


Figure 5.16: The total processed quad tiles during test 2

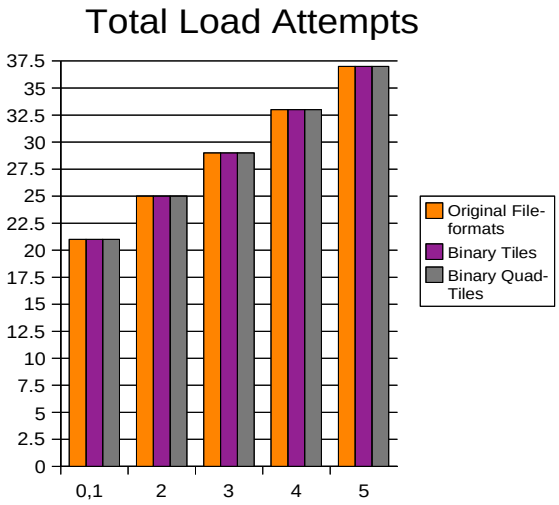


Figure 5.17: The total load attempts

5.10 Test 3 - Large Geographic area traversal

The purpose of this test is to investigate the testbed s rendering performance over a longer period of time, and to investigate the differences in the different file formats ability to handle a large amount of terrain data. This is done by simulating a user that is traversing a virtual terrain for over 12 minutes. For this test, the out of core capabilities in the software must be turned on.

As we can see from table 5.5, the frame per seconds vary. The Java3D binary files are performing better compared to the original file formats. This is probably due to the less number of tiles rendered on screen with these two formats.

Table 5.5: Average Frame Per Seconds

Original file formats	Java3D Binary tiles	Java3D Binary Quad-Tiles
1048.2	1752.9	1446.0

We can see from figure 5.18 that both Java3D binary formats are performing inferior to the original file formats, and are not able to deliver the same amount of data to the testbed. The total processed quad tiles correlates to the average load times, shown in Figure 5.17, meaning that the lowest average load-times are capable of loading more data.

The original file formats are not only able to provide the testbed with more data, but are also capable of creating more attempts, especially on higher resolution data. This indicates that the user has had longer traversal time on each tile.

During this test, it is necessary for the testbed to page data from memory to disk. Considering the large differences in file-sizes between the different file formats, this can have an impact in performance. Writing to as hard-drive that is already under heavy load can decrease the overall performance.

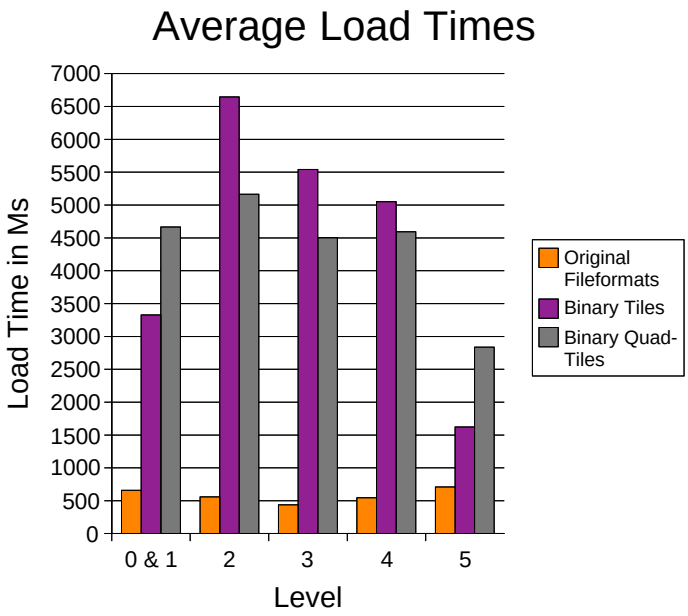


Figure 5.18: The average load times during test 3

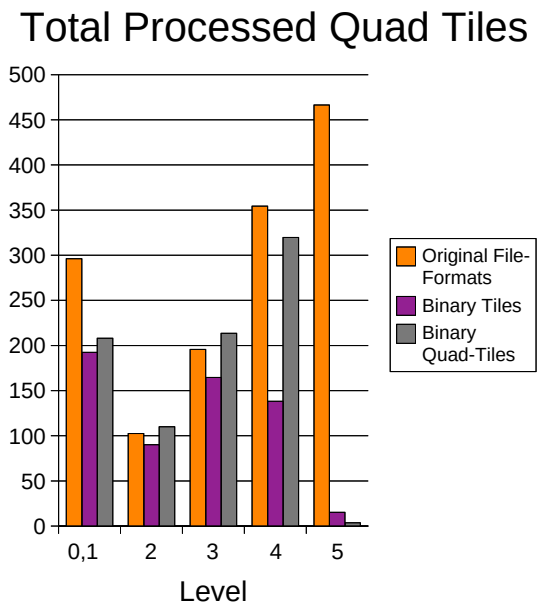


Figure 5.19: The total processed quad tiles during test 3

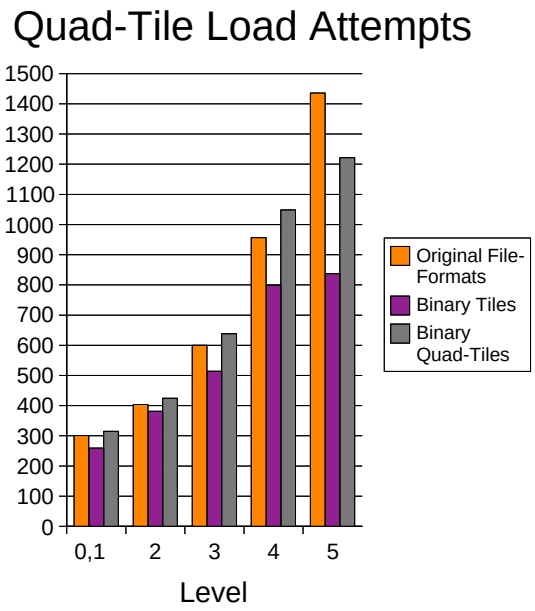


Figure 5.20: The total processed quad tiles load attempts during test 3

5.10.1 Large Geographic area traversal - geometry testing

When running the same tests without textures, the frame rate is more equal between formats, however the same pattern as the tests with textures indicates that both the Java3D binary formats are performing inferior to the original file formats.

Table 5.6: Average Frame Per Seconds

Original file formats	Java3D Binary tiles	Java3D Binary Quad-Tiles
1429.6	1664.7	1659.1

The average load time differs greatly, where the Java3D binary formats are performing poorer than the original file formats. The average load time corresponds to the number of total Quad-tiles loaded, so the big differences in results can explained by the testbed s need to page memory to an already overloaded disk.

The fairly equal number of load attempts can be explained by this test is moving the viewpoint slower compared to the other tests, giving enough time for lower resolution tiles to load. Even if the number of attempts are fairly equal, we can see that there is a significant difference in the ability to deliver high-resolution tiles to the rendering framework.

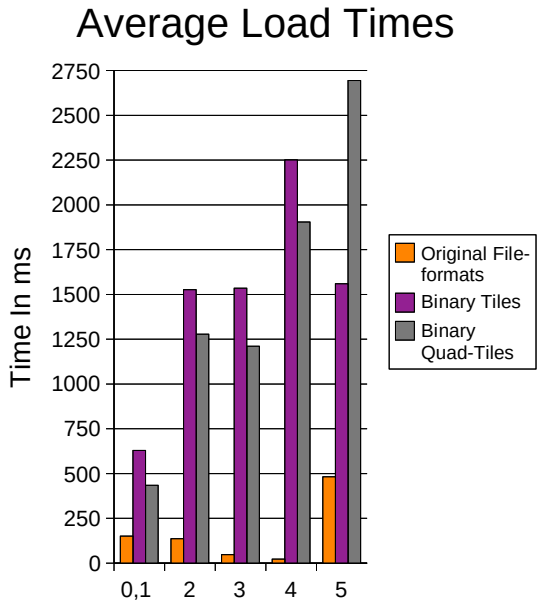


Figure 5.21: The average load time on geometry tiles during test 3

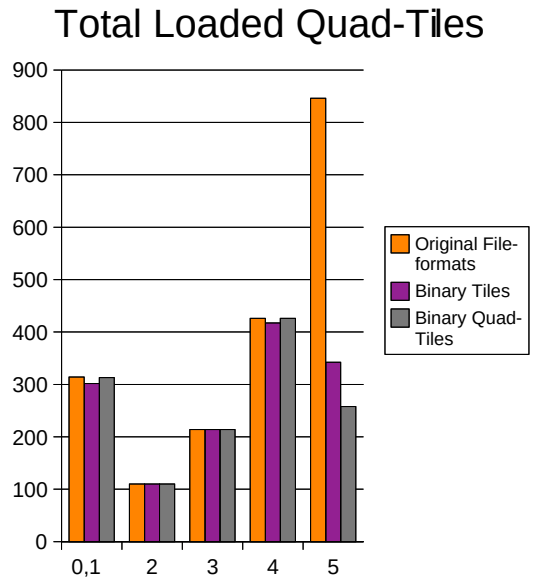


Figure 5.22: The total processed geometry quad tiles during test 3

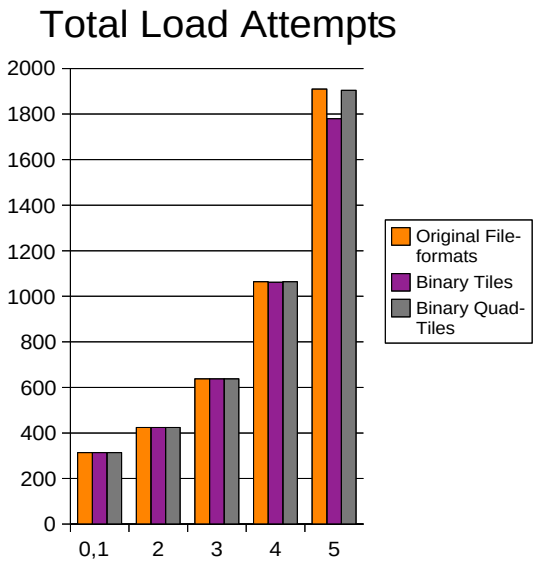


Figure 5.23: The total processed geometry quad tiles during test 3

5.11 Findings

The testing shows that the use of “real-life” test-runs has produced useful data. The results for each test-case differ greatly .

During test 1, the testbed s ability to handle data under high pressure was tested. When using Java3D binary tiles without textures, the test shows that this can be a good strategy when handling higher resolution geometry sets. All data for this test was handled in-core.

Another interesting observation that indicates that Java3D binary files can perform better compared to original file formats, is when performing a visual inspection of the test runs for the Binary Tile format. This shows that the 4 last test-runs on the Java3D binary tile format load quicker compared to original file formats and binary quad-tile. The first test run weakens the average load time excessively for LOD level 5, this can mean that the testbed has not loaded this LOD level at the first flightpath animation, and added it on the return, hence making the load time for that run very long.

During test 3 the Java3D binary file formats performed worst on all criteria. This test investigated the file format s ability to handle large, out-of-core, datasets. The inferior results for this test, indicate that when using larger files, the paging of memory to disk can decrease the overall software performance of a virtual terrain application. This indication is supported by the even larger differences in results for test 3 with textures, which have even larger files.

5.12 Summary

The testing has shown that it has been very useful to have a testbed consisting of many of the vital parts of a virtual terrain application, because it has been possible to extract and differentiate the test-results better. By running more isolated tests, we would not have been able to look at the totality of a virtual terrain application, and would have been produce data with the same quality. An example of this can be the differences in software performance when running tests with in-core datasets compared to out-of-core datasets. When running the out-of-core test, the Java3D binary file formats performed inferior on all criteria, while the Java3D binary file formats performed better on high-resolution data in the in-core test 1.

To further answer and investigate the research question “What is the overall impact on performance on multiresolution virtual terrains when utilising Java3D serialised binary files compared to utilising files from a Nasa Worldwind Server?” Further tests was undertaken to investigate the indication that Java3D binary tiles can perform better on higher resolution data.

Chapter 6

Discussion and clarifying tests

In this chapter, we review the test results and findings in relation to the research objective, and also gives some suggestions on the situations in which the different file-formats should be used.

The research objectives to be answered were as following:

- **What is the overall impact on performance on multiresolution virtual terrains when utilising Java3D serialized binary les compared to utilising les from a Nasa Worldwind Server**

6.1 The suitability of Java3D binary le formats for terrain rendering software

As explained in section 5.3.1, the serialized Java3D binary files can grow large in size. On a textured tile, the texture alone will result in an increase in file size of approximately 1 megabyte in size when using 512 x 512 pixel sized images for a Java3D binary tile. This makes the Java3D binary format unsuitable for transferring over the Internet, compared to the smaller sized existing file formats. However it is important to note that it is not in this context this testing has been performed. The Java3D binary files have been serialized after the NWW files have been added to the Java3D rendering framework, and the software performance effects have been analysed when loading data from a local disk.

6.2 The impact of large le sizes on terrain rendering software

The testing described in chapter 5, shows that textured Java3D binary files are not able to provide the rendering framework with as many quad-tiles as the original file-formats in a given timeframe. The situation is especially bad when using binary quad tiles in out-of-core situations. Knowing that Java3D binary quad tiles take up approximately 4 times more space on disk, and perform inferior to the other formats, the large file sizes was suspected to be one of the key factors for the poor performance.

In response to this, and in order to investigate and understand the software performance effects of using Java3D binary files better, a new large geographical area travel test (test 3) was run together with the performance monitoring application Perfmon. This specific test was made because of the significant differences in software performance on this test. The diversity was suspected to be due to the necessity of using out-of-core capabilities. When running a new test, a clearer, better explanation for the performance difference was hoped for.

The same test routines, as described in section 5.5, were performed but with Perfmon running simultaneously. The Perfmon application was setup to monitor the distribution in percent between processor time, disk activity time and disk idle time. This distribution between hardware resources was hoped to give us a clearer understanding of the diversity of results, and the distribution between hardware components when running the testbed. The output from Perfmon from test 3 using original file formats are shown in Figure 6.1

- The blue line shows the disk activity time in percent
- The red line shows the processor time in percent
- The yellow line shows the disk idle time

From figure 6.1 we can see that most of the time, the disk activity is below 15% while the processor time is mostly above 80%. Due to the low disk activity, the disk idle time is very high, emphazising the fact that the disk does not need to work hard to feed the rendering framework with data. This test shows that when using the original file formats, the rendering framework has no problems with getting the data it needs from the disk. When using the original file formats the rendering is CPU-bound, meaning that the bottleneck in the rendering framework is the capacity of the CPU and not the disk.

Figure 6.2 shows the Perfmon output from the same test with Java3D binary tiles. During the same test, the disk activity is always 100%. This clearly shows that when using Java3D binary tiles,

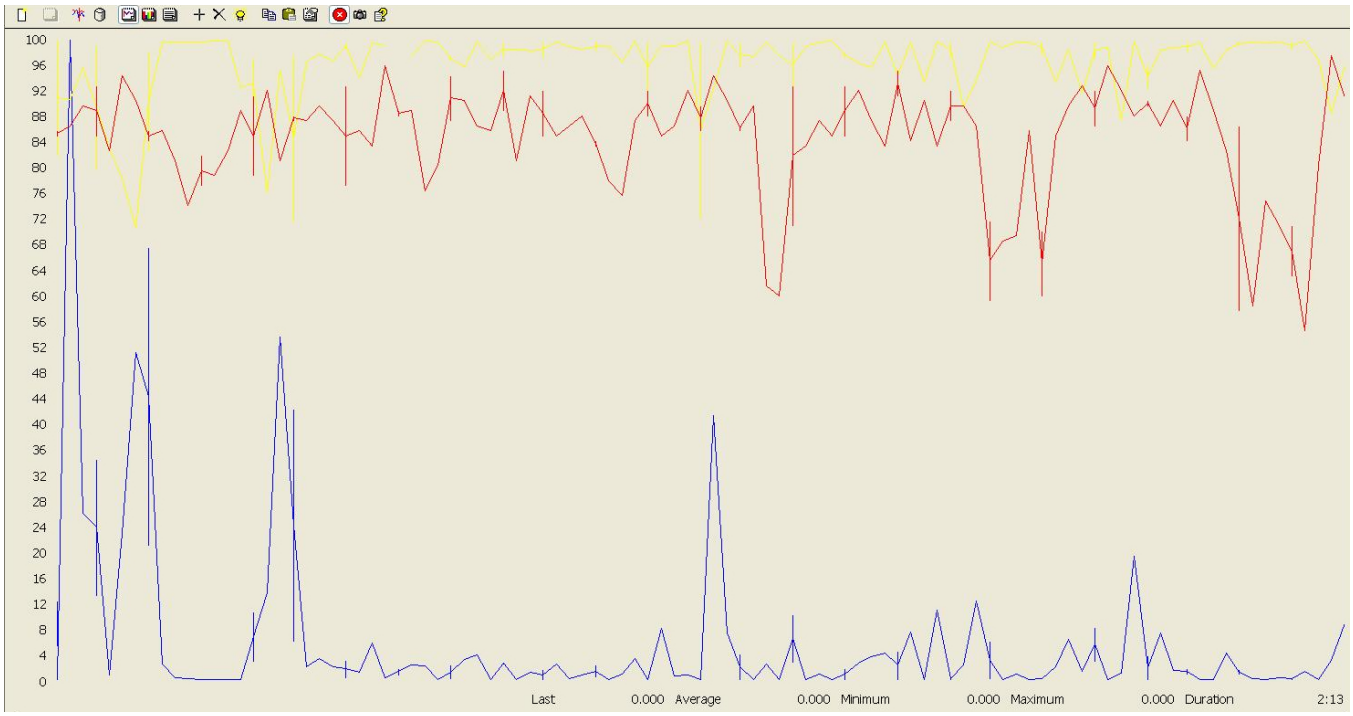


Figure 6.1: The distribution of resources on original file-formats

Figure 6.2: The distribution of resources on binary tiles

the software becomes disk-bound. The ability to render tiles is limited by the capability of the disk to provide the rendering framework with data. This can be one of the reasons why both binary file formats performed so much inferior in the large geographical test (test 3), as we are loading more data from disk into the rendering framework on this test compared to the previous two tests. The same trend was shown when running the test with binary quad tile file-format.

Having a situation where a hardware resource is tied up 100% is seldom desirable. Having no available resources from a hardware device makes the software vulnerable to impact from other processes or applications. Knowing that test 3 is the only test where the out-of-core functionality was used, can explain the much poorer result from Java3D binary formats for this test. When loading so much data in and out of memory as done with textured geometry tiles over a period of time like test 3, a situation where the Java Virtual Machine needs to swap some portion of the JVM heap to disk is likely to occur. A disk-write in a situation where the disk is already running at 100% can have a severe impact on system performance. In this case it would have created a latency in the entire chain of loading files from disk to rendering the data on screen.

6.3 Diversity in Test Results

During the analysis of test result data, there was especially one interesting finding concerning the Java3D Binary file formats, namely the better results on geometry loading during test 1 on higher resolution tiles (level 5).

To further investigate the reason for the boost in performance during geometry test 1, the test data was thourougly investigated for the different test runs. One can see a performance hit on the first run compared to the 4 following tests on test 1 and test 2. In most cases the first run is inferior to the latter 4 test runs. This is shown in Figure 6.3

Due to the performance hit on the first test-run, the actual difference in results is even larger. The Java3D binary file formats have performed better than average in 4 out of 5 runs, supporting our findings even stronger.

To give a precise explanation for the diversity between test-runs is not straightforward, but the fact that the test-machine was rebooted between each file-format test, can indicate that some sort of caching mechanisms are running behind the scenes. Most modern operating systems (OS) have inbuilt file caching mechanisms. These caching mechanisms are not only caching files physically [21], but are in addition storing large file pointers to recently used files. The purpose of the file-pointer cache is to reduce the seek time, and hence make the total delay of reading/writing data to

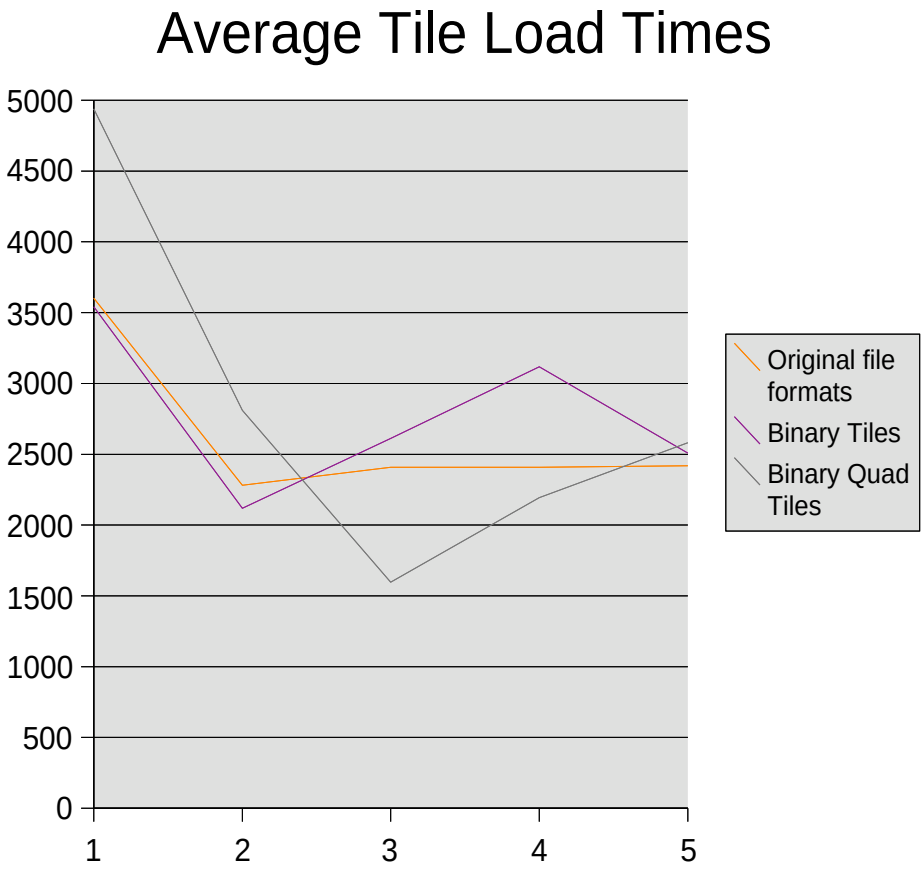


Figure 6.3: The average load times on tiles for all 5 test-runs on level 5 tiles during test 1

disk smaller.

6.4 High resolution geometry test

A final test was run to investigate why Java3D binary tiles perform better compared to original file formats on level 5 tiles. The chosen test for this purpose was test 1, since this tests the testbed s ability to deliver high resolution data, at high travel speeds, in non out-of-core mode. For reasons explained in 6.2, the out-of-core test was omitted.

This test was run without the programmatic LOD feature explained in section 5.2, hence making the tiles on every LOD level the same resolution as the high resolution level 5, 150 x 150. The test is only run once, but with a reboot between each test run. The framerate from the tests are output in the tables 6.1 and 6.2.

Table 6.1: Output from fraps on Original File Format 150 x 150 test

Frames	Time (ms)	Min	Max	Avg
10329	57230	0	1268	180.482

Table 6.2: Output from fraps on Binary Tiles 150 x 150 test

Frames	Time (ms)	Min	Max	Avg
34298	54645	1	1463	627.651

From the output from the Fraps application, we can see that the binary tiles have a significantly better average frame rate. We can see from table 6.1, that the original file-format has an average of 180.5, and from Table 6.2, that the binary tiles averaged 627.6. This is a total difference of 247.8% in frame-rate increase.

Table 6.3: Original File Format 150 x 150 test

Level	Total Tiles Procesed	Total Quad-tile Attempts	Acc. Time used on Quad-Tile in ms
0,1	38	38	146829
2	18	56	59018
3	28	86	162150
4	11	128	96767
5	2	135	906

Table 6.4: Binary Tile 150 x 150 test

Level	Total Tiles Procesed	Total Quad-tile Attempts	Acc. Time used on Quad-Tile in ms
0,1	33	33	73445
2	16	49	36277
3	22	75	8781
4	35	111	304848
5	17	150	36420

From the tables 6.3 and 6.4, that show the output from the testbed, we can see that when using the Java3D binary tile format, we are able to render a higher number of tiles compared to original file-formats. The original file formats managed to load a total of 97 tiles, and the binary tiles file format managed to load a total of 123 tiles. This is a difference of approximate 26.8% in performance.

This tells us that using binary file formats on regularly gridded tiles with high density can give us a performance gain, both in frame rate and in the capability to load non out-of-core data into the rendering framework.

Chapter 7

Future Research and Conclusion

In future work, looking into more efficient ways of representing binary data for virtual terrains, as well as finding ways to reduce the disk activity when using binary formats should be central.

7.1 Future Research and recommendations

The X3D binary encoding work [1] would be of special interest. A draft X3D Binary Encoding specification was approved by the Web3D Consortium [1], and was later approved for advancement to International Standard, ISO 19775. Sun Microsystems Inc. has provided royalty-free contribution of several advanced compression technologies that will help X3D to be a more compact and efficient 3D format. Some of this royalty free contributions is derived from the Java3D geometric compression algorithms.

Both lossy and lossless compression is possible with this approach. According to the specification the lossless compression should make it possible to maintain an identical dataset, compared to original fileformats, and compress the files in size. This attribute is desirable in 3D software that requires a high rendering throughput such as virtual terrain software.

In the specification, two basic kinds of size-reduction techniques are used in combination.: information-theoretic compression and geometry based compression. The information-theoretic aims to minimise duplication of data, similar to the zip and gzip compression format. The geometry compression compresses, rearranges, and combines polygons, colors, interpolators and so on to reduce the geometry in size.

[1] http://www.web3d.org/x3d/binary/

According to the X3D: Extensible 3D Graphics for Web Authors book [8] typical file-size reductions range from 10-25% of the original, consistently beating gzip reduction. Parsing speed when reading data at run-time, is typically, run 5-10 times faster. Of course this can make a significant improvement for large virtual terrain software.

To test the X3D binary compression format, one could utilise the Xj3D[2] software. Xj3D is an open source (LGPL) project of the Web3D Consortium Source Working Group focused on creating a toolkit for VRML97 and X3D content written completely in Java. It serves as a testbed for testing the X3D specification. The implementation of Xj3D supports several different renderers, OpenGL[3], Aviatrix[4], Java3D and a mobile renderer. Since the Xj3D software supports the Java3D renderer, it should be possible to test the X3D binary compression format directly in the software implemented for this thesis.

7.1.1 Memory Caching

One way to reduce the disk activity for the implemented prototype, is to introduce a memory cache. To store the binary data in memory that is likely to be used again, will prevent the software from re-fetching the same data from disk several times, and hence reduce the disc activity.

In the paper "modeling the digital earth" [4] a LRU (Least Recently Used)caching algorithm is used. Considering the similarities between the implemented prototype and the geovrml implementation, this seems like a viable option for further work. LRU caching discards the least recently used items first, keeping only the frequently used items in memory.

7.1.2 Prefetching

Another optimization technique that can potentially increase the rendering throughtput of virtual terrain software, is predictive pre-fetching. By predicting future values of the viewpoint movement, and loading them into the memeory cache, one can reduce the latency of loading tile data, and thus improve rendering throughput. Although this technique is especially interesting for implementations that loads tiles over a network, it can be used in the implemented prototype to perform predictive pre-fetching of tiles into a memory cache for quicker access.

The Terravision software utilises predictive tile prefetching based on a linear extrapolation of the users viewpoint, so that tiles are immediately available for rendering [18].

[2] http://www.xj3d.org
[3] http://www.opengl.org
[4] http://aviatrix.j3d.org

7.2 Conclusion

Much of the work on this thesis has presented ways of building a testbed for studying the performance impact of Java3D binary serialized files on multiresolution terrains. In the initiation and preparation phase, the most fundamental concepts and algorithms for generating large virtual terrain models were studied. Based on the output from this phase, a virtual terrain testbed was implemented. The testbed was implemented in order to be able to study, and understand, the performance impact of Java3D binary serialized files by looking at the totality of a virtual terrain rendering system. Being able to study the entirety was an advantage when running the tests, as it made it possible to retrieve more information from the different test runs than tests of more limited scope. By running more isolated tests, and not look at the totality, one would running the risk of not retrieving the same amount of information, hence weakening the findings.

In 2001, R. Aasgaard and T. Sevalrud [3] suggested to replace the Java platform with C++ due to the limitations of performing optimisations in the Java Language in order to achieve a higher rendering throughput. Due to the rapid advance in graphics hardware, and the utilisation of a new level-of-detail structure for rendering terrains. F. Losasso, H. Hoppe [16] claims in 2004, that the rendering throughput has reached a level that enables a novel approach to level-of-detail (LOD) control in terrain rendering.

The work with this thesis has shown that in 2009, it is technically possible to achieve a high rendering throughput, with out of core functionality, by using a novel approach to LOD control with a high level graphics API like Java3D. The results from the testing also shows that it is technically possible to use Java3D serialized branches as a file format for terrain rendering, but it is not always the best choice. When using Java3D binary file-formats on a textured terrain, the software performance is always poorer compared to the original file-formats.

When testing the implemented testbed without textures and using only the geometry. The testing shows that using Java3D Binary tiles is an advantage only when having regular gridded tiles at higher resolutions in non out-of-core mode. For the computer used for this testing, when looking totality of the implemented prototype, the Java3D binary geometry tiles performed inferior on lower resolution tiles. This might be due to the cost of processing data being too small on data-sets of this size, as serializing is only efficient for more complex or higher resolution geometry. When running the geometry tiles test with resolution at 150 x 150 at all LOD levels, the geometry tiles managed to increase the number of loaded tiles by 26.8% in addition to increase the frame rates with 247.8%.

The increased frame rate and computational power when using Java3D binary tiles, may be

especially important when the software is integrated into the existing software platform developed at the Institute for Energy Technology for scenario management and simulation combined with hazard visualisation. This software framework is using complex real-time calculations, restricting the terrain visualisation module to run with limited hardware resources.

References

[1] X3DBinary, Extensible 3D (X3D) encodings ISO/IEC 19776-3:2007. URL: `http://www.web3d.org/x3d/specifications/ISO-IEC-19776-3-X3DEncodings-CompressedBinaryEncoding/`.

[2] Louka M. N., Nystad E., Gustavsen M. A. and Olsen A. CollabVE: A Research and Development Platform for Collaborative Virtual Environments . *HWR-850* , 2007.

[3] R. Aasgaard and T. Sevalrud. Distributed Handling of Level of Detail Surfaces with Binary Triangle Trees. *ScanGIS 2001*, 2001.

[4] Martin Reddy, Yvan G. Leclerc, Lee Iverson, Nat Bletter and Kiril Vidimce. Modeling the Digital Earth in VRML . *http://www.ai.sri.com/ reddy/pubs/pdf/aipr99.pdf*, 1999.

[5] Y. Leclerc, M. Reddy, M. Eriksen, J. Brecht and D. Collee. SRI s digital earth project, Technical Note No. 560, 2002. URL: `www.ai.sri.com/pubs/files/908.pdf`.

[6] B Hallen D. Loubke. Perceptually Driven Interactive Rendering. *Technical Report CS-2001-01*, Mar 2001.

[7] A Varshney B Watson. R. Huebner. D. Luebke M. Reddy J.D Cohen. *Level Of Detail for 3D Graphics*, chapter .1 Introduction, pages 3–5. Morgan Kaufmann Publishers, 2003.

[8] D. Brutzman; L Daly. *X3D: Extensible 3D Graphics for Web Authors*, chapter .2 Technical Overview, pages 29–30. The Morgan Kaufmann Series in Interactive 3D Technology, 2007.

[9] E. Nystad, A. Drivoldsmo and A. Sebok. Use of Radiation Maps in a Virtual Training Environment for NPP Field Operations. *HWR-681, Halden, Norway*, 2002.

[10] Nystad E. Training of Safety Practices in a Virtual Environment. *Presented at OECD EHPG Meeting 1999, Loen*, 2007.

[11] Nystad E. and Strand S. Using virtual reality technology to include field operators in simulation and training. *Paper presented at the Canadian Nucelar Society 27th Annual Conference*, June 11-14 2006 2006.

[12] Rindahl G., Johnsen T,. Mark N-Kr. F. and Meyer G. Virtual Reality in Planning and Operations from Research Topic to Practical Issue . *In Proceedings of 5th International Topical Meeting on NPIC, and HMI Technology at the ANS 2006 Meeting*, 2006.

[13] Rindahl G. Strategies for Handling Large Volumetric Datasets to Support Work Planning Activites in High-Risk Environments. *HWR-859*, 2007.

[14] Rindahl G. and Mark N.K.M. VRdose and emerging 3D software solutions to support decommissioning activities. Experiences and expectations from development and deployment of innovative technology . *in International Atomic Energy Agency, Innovative and Adaptive Technologies in Decommissioning of Nuclear Facilities, Results of a Coordinated Research Project 2004-2008 IAEA TECDOC-1602 October 2008*, 2008.

[15] M Granlund. Perspective Based Level of Detail Management of Topographical Data. Master s thesis, Østfold University College, 20024.

[16] F. Losasso, H. Hoppe. Geometry clipmaps: Terrain rendering using nested regular grids. *ACM Transactions on Graphics (Proc. SIGGRAPH 2004), 23(3), 769-776.*, 2004.

[17] D. Leubke, M.Reddy J.D., Cogen A., Varshney B., Watson R Huebner. *Level of Detail for 3D Graphics*. Morgan Kaufman Publisher, San Francisco, 2003.

[18] M. Reddy, Y. G. Leclerc, L. Iverson and N. Bletter. TerraVision II: Visualizing Massive Terrain Databases in VRML. *EEE Computer Graphics and Applications*, 1999.

[19] Craig Larman and Victor R. Basili. Iterative and Incremental Development: A Brief History, 2003. URL: `www.highproductivity.org/r6047.pdf/`.

[20] Y. G. Leclerc and S. Q. Lau. TerraVision: A Terrain Visualization System. *AIC Technical Note 540*, Mar 1994.

[21] Odysseas Pentakalos Mark Friedman. *Windows 2000 Performance Guide*, chapter 7. File Caching (Windows). 2002. URL: `http://technet.microsoft.com/en-us/library/bb742613.aspx`.

[22] G. Misund and M. Granlund. Global models and the w3ds specification - challenges and solutions. In *In First International Workshop on Next Generation 3D City Models*, 2005.

[23] Louka M. N. A Software Toolkit for implementing low-cost virtual reality training systems. *Presented at OECD EHPG Meeting 1999, Loen*, 1999.

[24] Louka M. N. and Balducelli C. Virtual Reality Tools for Emergency Operation Support and Training. *In Proceedings of the International Conference on Emergency Management Towards Co-operation and Global Harmonization (TIEMS 2001). Oslo*, 2001.

[25] M. Reddy. Introduction Presentation for course, Level of Detail Management for Games. URL: `http://lodbook.com/course/2003/Reddy_Terrain.ppt`.

[26] M. Reddy. The Effects of Low Frame Rate on a Measure for User Performance in Virtual Environments, 1997.

[27] B. J. Schachter. Computer Image Generation for Flight Simulation. *IEEE Computer Graphics and Applications 1*, pages 29–68, 1981.

List of Figures

List of Tables

Appendix A

User Requirements

User requirements fall into two categories:

- Capabilities needed by users to solve a problem or achieve an objective
- Constraints placed by users on how the problem is to be solved

The categories above are displayed in a tabular format

1. Identifier is a unique identifier listed in the format:UR- <Unique identifier that reflects a hierarchical structure>
2. Description is a natural language statement of the requirement. It should be clear and verifiable.
3. Source is a reference to an external document (e.g. a system requirement document), or the name of the user or user group, that provided the requirement.
4. Need indicates whether the requirement is essential (ESS) or desirable (DES). Essential requirements are those that are non-negotiable, others are considered less important and subject to negotiation.
5. Rank is the level of desirability, where 1 is highest. Essential needs should always be ranked as level 1.
6. Stability indicates the likelihood that a requirement will change. Some user requirements may be known to be stable over the expected life of the software while others may be dependent on

feedback from the Software Requirement, Architectural Design and Detailed Design phases, or later. Unstable requirements are flagged TBC (to be confirmed).

A.1 Speci c Requirements

Table A.1: Geometry

UR	Description	Source	Need	Rank	Stability
UR-1.1.1	A 3D Virtual Terrain with height fields should be visible	Analysis	ESS	1	Stable
UR-1.1.2	The geometry may use textures (images) to enhance performance and the visual appearance of the virtual terrain.	Analysis	ESS	1	Stable
UR-1.1.3	The geometry meshes should be organised in such a way that the meshes with the highest resolution should be closest to the viewers viewpoint	Analysis	ESS	1	Stable
UR-1.1.4	The Virtual Terrain should be created from georeferenced data	Analysis	ESS	1	Stable
UR-1.1.5	A caching mechanism of data will be used to improve the performance of the application	Analysis	ESS	1	Stable
UR-1.1.6	The caching of data will be able to turn on/off	Analysis	ESS	1	Stable
UR-1.1.7	The size of the virtual terrain should be larger than the view frustums far clip-plane in all directions	Analysis	ESS	1	Stable
UR-1.1.8	It should be possible to load geometry from files, and put the loaded geometry on top of the 3D Virtual Terrain to get a better perspective of the actual size of the rendered terrain	Analysis	DES	2	TBC

Table A.2: Viewer

UR	Description	Source	Need	Rank	Stability
UR-1.2.1	The application should be possible to start by double-clicking an icon or by command prompt with input parameters	Analysis	ESS	1	Stable
UR-1.2.2	The user will be able to freely navigate in a virtual terrain environment to view geometry from any angle	Analysis	ESS	1	Stable
UR-1.2.3	Predefined user viewpoints will be provided to perform viewpoint transitions.	Analysis	ESS	1	Stable
UR-1.2.4	It should be possible to retrieve the current frame-rate from the application	Analysis	ESS	1	Stable
UR-1.2.5	It should be possible to retrieve the average frame-rate from the application	Analysis	ESS	1	Stable
UR-1.2.6	It should be possible to retrieve the stability of frame-rates over some time	Analysis	ESS	1	Stable
UR-1.2.7	It should be possible to time essential parts of the implemented prototype	Analysis	ESS	1	Stable
UR-1.2.8	The speed of the users viewpoint should be relative to the height over ground	Analysis	DES	2	Stable
UR-1.2.9	Snapshots in the form of raster images of the 3D environment can be taken at, at least, full-screen resolution and stored to the local file system	Analysis	DES	2	Stable

A.2 Constraint requirements

Table A.3: User Interfaces

UR	Description	Source	Need	Rank	Stability
UR-2.1.1	The primary input devices supported shall be a keyboard and a mouse	Analysis	ESS	1	Stable
UR-2.1.2	2D menus and dialogs shall be used as the primary user interface for selecting data files and controlling visualisation options	Analysis	ESS	1	Stable
UR-2.1.3	The language used to display information to the user shall be English	Analysis	ESS	1	Stable

Table A.4: Portability

UR	Description	Source	Need	Rank	Stability
UR-2.2.1	The software shall be platform independent, and run on most popular operating systems, including Windows 2000/XP, Mac OSX and Linux x86	Analysis	ESS	1	Stable

Table A.5: Standards

UR	Description	Source	Need	Rank	Stability
UR-2.3.1	The georeferenced data used to build up the Virtual Terrain should be the same as used in Nasa Worldwind	Analysis	ESS	1	Stable

Appendix B

Software Requirements

B.1 Speci c Requirements

Each software requirement is listed in a table, and contains fields for the following information.

- Identifier
- Description
- Source
- Need
- Rank
- Stability
- Type

B.1.1 Identi er

Uniquely identifies each software requirement. Each identifier will start with SR, which is an abbreviation for Software Requirement, followed by a unique number. The numbers will start with 000 and increase for each new requirement. The number does not imply the importance of the requirement.

B.1.2 Description

Describes what the requirement should do, it is intended that the description should be simple and clear.

B.1.3 Source

Identifies the origin of the requirement. Sources are given in respect of the applicable and reference documents list. When referenced to the User Requirement, please note when the user requirement document states the source Analysis, we consider the source for being the found by the developer under the initial analysis phase of the implementation.

B.1.4 Need

Identifies whether a requirement is essential for the operation of the system or desirable (and consequently negotiable). Each field is marked as essential (ESS) or desirable (DES).

B.1.5 Rank

Positions the requirement within a fixed hierarchy from 1 to 3, where one refers to the highest priority. Normally the requirements marked as ESS are of rank 1, whereas DES requirements are of rank 2 or more.

B.1.6 Stability

Identifies the perceived status of the source information at the time of publication. It takes the values STABLE or TBC.

B.1.7 Type

Each table refers to the requirement category. The functional requirements are divided into Columbus Geometry, 3D Viewer and Radiation Visualisation. This is done to reflect the User Requirement document better. Functional requirements The functional requirements will be divided into Columbus geometry, 3D Viewer and radiation visualisation.

Table B.1: Geometry

SR	Description	Source	Need	Rank	Stability	type
SR-1.1.1	A Java3D class creating elevation grids will be developed to generate a virtual terrain. DEM files will be used to retrieve the height data	UR-1.1.1	Analysis	ESS	1	Stable
SR-1.1.2	Sufficient information (Texture Coordinates) will be provided from the elevation grid class to make it possible to place images on top of created geometry	UR-1.1.2	Analysis	ESS	1	Stable
SR-1.1.3	A Quadtree multiresolution structure together with Java3Ds inbuilt DistanceLOD feature, will be utilised to make sure we have the highest resolution meshes closest to the users viewpoint	UR-1.1.3	Analysis	ESS	1	Stable
SR-1.1.4	The implemented prototype will use the same data as Nasa Worldwind. The data will be fetched through WMS and HTTP	UR-1.1.4	Analysis	ESS	1	Stable
SR-1.1.5	The design of the prototype will support strategies for caching of data through pluggable Java interfaces	UR-1.1.5	Analysis	ESS	1	Stable
SR-1.1.6	The design of the prototype will support strategies for enabling/disabling caching of tiles through pluggable Java interfaces	UR-1.1.5	Analysis	ESS	1	Stable
SR-1.1.7	Paging of tiles with textures from and to persistent hardware will be used to render a terrain larger than the view frustum in all directions	UR-1.1.6	Analysis	ESS	1	Stable
SR-1.1.8	By using the VRML loader for Java3D, we can load VRML files from disk and put them on top of other Java3D geometry	UR-1.1.6	Analysis	DES	2	Stable

Table B.2: Viewer

SR	Description	Source	Need	Rank	Stability	typ
SR-1.2.1	The application should be deployed from Java Web Start and/or shell script	UR-1.2.1	Analysis	ESS	1	Stab
SR-1.2.2	The already implemented navigation code from Halden Virtual Reality Centre will be utilised as navigation	UR-1.2.2	Analysis	ESS	1	Stab
SR-1.2.3	The already implemented viewpoint transition code from Halden Virtual Reality Centre will be utilised to perform viewpoint transition animations	UR-1.2.3	Analysis	ESS	1	Stab
SR-1.2.4	A Java3D behaviour will be developed that counts the current frame rate	UR-1.2.4	Analysis	ESS	1	Stab
SR-1.2.5	A Java3D behaviour will be developed that calculates the average framerate	UR-1.2.5	Analysis	ESS	1	Stab
SR-1.2.6	A Java3D behaviour will be developed that shows the stability of the framerate	UR-1.2.6	Analysis	ESS	1	Stab
SR-1.2.7	A static java class using the system clock in milliseconds will be used to calculate the time essential parts of the system	UR-1.2.7	Analysis	ESS	1	Stab
SR-1.2.8	The already implemented navigation from Halden Virtual Reality Centre will be extended to change the navigation speed relative to the users viewpoint over the ground	UR-1.2.8	Analysis	DES	2	Stab
SR-1.2.9	Snapshots in the form of raster images of the 3D environment can be taken at, at least, full-screen resolution and stored to the local file system. Rendering an off screen image of the scene, and save that to a JPG file format will handle this.	UR-1.2.9	Analysis	ESS	1	Stab

Table B.3: User Interfaces

SR	Description	Source	Need	Rank	Stability	type
SR-2.1.1	The primary input devices supported shall be a keyboard and a mouse.	UR-2.1.1	Analysis	ESS	1	Stable
SR-2.1.2	2D menus and dialogs shall be used as the primary user interface for selecting data files and controlling visualisation options. Utilising the java swing system will solve this..	UR-2.1.2	Analysis	ESS	1	Stable
SR-2.1.3	The language used to display information to the user shall be English	UR-2.1.3	Analysis	ESS	1	Stable

Table B.4: Portability

SR	Description	Source	Need	Rank	Stability	type
SR-2.2.1	The software shall be platform independent, and run on most popular operating systems, including Windows 2000/XP, Mac OS X and Linux x86. When software written in the Java programming language is compiled with Java technology, byte code results. The Java virtual machine, can explain (interpret) that byte code to any platform on which the Java virtual machine is installed. A Java virtual machine exists on most popular operating systems, including the ones that are mentioned above.	UR-2.2.1	Analysis	ESS	1	Stable

Table B.5: Standards

SR	Description	Source	Need	Rank	Stability	type
SR-2.3.1	The georeferenced data used to build up the Virtual Terrain should be the same as in Nasa Worldwind. The images used as textures will probably be fetched from the Web Mapping Server. Digital Elevation Models will be retrieved over the HTTP protocol	UR-2.3.1	Analysis	ESS	1	TBC

Appendix C

Class Diagram

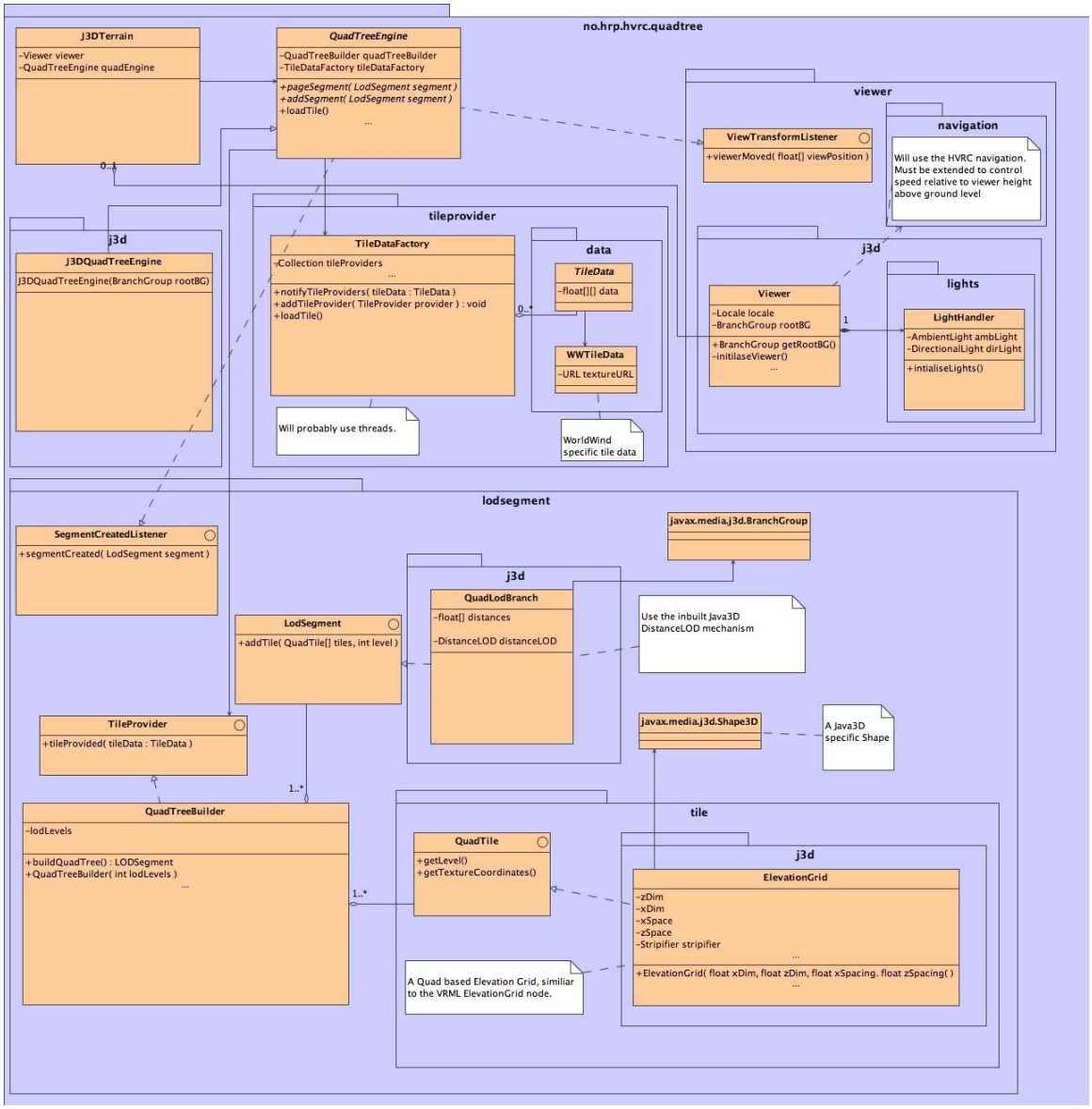


Figure C.1: The initial logical class diagram design

Appendix D

Sequence Diagram

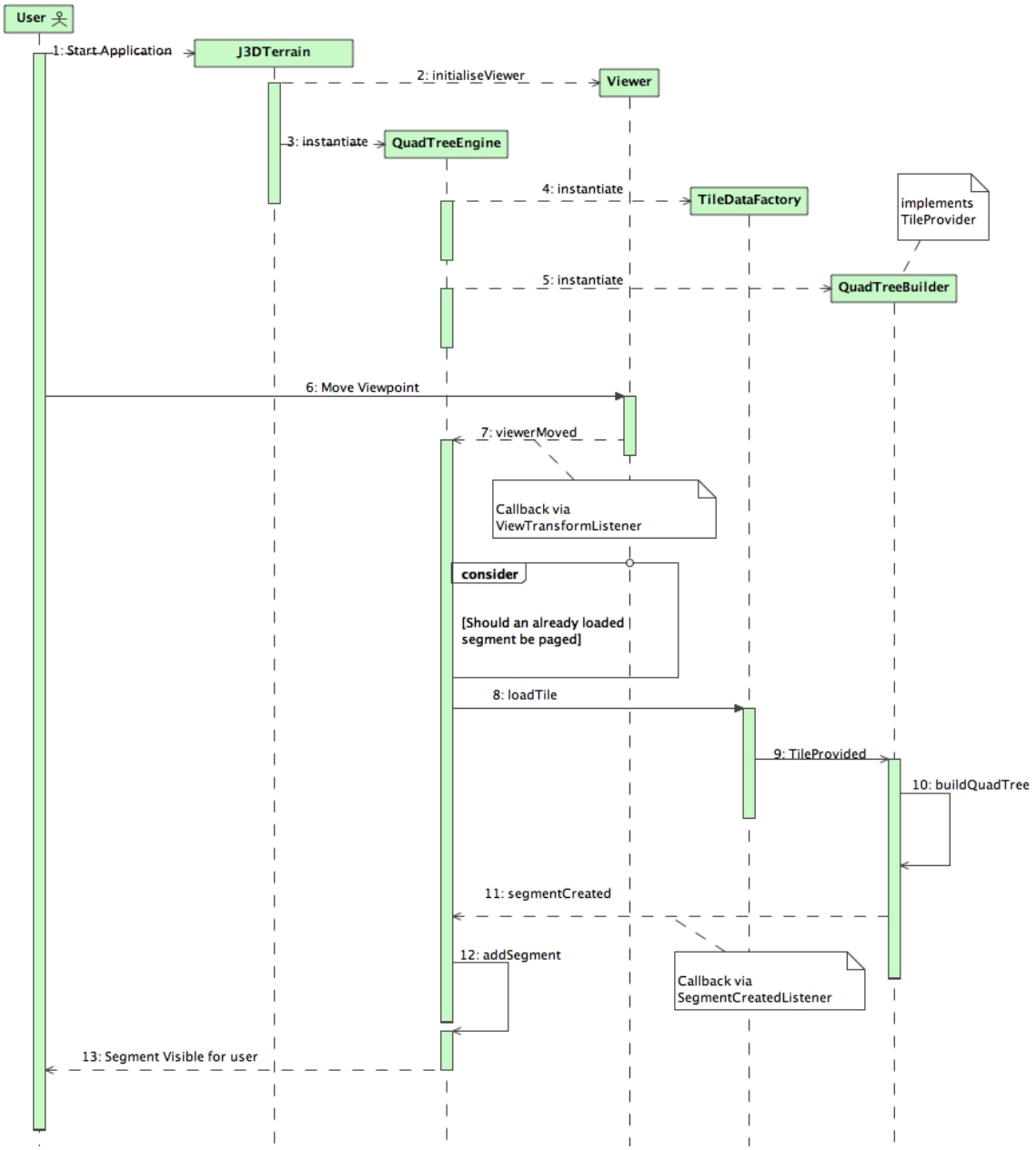


Figure D.1: The initial sequence diagram to support the logical class diagram

Appendix E

Component Description

E.1 Component Description

To give a better understanding of components in the logical models, the main components of this model are described below. The components of this systems are described as packages in the class diagram.

Function

The root and start uppackage for the prototype, the package contains the objects *J3DTerrain* and *QuadTreeEngine*. The objective of the *J3DTerrain* object is to start the tool, and create and initiliase the QuadTreeEngine object, which is a very important object in the prototype. It is responsible to

- Construct the objects that provides the application with the necessary data
- Construct the viewer that handle the 3D viewer part of the system
- Connect main parts of the system, so all necessary data is communicated within the prototype
- Defines the main functionality concerning the organising and paging of the tiles through public or abstract methods.

Interface

The *QuadTreeEngine* receives callbacks from the viewer each time the viewer moves the viewpoint, and connects the two parts together via the *ViewTransformListener*.

Processing

Based on the position and rotation received from the viewer, the *QuadTreeEngine* process this information and communicates with the *TileDateFactory* if new data should be loaded.

E.1.1 Viewer

Function

The viewer is responsible for creating and setting up the 3D part of the system. The 3D part is setup by reading Java3D confugiration file. By defining the functionality of the viewer in a configuration file, we will achieve better scalability for different viewing environments.

The viewer is also responsible for creating and initialising the navigation code. The component is also responsible for adding and replacing of loaded geometry within the prototype.

Depends

The viewer needs a java3d configuration file to be set up correctly, if not a proper configuration file is used, the prototype will not be able to start up.

The component is dependent on a navigation software library already developed at HVRC.

Processing

The viewer is responsible for processing the navigation inside the Virtual Terrain.

E.1.2 lodsegment

Function

The *lodsegment* package main responsibility is to create the geometry at various resolutions and setting up the level-of-detail optimisation. All 3D related parts are put into its own Java3D package for better de-coupling in the software.

Interface

The *lodsegment* component receives data via the *TileProvider* interface. When a new *lodsegment* created, it is communicated to all *SegmentCreatedListeners* via its interface.

Depends

The *lodsegment* depends on data from a *TileProvider* to be able to construct geometry

Processing

When *TileData* is received, the *QuadTreeBuilder* starts creating and setting up a quadtree structure with level-of-detail optimisation.

The *QuadLodBranch* extends a Java3D branchgroup, so we can detach and attach the entire branch runtime.

Data

The data that is needed is defined in the *TileData* interface inside the *tileprovider* package.

E.1.3 tileprovider

Function

The tileprovider package is responsible to provide all *TileProvider* listeners with data. The format of the data is defined in the *TileData* interface.

Depends

The component is controlled by the *QuadTreeEngine*, and needs a message when to load new data *Depends on WMS?*

Processing

When a message is received that a tile must be loaded, the tileprovider receives a callback from *QuadTreeEngineClass*, *connects to a WMS server, or cache????*, parses the data and puts it into the form that is defined in the *TileData* interface

Data

The component is using Nasa Worldwind Specific Data loaded through WMS. *It fetches the data over WMS. The specific dataset used for this prototype is SRTM30?*

Resources

Will use JCS (Java Caching System)?(not decided yet...)

E.1.4 timing

Function

The timing component is used to time essential parts of the virtual terrain implementation. There exists external software to perform measurement in java software, but for better control and platform-indepence two java classes are implemented for this purpose.

Processing

The *FPSBehaviour* class is a Java3D specific behaviour that ”wakes up” every time a new frame is rendered to the screen, this feature is used to count the current framerate.

The *Timing* class utilises the current time in milliseconds to calculate the time an event has performed, the timing is only used for analysis of different implementations of parts, and has no effect on the actual viretual terrain rendering.

Appendix F

Test Results

F.1 Test 1 - Quick Traversal

F.1.1 Averaged Test Results - Test 1

Average Frame Rate measured from FRAPS over all 5 test runs: 258.2

Table F.1: Average Times Original File-formats

Level	Total Tiles Processed	Total Quad-Tile Attempts	Acc. Time used on Quad-Tile in ms
0,1	38.8	38.8	40176.8
2	17.2	56.8	13990.8
3	30.2	87	19616.6
4	48.4	136.6	31034.6
5	33.6	188.8	131442.4

Average Frame Rate on Binary Tiles test measured from FRAPS over all 5 test runs: 1463.2506

Average Frame Rate on Binary Quad-Tiles test measured from FRAPS over all 5 test runs: 1171.213

Table F.2: Average Times Binary Tiles

Level	Total Tiles Processed	Total Quad-Tile Attempts	Acc. Time used on Quad-Tile in ms
0,1	37.2	37.2	232862.4
2	16.2	55.2	119609
3	15.2	80	83304.4
4	11	92.6	192207.6
5	1.4	94.4	41394.8

Table F.3: Average Times Binary Quad-Tiles

Level	Total Tiles Processed	Total Quad-Tile Attempts	Acc. Time used on Quad-Tile in ms
0, 1	39.6	39.6	72772.4
2	16.8	58.8	64233.6
3	17.2	87.2	123716
4	16.6	113	181858.6
5	2.4	94.4	62403.8

F.2 Test 1, Quick Traversal - Original Fileformats

Table F.4: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
20346	54413	0	1262	373.918

Table F.5: Test Run 1

Level	Total Tiles Processes	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	40	40	61105
2	18	58	21341
3	30	88	27578
4	49	137	44197
5	22	189	192379

Table F.6: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
12210	54465	0	1653	224.181

Table F.7: Test Run 2

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	38	38	32915
2	16	56	8827
3	31	87	22514
4	48	135	28885
5	39	188	175210

Table F.8: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
13190	55854	0	1647	236.151

Table F.9: Test Run 3

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	38	38	40669
2	17	56	11156
3	30	86	19361
4	49	137	31046
5	32	189	70452

Table F.10: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
15040	58649	0	1632	256.441

Table F.11: Test Run 4

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	39	39	30789
2	17	57	8451
3	30	87	13956
4	48	137	26813
5	36	189	130419

Table F.12: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
11480	56922	0	1633	201.679

Table F.13: Test Run 5

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	39	39	35406
2	18	57	9639
3	30	87	14674
4	48	137	24232
5	39	189	88752

F.3 Test 1 - Binary Tiles

Table F.14: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
103127	54146	1357	2257	1904.610

Table F.15: Test Run 1

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	34	34	318637
2	15	52	230729
3	8	68	81854
4	1	69	2074
5	0	69	0

Table F.16: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
62963	54113	8	1867	1163.547

Table F.17: Test Run 2

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	37	37	96896
2	16	55	3214
3	26	85	144256
4	20	109	431475
5	0	111	69041

Table F.18: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
86660	54104	1	2134	1601.730

Table F.19: Test Run 3

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	38	38	326125
2	18	56	182273
3	12	82	110810
4	3	89	6468
5	0	89	0

Table F.20: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
57427	54265	10	1909	1058.270

Table F.21: Test Run 4

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	38	38	115354
2	16	56	55151
3	18	82	3545
4	28	110	457462
5	7	117	197763

Table F.22: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
85808	54032	3	2185	1588.096

Table F.23: Test Run 5

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	39	39	307300
2	16	57	126678
3	12	83	76057
4	3	86	6186
5	0	86	0

F.4 Test 1 -Binary Quad-Tiles

Table F.24: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
88323	54188	0	2126	1629.937

Table F.25: Test Run 1

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	40	40	248459
2	18	60	124484
3	16	92	204824
4	5	107	53797
5	0	107	0

Table F.26: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
34490	63391	0	1861	544.084

Table F.27: Test Run 2

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	40	40	149573
2	14	58	2573
3	25	85	23236
4	34	124	437125
5	1	155	9211

Table F.28: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
71506	54098	0	2150	1321.786

Table F.29: Test Run 3

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	40	40	225626
2	19	60	95785
3	18	90	200902
4	3	110	9633
5	0	110	0

Table F.30: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
50903	66182	0	1882	769.137

Table F.31: Test Run 4

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	38	38	148084
2	14	56	20575
3	22	78	4199
4	36	114	375207
5	11	141	302808

Table F.32: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
86051	54082	0	2076	1591.121

Table F.33: Test Run 5

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	40	40	215778
2	19	60	109023
3	20	91	185419
4	5	110	33531
5	0	110	0

F.5 Test 2 -Original File-formats

F.6 Averaged Test Results - Test 2

Average Frame Rate on Original File-formats test measured from FRAPS over all 5 test runs: 1598.122

Table F.34: Average Times Original File-formats

Level	Total Tiles Processes	Total Quad-Tile Attempts	Acc. Time used on Quad-Tile in ms
0,1	21	21	12883
2	4	25	1023,4
3	4	29	1144,8
4	4	33	1764,8
5	4	37	2485,2

Average Frame Rate on Binary-Tiles test measured from FRAPS over all 5 test runs: 1623.8856

Table F.35: Average Times Binary Tiles

Level	Total Tiles Processed	Total Quad-Tile Attempts	Acc. Time used on Quad-Tile in ms
0,1	19.8	19.8	34143
2	4	26.2	5971,6
3	4	27.8	4690,4
4	3.6	31.4	2009,6
5	3.2	34.6	1664,2

Average Frame Rate on Binary Quad-Tiles test measured from FRAPS over all 5 test runs: 1648.7886

Table F.36: Average Times Binary Quad-Tiles

Level	Total Tiles Processed	Total Quad-Tile Attempts	Acc. Time used on Quad-Tile in ms
0,1	21	19.8	51086.2
2	4	25	4314,6
3	4	29	3458,6
4	4	33	3134
5	3.8	36.8	3811,8

Table F.37: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
29618	18595	366	2167	1592.794

Table F.38: Test Run 1

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	20955
2	4	25	1854
3	4	29	2462
4	4	33	2462
5	4	37	2850

Table F.39: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
30172	18597	306	2180	1622.412

Table F.40: Test Run 2

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	10587
2	4	25	810
3	4	29	826
4	4	33	1011
5	4	37	2321

Table F.41: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
29549	18584	140	2152	1590.024

Table F.42: Test Run 3

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	11078
2	4	25	827
3	4	29	810
4	4	33	1963
5	4	37	2011

Table F.43: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
29657	18726	224	2163	1583.734

Table F.44: Test Run 4

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	10554
2	4	25	829
3	4	29	938
4	4	33	1046
5	4	37	3231

Table F.45: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
29773	18589	99	2156	1601.646

Table F.46: Test Run 5

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	11241
2	4	25	797
3	4	29	686
4	4	33	2342
5	4	37	2013

F.7 Test 2 - Binary Tiles

Table F.47: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
34881	18552	1618	2063	1880.175

Table F.48: Test Run 1

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	15	15	102947
2	4	19	24740
3	4	23	19827
4	2	25	5256
5	0	25	0

Table F.49: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
28984	18724	82	2165	1547.960

Table F.50: Test Run 2

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	27185
2	4	25	1701
3	4	29	954
4	4	33	1686
5	4	37	1951

Table F.51: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
29192	18616	23	2177	1568.113

Table F.52: Test Run 3

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	14568
2	4	25	796
3	4	29	952
4	4	33	936
5	4	37	2169

Table F.53: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
29049	18574	158	2151	1563.960

Table F.54: Test Run 4

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	13163
2	4	25	1123
3	4	29	892
4	4	33	1108
5	4	37	2062

Table F.55: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
29120	18676	155	2139	1559.220

Table F.56: Test Run 5

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	12854
2	4	25	1498
3	4	29	827
4	4	33	1062
5	4	37	2139

F.8 Test 2 - Binary Quad Tiles

Table F.57: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
32956	18545	1435	2080	1777.083

Table F.58: Test Run 1

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	122984
2	4	25	17934
3	4	29	13512
4	4	33	10530
5	3	36	10577

Table F.59: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
32956	18545	1435	2080	1777.083

Table F.60: Test Run 2

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	34514
2	4	25	954
3	4	29	812
4	4	33	1002
5	4	37	2109

Table F.61: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
30508	18712	68	2182	1630.398

Table F.62: Test Run 3

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	33680
2	4	25	858
3	4	29	672
4	4	33	2248
5	4	37	1967

Table F.63: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
30467	18791	149	2172	1621.361

Table F.64: Test Run 4

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	32213
2	4	25	874
3	4	29	1140
4	4	33	905
5	4	37	2406

Table F.65: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
29990	18683	299	2156	1605.203

Table F.66: Test Run 5

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	32040
2	4	25	953
3	4	29	1156
4	4	33	1030
5	4	37	2000

F.9 Test 3 - Original Fileformats

F.10 Averaged Test Results - Large Geographic area traversal - Test 3

Average Frame Rate on Original File-formats test measured from FRAPS over all 5 test runs: 1048.2038

Table F.67: Average Times Original Fileformats

Level	Total Tiles Processed	Total Quad-Tile Attempts	Acc. Time used on Quad-Tile in ms
0,1	296.2	301	194760.8
2	102.6	403.6	57311.4
3	195.6	599.8	85485.4
4	354.4	956.8	193339.4
5	466.4	1436.2	330865.6

Average Frame Rate on Binary Tiles test measured from FRAPS over all 5 test runs: 1752.9192

Table F.68: Average Times Binary tiles

Level	Total Tiles Processed	Total Quad-Tile Attempts	Acc. Time used on Quad-Tile in ms
0,1	192.4	259.2	640338
2	90	381.2	598244.6
3	164.6	513.8	911979.8
4	138.2	798.2	697905.6
5	15.2	837.4	24651

Average Frame Rate on Binary Tiles test measured from FRAPS over all 5 test runs: 1445.9676

Table F.69: Average Times Binary Quad-Tiles

Level	Total Tiles Processed	Total Quad-Tile Attempts	Acc. Time used on Quad-Tile in ms
0,1	208	314.6	970763.6
2	110	424.6	567943.4
3	213.6	638.2	961899.6
4	319.8	1048.6	1468871.4
5	3.6	1221.4	10210.4

Table F.70: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	483964	0	2325	1033.135

Table F.71: Test Run 1

Level	Total Tiles Processes	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	295	300	206288
2	103	403	55566
3	199	602	108316
4	357	962	204147
5	467	1444	352330

Table F.72: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	499948	0	2122	1000.104

Table F.73: Test Run 2

Level	Total Tiles Processes	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	304	310	209981
2	106	416	71978
3	200	616	87746
4	360	980	195582
5	473	1467	346836

Table F.74: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	468651	0	2285	1066.892

Table F.75: Test Run 3

Level	Total Tiles Processes	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	300	302	203969
2	103	405	62162
3	198	603	80387
4	360	965	189192
5	473	1451	339642

Table F.76: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	464179	0	2245	1077.171

Table F.77: Test Run 4

Level	Total Tiles Processes	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	291	295	173135
2	100	395	48887
3	190	585	69161
4	343	931	181400
5	457	1397	315517

Table F.78: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	470050	0	2280	1063.717

Table F.79: Test Run 5

Level	Total Tiles Processes	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	291	298	180431
2	101	399	47964
3	194	593	81817
4	352	946	196376
5	462	1422	300003

F.11 Test 3 - Binary Tiles

Table F.80: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	400015	0	2235	1249.953

Table F.81: Test Run 1

Level	Total Tiles Processes	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	198	271	736329
2	93	364	662354
3	173	537	990693
4	132	834	692851
5	6	856	8157

Table F.82: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	410543	0	2170	1217.899

Table F.83: Test Run 2

Level	Total Tiles Processes	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	206	269	598233
2	90	359	563523
3	164	523	859902
4	165	808	791986
5	28	870	41317

Table F.84: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	197383	737	2565	2533.146

Table F.85: Test Run 3

Level	Total Tiles Processes	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	191	256	578076
2	85	341	563741
3	152	493	833485
4	121	759	623855
5	21	797	39529

Table F.86: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	392412	0	2212	1274.171

Table F.87: Test Run 4

Level	Total Tiles Processes	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	179	249	658138
2	95	344	640824
3	174	518	953593
4	143	816	739877
5	6	848	10750

Table F.88: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
94583	37857	2377	2547	2498.428

Table F.89: Test Run 5

Level	Total Tiles Processes	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	188	251	630914
2	87	338	560781
3	160	498	922226
4	130	774	640959
5	15	816	23502

F.12 Test 3 - Binary Quad-Tiles

Table F.90: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	292908	0	2188	1707.021

Table F.91: Test Run 1

Level	Total Tiles Processes	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	209	317	1098103
2	110	427	579157
3	213	640	995547
4	308	1054	1442166
5	4	1195	11223

Table F.92: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	373046	0	2143	1340.317

Table F.93: Test Run 2

Level	Total Tiles Processes	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	210	317	1231272
2	110	427	613391
3	214	641	1005508
4	289	1044	1351799
5	2	1188	3451

Table F.94: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	399661	0	2127	1251.060

Table F.95: Test Run 3

Level	Total Tiles Processes	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	203	313	838033
2	110	423	533234
3	214	637	936896
4	344	1048	1597936
5	2	1238	4688

Table F.96: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	406521	0	2093	1229.949

Table F.97: Test Run 4

Level	Total Tiles Processes	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	209	313	834200
2	110	423	563427
3	213	636	934420
4	341	1048	1522489
5	5	1242	13969

Table F.98: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	293860	785	2105	1701.491

Table F.99: Test Run 5

Level	Total Tiles Processes	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	209	313	852210
2	110	423	550508
3	214	637	937127
4	317	1051	1429967
5	5	12442	17721

Appendix G

Test Results Geometry

G.1 Testing with geometry - Test 1

Average Frame Rate on Original File-formats test measured from FRAPS over all 5 test runs: 777.3332

Table G.1: Average Times Original Fileformats

Level	Total Tiles Processed	Total Quad-Tile Attempts	Acc. Time used on Quad-Tile in ms
0,1	40	40	8206.6
2	20	60	6612
3	32	92	11750.4
4	60.6	153.8	28460.4
5	74.8	264.2	207963.2

Average Frame Rate on Binary Tiles test measured from FRAPS over all 5 test runs: 1511.9608

Table G.2: Average Times Binary Tiles

Level	Total Tiles Processed	Total Quad-Tile Attempts	Acc. Time used on Quad-Tile in ms
0,1	40	40	10765
2	20	60	7182,6
3	32	92	12059.6
4	60.4	154	32594.2
5	81	265	212854.6

Average Frame Rate on Binary Tiles test measured from FRAPS over all 5 test runs: 1445.0612

Table G.3: Average Times Binary Quad-Tiles

Level	Total Tiles Processed	Total Quad-Tile Attempts	Acc. Time used on Quad-Tile in ms
0,1	39.8	39.8	33837
2	20	59.8	30914.4
3	32	91.8	40566.2
4	55	149.8	382614.4
5	46.6	239.4	658250

G.2 Test 1 - Original Fileformats

Table G.4: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
44912	56487	0	1869	795.086

Table G.5: Test Run 1

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	40	40	6322
2	20	60	4049
3	32	92	10123
4	60	153	31942
5	74	263	262074

Table G.6: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
40463	58202	0	1746	695.217

Table G.7: Test Run 2

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	40	40	10327
2	20	60	7674
3	32	92	11733
4	60	154	24009
5	76	261	161018

Table G.8: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
46557	55394	0	1897	840.470

Table G.9: Test Run 3

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	40	40	7059
2	20	60	5856
3	32	92	12755
4	62	154	28367
5	76	265	198540

Table G.10: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
40543	55224	0	1923	734.155

Table G.11: Test Run 4

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	40	40	9201
2	20	60	8202
3	32	92	12644
4	60	154	24918
5	77	268	240041

Table G.12: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
44696	54392	0	1939	821.738

Table G.13: Test Run 5

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0	40	40	8124
2	20	60	7279
3	32	92	11497
4	61	154	32066
5	71	264	178143

G.3 Test 1 - Binary Tiles

Table G.14: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
97266	54667	1283	2161	1779.245

Table G.15: Test Run 1

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	40	40	8624
2	20	60	8775
3	32	92	14554
4	60	154	33250
5	82	264	295468

Table G.16: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
75490	54257	561	1877	1391.341

Table G.17: Test Run 2

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	40	40	6923
2	20	60	5147
3	32	92	15338
4	62	154	35449
5	78	265	177889

Table G.18: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
78923	54251	399	2040	1454.775

Table G.19: Test Run 3

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	40	40	15588
2	20	60	6958
3	32	92	9191
4	60	154	26591
5	84	266	202318

Table G.20: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
83788	54265	33	2066	1544.052

Table G.21: Test Run 4

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	40	40	15588
2	20	60	6958
3	32	92	9191
4	60	154	26591
5	84	266	202318

Table G.22: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
75241	54115	0	2055	1390.391

Table G.23: Test Run 5

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	40	40	7102
2	20	60	8075
3	32	92	12024
4	60	154	41090
5	77	264	186280

G.4 Test 1 - Binary Quad-Tiles

Table G.24: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
86947	54718	765	1962	1589.002

Table G.25: Test Run 1

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	39	39	47180
2	20	59	46949
3	32	91	56335
4	49	146	557630
5	5	174	123485

Table G.26: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
74015	54056	378	1867	1369.228

Table G.27: Test Run 2

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	40	40	27926
2	20	60	21107
3	32	92	33462
4	56	151	109196
5	62	254	174288

Table G.28: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
77442	54040	449	1927	1433.050

Table G.29: Test Run 3

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	40	40	22742
2	20	60	20325
3	32	92	29838
4	57	151	343105
5	77	260	122899

Table G.30: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
77371	54040	501	1940	1431.736

Table G.31: Test Run 4

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	40	40	33321
2	20	60	30558
3	32	92	38341
4	58	150	436841
5	54	252	118495

Table G.32: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
75676	53966	4	1874	1402.290

Table G.33: Test Run 5

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	40	40	38016
2	20	60	35633
3	32	92	44855
4	55	151	466300
5	50	257	119083

G.5 Test 2 - Zoom In And Out Test

Average Frame Rate on Original File-formats test measured from FRAPS over all 5 test runs: 1922.9364

Table G.34: Average Times Original Fileformats

Level	Total Tiles Processed	Total Quad-Tile Attempts	Acc. Time used on Quad-Tile in ms
0,1	21	21	1466,4
2	4	25	331.22
3	4	29	281.4
4	4	33	506.4
5	4	37	1847.6

Average Frame Rate on Binary Tiles test measured from FRAPS over all 5 test runs: 1956.9636

Average Frame Rate on Binary Quad-Tiles test measured from FRAPS over all 5 test runs: 1929.715

Table G.35: Average Times Binary Tiles

Level	Total Tiles Processed	Total Quad-Tile Attempts	Acc. Time used on Quad-Tile in ms
0,1	21	21	8413,4
2	4	25	1490.8
3	4	29	1033.6
4	4	33	1204.2
5	4	37	3952.8

Table G.36: Average Times Binary Tiles

Level	Total Tiles Processed	Total Quad-Tile Attempts	Acc. Time used on Quad-Tile in ms
0,1	21	21	3523,2
2	4	25	486.96
3	4	29	543.8
4	4	33	1052.8
5	4	37	3567

Table G.37: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
35577	18498	1079	2337	1923.289

Table G.38: Test Run 1

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	1296
2	4	25	266
3	4	29	219
4	4	33	392
5	4	37	1625

Table G.39: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
34377	18505	757	2274	1857.714

Table G.40: Test Run 2

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	1467
2	4	25	266
3	4	29	390
4	4	33	390
5	4	37	1844

Table G.41: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
35906	18495	951	2328	1941.390

Table G.42: Test Run 3

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	1740
2	4	25	390
3	4	29	187
4	4	33	452
5	4	37	1986

Table G.43: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
35722	18494	853	2322	1931.545

Table G.44: Test Run 4

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	1328
2	4	25	423
3	4	29	330
4	4	33	672
5	4	37	1860

Table G.45: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
36262	18494	963	2309	1960.744

Table G.46: Test Run 5

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	1501
2	4	25	311
3	4	29	281
4	4	33	626
5	4	37	1923

G.6 Test 2 - Binary Tiles

Table G.47: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
36852	18730	1603	2263	1967.539

Table G.48: Test Run 1

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	37521
2	4	25	6865
3	4	29	4515
4	4	33	5162
5	4	37	15878

Table G.49: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
36942	18739	1186	2342	1971.397

Table G.50: Test Run 2

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	1251
2	4	25	139
3	4	29	171
4	4	33	187
5	4	37	1050

Table G.51: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
36889	18724	1205	2315	1970.145

Table G.52: Test Run 3

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	1031
2	4	25	186
3	4	29	218
4	4	33	262
5	4	37	1052

Table G.53: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
36318	18718	1192	2330	1940.271

Table G.54: Test Run 4

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	1272
2	4	25	110
3	4	29	154
4	4	33	246
5	4	37	928

Table G.55: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
35988	18593	1211	2286	1935.567

Table G.56: Test Run 5

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	992
2	4	25	154
3	4	29	110
4	4	33	264
5	4	37	856

G.7 Test 2 - Binary Quad-Tiles

Table G.57: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
35943	18708	1596	2285	1921.264

Table G.58: Test Run 1

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	12718
2	4	25	1748
3	4	29	2228
4	4	33	4284
5	4	37	14181

Table G.59: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
36285	18693	1181	2289	1941.101

Table G.60: Test Run 2

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	1177
2	4	25	170
3	4	29	139
4	4	33	246
5	4	37	992

Table G.61: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
35807	18627	1207	2254	1922.317

Table G.62: Test Run 3

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	1043
2	4	25	173
3	4	29	172
4	4	33	250
5	4	37	902

Table G.63: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
36260	18588	1215	2296	1950.721

Table G.64: Test Run 4

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	1460
2	4	25	158
3	4	29	60
4	4	33	234
5	4	37	872

Table G.65: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
35585	18600	1140	2254	1913.172

Table G.66: Test Run 5

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	21	21	1218
2	4	25	186
3	4	29	60
4	4	33	250
5	4	37	888

G.8 Test 3 - Large Geographic Area Travel - Original File-formats

Average Frame Rate on Binary Quad-Tiles test measured from FRAPS over all 5 test runs: 1429.6442

Table G.67: Average Times Original Fileformats

Level	Total Tiles Processed	Total Quad-Tile Attempts	Acc. Time used on Quad-Tile in ms
0,1	314	314	47508.2
2	110	424	14984.2
3	214	638	10201.6
4	426	1064	9714.6
5	846	1910	407989

Average Frame Rate on Binary Quad-Tiles test measured from FRAPS over all 5 test runs: 1664.6934

Table G.68: Average Times Binary Files

Level	Total Tiles Processed	Total Quad-Tile Attempts	Acc. Time used on Quad-Tile in ms
0,1	301.6	314	189879.6
2	110	424	167993.22
3	214	638	328433.8
4	417.4	1062	927001.2
5	342.4	1779.6	534183

Average Frame Rate on Binary Quad-Tiles test measured from FRAPS over all 5 test runs: 1659.1369

Table G.69: Average Times Binary Files

Level	Total Tiles Processed	Total Quad-Tile Attempts	Acc. Time used on Quad-Tile in ms
0,1	313	314	136224.8
2	110	424	140694.2
3	214	638	259155
4	426	1064	811324.6
5	257.8	1904.8	694559.4

Table G.70: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	355932	4	2250	1404.763

Table G.71: Test Run 1

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	314	314	42853
2	110	424	14536
3	214	638	10105
4	426	1064	114266
5	846	1910	433892

Table G.72: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	350678	96	2280	1425.809

Table G.73: Test Run 2

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	314	314	55962
2	110	424	14480
3	214	638	9597
4	426	1064	101473
5	846	1910	404095

Table G.74: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	347845	5	2253	1437.422

Table G.75: Test Run 3

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	314	314	47904
2	110	424	16815
3	214	638	9685
4	426	1064	105109
5	846	1910	404432

Table G.76: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	343889	15	2240	1453.958

Table G.77: Test Run 4

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	314	314	45543
2	110	424	12754
3	214	638	10319
4	426	1064	95690
5	846	1910	387166

Table G.78: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	350565	5	2278	1426.269

Table G.79: Test Run 5

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	314	314	45279
2	110	424	16336
3	214	638	8867
4	426	1064	102365
5	846	1910	410360

G.9 Test 3 - Binary Tiles

Table G.80: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	283207	1090	2236	1765.493

Table G.81: Test Run 1

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	272	314	240221
2	110	424	277707
3	214	638	512532
4	411	1061	1234993
5	158	1672	265847

Table G.82: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	302750	724	2279	1651.528

Table G.83: Test Run 2

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	309	314	193260
2	110	424	160020
3	214	638	297850
4	418	1063	882522
5	353	1794	596188

Table G.84: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	304736	8	2289	1640.764

Table G.85: Test Run 3

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	308	314	172359
2	110	424	134164
3	214	638	266448
4	422	1063	869694
5	405	1808	632672

Table G.86: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	305937	76	2259	1634.323

Table G.87: Test Run 4

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	310	314	169418
2	110	424	137136
3	214	638	275304
4	420	1062	848143
5	404	1816	600844

Table G.88: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	306493	0	2284	1631.359

Table G.89: Test Run 5

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	309	314	174140
2	110	424	130939
3	214	638	290035
4	416	1061	799654
5	392	1808	575364

G.10 Test 3 - Binary Quad-Tiles

Table G.90: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	291307	685	2295	1716.402

Table G.91: Test Run 1

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	313	314	207063
2	110	424	181041
3	214	638	359922
4	426	1064	1007726
5	117	1894	334975

Table G.92: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	303036	703	2265	1649.969

Table G.93: Test Run 2

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	314	314	131170
2	110	424	133779
3	214	638	235945
4	426	1064	769958
5	270	1908	790243

Table G.94: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	297194	273	2289	1682.403

Table G.95: Test Run 3

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	312	314	111257
2	110	424	133735
3	214	638	236336
4	426	1064	765648
5	297	1909	808827

Table G.96: Output from FRAPS to test run in table below

Frames	Time (ms)	Min	Max	Avg
500000	310495	3	2258	1610.332

Table G.97: Test Run 4

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	313	314	107689
2	110	424	121513
3	214	638	225080
4	426	1064	749293
5	330	1909	832965

Table G.98: Output from FRAPS to test run in table below

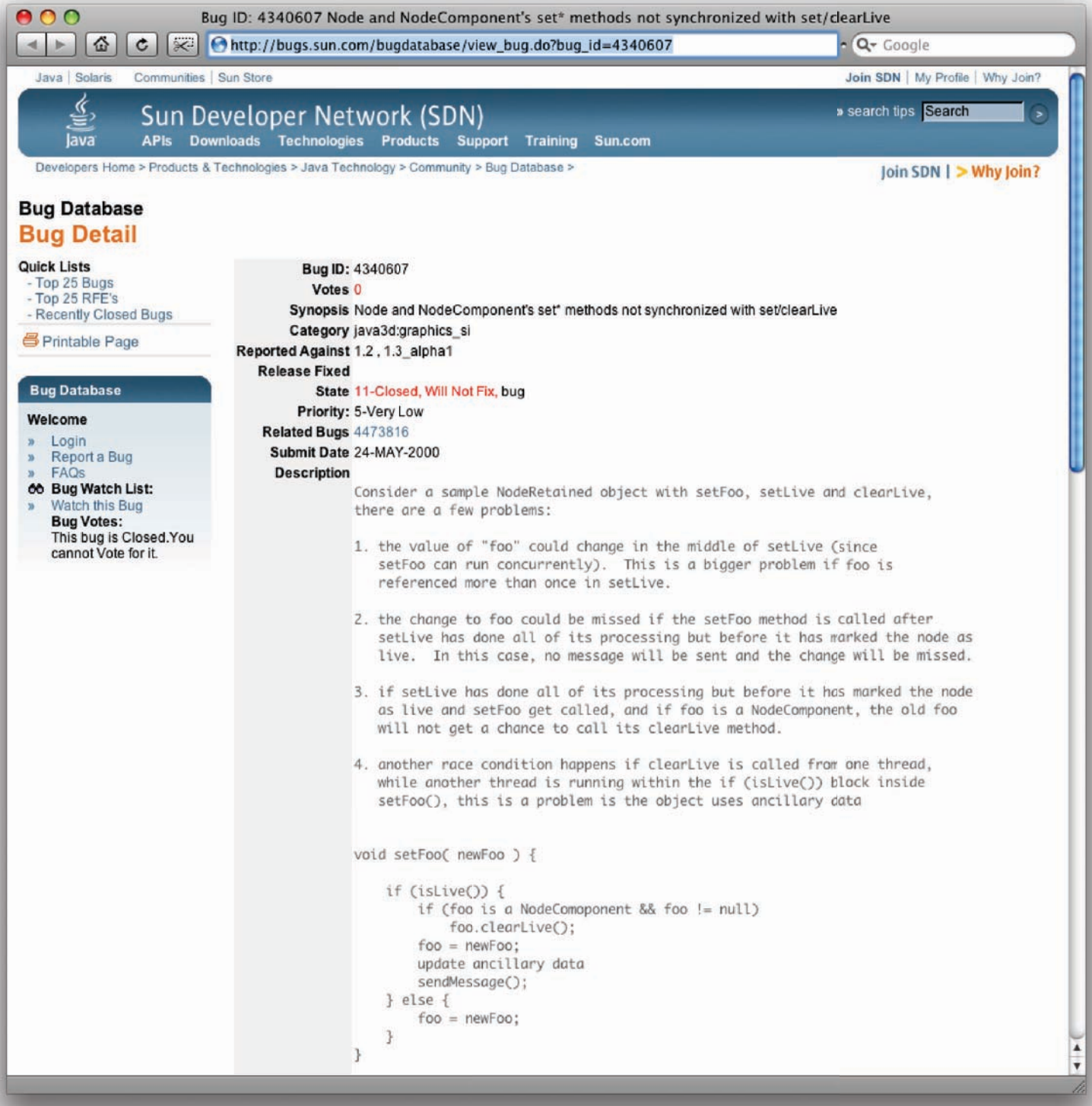
Frames	Time (ms)	Min	Max	Avg
500000	305516	214	2264	1636.575

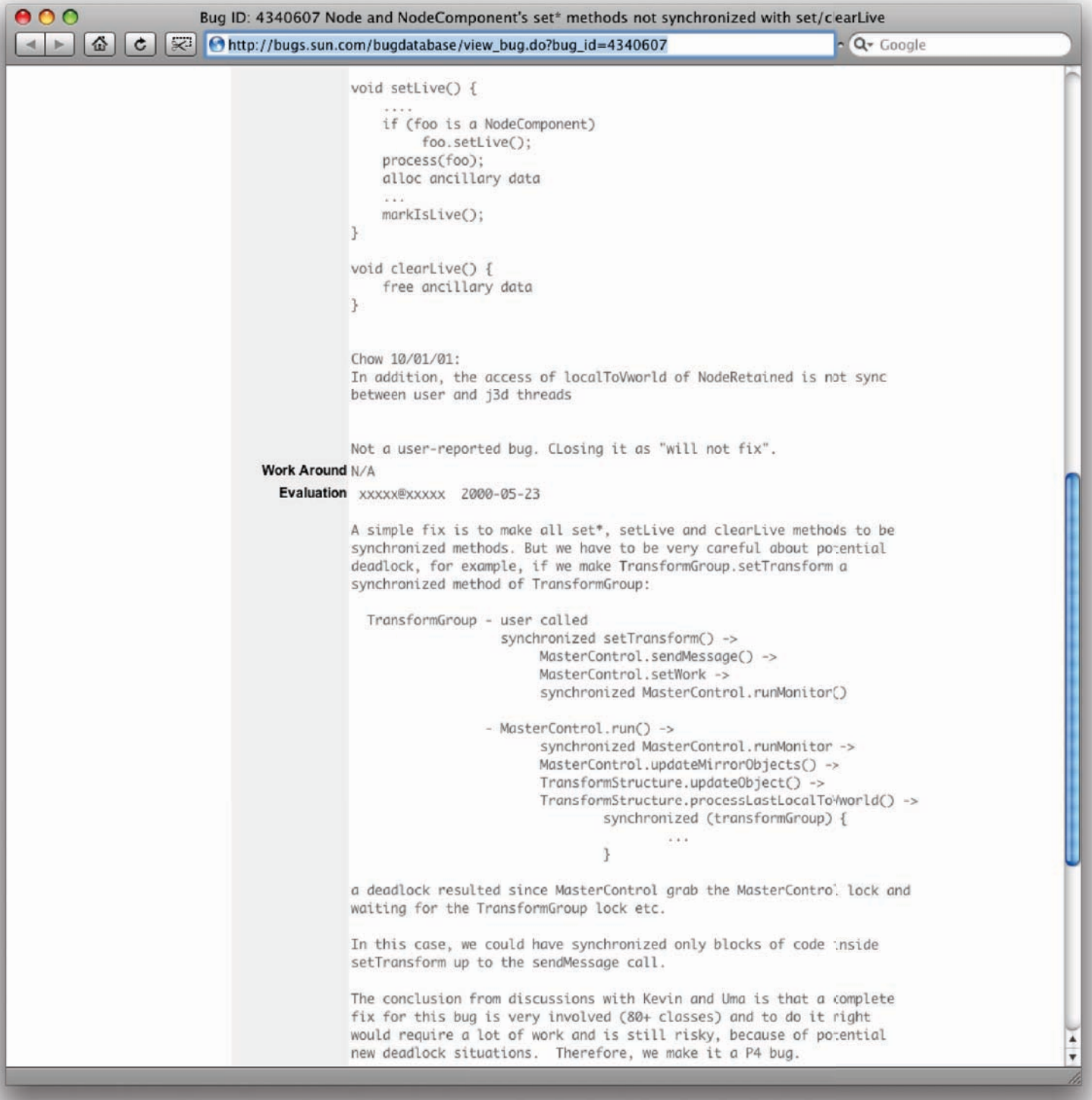
Table G.99: Test Run 5

Level	Total Tiles Processed	Total Quad-Tile Attempts	Accumulated Time used on Quad-Tile
0,1	313	314	123945
2	110	424	133403
3	214	638	238492
4	426	1064	763998
5	275	1904	705787

Appendix H

Java3D bug 4340607


Bug ID: 4340607 Node and NodeComponent's set* methods not synchronized with set/clearLive
http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4340607
Google
Java | Solaris Communities | Sun Store
Join SDN | My Profile | Why Join?
Sun Developer Network (SDN)
» search tips Search
Java APIs Downloads Technologies Products Support Training Sun.com
Developers Home > Products & Technologies > Java Technology > Community > Bug Database >
Join SDN | > Why Join?
Bug Database
Bug Detail
Quick Lists
- Top 25 Bugs
- Top 25 RFE's
- Recently Closed Bugs
Printable Page
Bug Database
Welcome
» Login
» Report a Bug
» FAQs
Bug Watch List:
» Watch this Bug
Bug Votes:
This bug is Closed.You cannot Vote for it.
Bug ID: 4340607
Votes 0
Synopsis Node and NodeComponent's set* methods not synchronized with set/clearLive
Category java3d:graphics_si
Reported Against 1.2 , 1.3_alpha1
Release Fixed
State 11-Closed, Will Not Fix, bug
Priority: 5-Very Low
Related Bugs 4473816
Submit Date 24-MAY-2000
Description
Consider a sample NodeRetained object with setFoo, setLive and clearLive,
there are a few problems:
1. the value of "foo" could change in the middle of setLive (since
setFoo can run concurrently). This is a bigger problem if foo is
referenced more than once in setLive.
2. the change to foo could be missed if the setFoo method is called after
setLive has done all of its processing but before it has marked the node as
live. In this case, no message will be sent and the change will be missed.
3. if setLive has done all of its processing but before it has marked the node
as live and setFoo get called, and if foo is a NodeComponent, the old foo
will not get a chance to call its clearLive method.
4. another race condition happens if clearLive is called from one thread,
while another thread is running within the if (isLive()) block inside
setFoo(), this is a problem is the object uses ancillary data
void setFoo(newFoo) {
if (isLive()) {
if (foo is a NodeComoponent && foo != null)
foo.clearLive();
foo = newFoo;
update ancillary data
sendMessage();
} else {
foo = newFoo;
}
}


Bug ID: 4340607 Node and NodeComponent's set* methods not synchronized with set/clearLive
http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4340607
Google
void setLive() {
....
if (foo is a NodeComponent)
foo.setLive();
process(foo);
alloc ancillary data
...
markIsLive();
}
void clearLive() {
free ancillary data
}
Chow 10/01/01:
In addition, the access of localToVworld of NodeRetained is not sync
between user and j3d threads
Not a user-reported bug. CLosing it as "will not fix".
Work Around N/A
Evaluation xxxxx@xxxxx 2000-05-23
A simple fix is to make all set*, setLive and clearLive methods to be
synchronized methods. But we have to be very careful about potential
deadlock, for example, if we make TransformGroup.setTransform a
synchronized method of TransformGroup:
TransformGroup - user called
synchronized setTransform() ->
MasterControl.sendMessage() ->
MasterControl.setWork ->
synchronized MasterControl.runMonitor()
- MasterControl.run() ->
synchronized MasterControl.runMonitor ->
MasterControl.updateMirrorObjects() ->
TransformStructure.updateObject() ->
TransformStructure.processLastLocalToVworld() ->
synchronized (transformGroup) {
...
}
a deadlock resulted since MasterControl grab the MasterControl lock and
waiting for the TransformGroup lock etc.
In this case, we could have synchronized only blocks of code inside
setTransform up to the sendMessage call.
The conclusion from discussions with Kevin and Uma is that a complete
fix for this bug is very involved (80+ classes) and to do it right
would require a lot of work and is still risky, because of potential
new deadlock situations. Therefore, we make it a P4 bug.