**A HIERARCHICAL MEMORY MODEL FOR TERRAIN DATA MANAGEMENT IN INTERACTIVE TERRAIN VISUALIZATION**

By

Ricardo Veguilla González

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

UNIVERSITY OF PUERTO RICO
MAYAGÜEZ CAMPUS

December, 2007

Approved by:

| | |
|---|---|
| Domingo Rodríguez, Ph.D<br>Member, Graduate Committee | Date |
| Manuel Rodríguez, Ph.D<br>Member, Graduate Committee | Date |
| Nayda G. Santiago, Ph.D<br>Chairperson, Graduate Committee | Date |
| Pedro Vásquez, Ph.D<br>Representative of Graduate School | Date |
| Isidoro Couvertier, Ph.D<br>Chairperson of the Department | Date |

Abstract of Thesis Presented to the Office of Graduate Studies
of the University of Puerto Rico in Partial Fulfillment of the
Requirements for the Degree of Master of Science

**A HIERARCHICAL MEMORY MODEL FOR TERRAIN DATA MANAGEMENT IN INTERACTIVE TERRAIN VISUALIZATION**

By

Ricardo Veguilla González

December 2007

Chair: Nayda G. Santiago
Major Department: Electrical and Computer Engineering

This work describes the use of a hierarchical memory model for the performance estimation of the interactive terrain visualization system as part of the system design process. Such visualization systems must be capable of: a) working with potentially massive datasets which can exceed the both the memory and the disk storage capacity; b) rendering the data at a real-time frame-rate which exceed the capabilities of general-purpose CPUs. While several solutions to these problems have been presented in the literature, they failed to address the general recurrent performance problem of the timely movement of data throughout the complete visualization system. This work presents a model for the estimation of the performance of interactive terrain visualization systems which incorporates the characteristics of hardware components found in visualization systems as well as the characteristics and access patterns associated with terrain data. To validate our model, we performed a test case study. Our results show that the model is able to characterize the general performance behavior of the visualization systems, but the estimated performance diverged from the measured performance as the dataset sizes increased.

Resumen de Tesis Presentado a la Oficina de Estudios Graduados
de la Universidad de Puerto Rico como Cumplimiento Parcial de los
Requisitos para el Grado de Maestría en Ciencias

# MODELO DE MEMORIA JERÁRQUICA PARA MANEJO DE DATA EN LA VISUALIZACIÓN INTERACTIVA DE TERRENOS

Por

Ricardo Veguilla González

Diciembre 2007

Consejera: Nayda G. Santiago
Departamento: Ingeniería Eléctrica y Computadoras

En este trabajo se describe el uso de un modelo jerárquico de memoria para estimar el rendimiento de sistemas interactivos de visualización de terrenos con la meta de asistir procesos de diseño durante el desarrollo del sistems. Dichos sistemas deben ser capaces de: a) manejar datos cuyo tamaño puede exceder la capacidad de la memoria primaria y la capacidad de almacenaje en disco; b) dibujar la data en tiempo real, lo que excede las capacidades de procesamiento de las unidades centrales de procesamiento. A pesar de que numerosas soluciones han sido presentadas en la literatura, estas no consideran el problema recurrente de rendimiento asociado al movimiento de data a través del sistema completo de visualización. En este trabajo presentamos un modelo que incorpora tanto las características de los componentes de hardware que forman parte de los sistemas de visualización, como las características y patrones de acceso de la data de terrenos. Utilizando el modelo descrito se puede identificar las características de los componentes de hardware y de software requeridos para alcanzar las metas de rendimiento deseadas en sistemas de visualización interactiva de terrenos.

I dedicate this thesis to my family.

ACKNOWLEDGMENTS

The author wishes to express his gratitude to the following people:

- Dr. Nayda Santiago and Dr. Domingo Rodríguez for giving me the opportunity to work in the WALSAIP project.
- Dr. Manuel Pérez Quiñones who helped me get started in the field of computer graphics.
- Dr. Néstor Rodríguez, whose lectures on memory systems were central to the development of the main ideas presented in this work.
- Angel Villalain, Eric Fortis, Javier Malavé, Rubén Nieves, Carlos Pérez, Héctor Ramos, Héctor Irizarry, Martha Roldán, Peter Bangdiwala, and Vazjier Rosario for their various contributions to the development of the WALSAIP Visual Terrain Explorer.

This work has been supported by the National Science Foundation under the CISE-CNS Grant Number 0424546.

TABLE OF CONTENTS

LIST OF TABLES

Table page

LIST OF FIGURES

# LIST OF ABBREVIATIONS

| | |
|---|---|
| CPU | Central Processor Unit |
| GPU | Graphic Processor Unit |
| API | Application Programming Interface |
| LOD | Level of Detail |
| EMM | External Memory Management |
| DS | Data Streaming |
| HMM | Hierarchical Memory Model |
| TIN | Triangulated Irregular Network |
| PCA | Principal Component Analysis |
| WALSAIP | Wide Area, Large-Scale Automated Information Processing |
| VTE | Visual Terrain Explorer |
| JOGL | Java OpenGL |
| FPS | Frames per second |
| GUI | Graphical User Interface |
| MVC | Model View Controller |
| J2SE | Java 2 Standard Edition |
| JVM | Java Virtual Machine |
| GC | Garbage Collection |
| JIT | Just-In-Time |
| RCP | Rich-Client Platform |
| TPTP | Test and Performance Tools Platform |
| RGB | Red, Green, Blue |
| GIS | Geographic Information System |
| RAID | Redundant Array of Independent Drives |
| DEM | Digital Elevation Map |
| OS | Operating System |
| RAM | Random Access Memory |
| AGP | Accelerated Graphics Port |
| PCI | Peripheral Component Interconnect |

# CHAPTER 1
# INTRODUCTION

The field of data visualization is concerned with the use of visual representations of abstract data to facilitate human exploration and understanding of patterns and structural relations in the abstract data being displayed, as part of a cognition, hypothesis building, and reasoning process (see Figure 1–1 for examples). Terrain visualization incorporates visual representations of georeferenced data with a visual representation of the geographic region where the data originated. Computer based visualization systems have been used for several decades for both general data and terrain visualization scenarios. In order to construct a realistic visual representation of the geographic region of interest, a visualization system incorporates digital representations of the geographic region surface with corresponding satellite or aerial images, and combine them using 3D Computer Graphics techniques to produce the desired terrain visualization.

Performing 3D visualization in a computer requires working with a mathematical description or model of a 3D object. The most commonly used 3D model is the polygonal model, in which a set 3D vertices are interconnected, forming triangles or polygons which describe the 3D surface of the object of interest. To create a 3D visualization of an object, the corresponding 3D model is processed to produce a 2D image in which the 3D nature of the object is evident. This process, called 3D rendering, is accomplished via a series of steps which are performed in sequence and are generally referred to as the 3D rendering pipeline. The general steps performed as part of the 3D rendering pipeline are the following [1]:

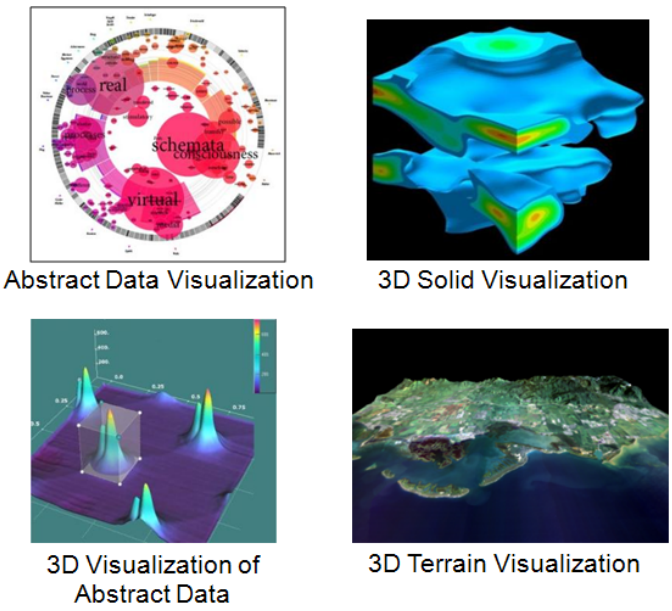



Figure 1–1: Different examples of visualizations.

- Define each object geometry in a coordinate system local to each object (model space).
- Transform each object from its local coordinate system into the scene's global or world coordinate system (world space).
- Transform the complete scene geometry from the world coordinate system into the camera or view coordinate system (view space).
- Transform the scene geometry from the view coordinate system into the screen coordinate system (screen space) based on the perspective projection.

Figure 1–2 describes the various general transformations, processes and spaces involved in the 3D rendering pipeline. Each transformation is accomplished via a series of translation, rotation and scaling operations performed on the 3D geometry being processed at each particular step of the pipeline. In addition to the operations which transform the geometry between spaces, other operations are performed in each particular space. The most significant of these operations are: the light and surface specifications performed in world space; back face culling (removal of geometry facing away from the camera) performed in view space; view frustum culling

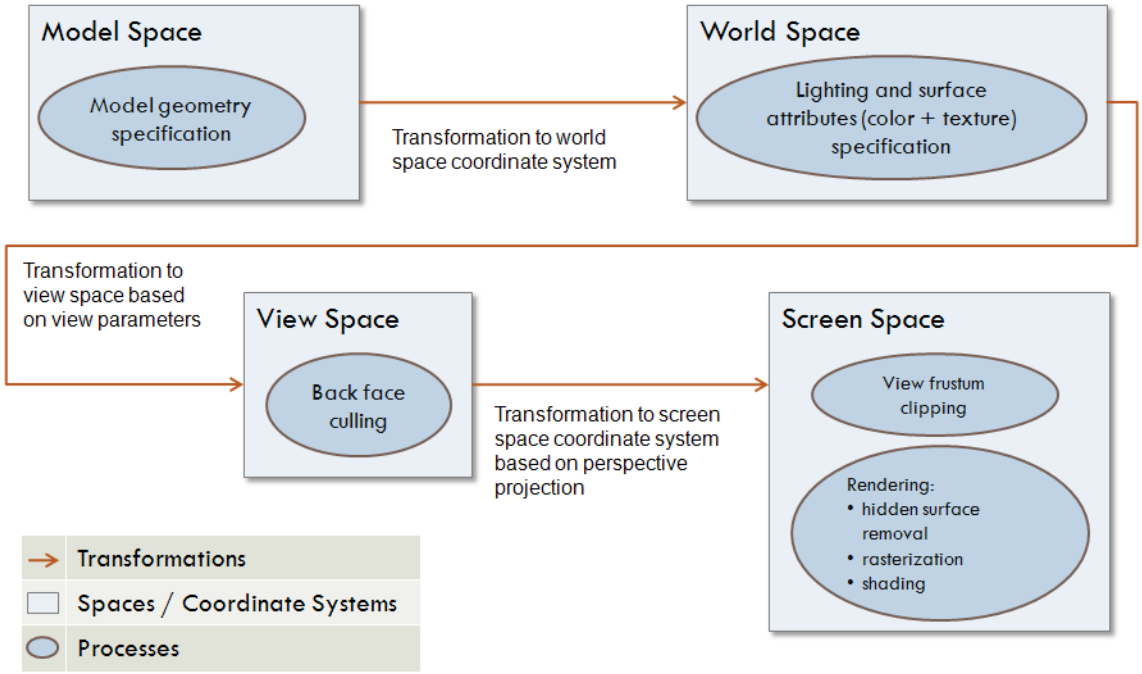



Figure 1–2: 3D Rendering Pipeline.

(removal of geometry outside of the projected viewing area) and hidden surface removal performed in screen space; and finally the rasterization and shading process in which the final pixel color value is calculated for each visible object in the scene.

The perspective projection is performed from the position of a virtual camera, which determines 3D region (view-frustum or viewing-volume) of interest to be rendered. The actual calculation required to perform the perspective projection are implemented by a series of matrix transformations applied to each vertex in the model. Hardware-accelerated support for performing operations related to 3D rendering is available in consumer-level video cards (GPU) and accessible via application programming interfaces (API).

Interactive visualization refers to a type of computer-based visualization in which: a) the user of the visualization system controls some aspects of the visualization being performed, and b) the changes resulting from the user input are incorporated into the visualization in a timely manner. Performing analysis through interactive visualization is qualitatively different from the study of a sequence of

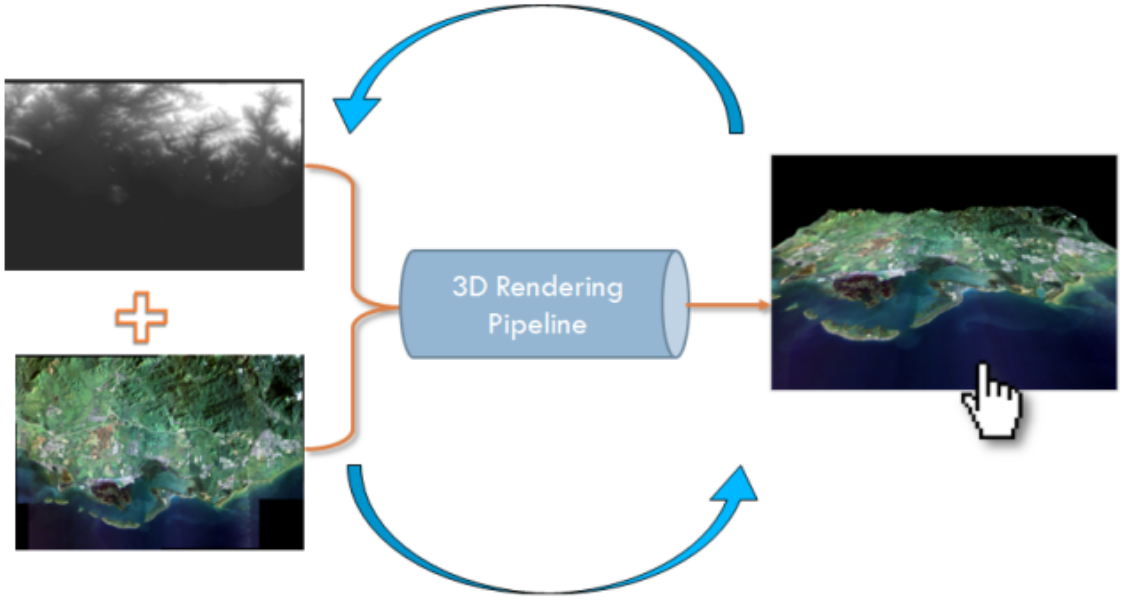



Figure 1–3: Interactive Terrain Visualization.

static images, due to the immediate feedback to user interaction which allows for a more dynamic discovery process [2]. Figure 1–3 presents a conceptual representation of the process required for interactive terrain visualization.

## 1.1 Performance Goals

For many general applications, computing system performance has been characterized in terms of either response time (also referred to as execution time or latency) which measures how long does it takes to finish a job, or throughput which measures how much work is accomplished per time unit [3]. In the case of 3D interactive visualization system, the two primary measures of performance are the visual accuracy and the rendering frame-rate. Visual accuracy is closely related to throughput while frame-rate is more closely related to response time. Traditional non-interactive 3D rendering systems are designed with visual accuracy as primary performance goal. This type of visualization system performs batched processing of 3D data which, depending on the dataset and rendering parameters, may take hours, days, or weeks to complete. On the other hand, interactive visualization systems have been traditionally characterized as soft real-time systems, and optimized primarily for rendering frame-rate. In the following sections we discuss the different

issues associated with both visual accuracy and interactive rendering frame-rate, in the context of interactive 3D terrain visualization.

### 1.1.1 Visual accuracy

The first criterion, visual accuracy, implies that the visual representation displayed to the user must correctly represent the underlying data being explored. In the ideal case, the visual representation will always express all information contained in the actual data. However, this may not always be possible due to limitations of the available hardware technology to completely express the full range of the datasets. This limitation has two general manifestations:

- Inability of the hardware to express the full range of potential values for any element in the dataset.
- Inability of the hardware to store the complete dataset.

The first manifestation of the problem can result in a mismatch between the range of the underlying data and the range of available visual elements used to represent it. In the simplest example, digital display technology impose limits in terms of the number of colors available for display, typically 24-bit or 16,777,216 spread on 3 bands (red, green and blue, generally referred to as RGB) each consisting of 8-bit or 256 colors per band. In the case of a dataset consisting of discrete values with variability that exceeds 16,777,216, it would not be possible to attempt a one-to-one mapping of data values to colors. In the case of terrain visualization applications, this is not a current concern thanks to the characteristics of the datasets and the way the data is visually represented. Terrain datasets consist of satellite images and elevation maps which are combined in a 3D visual representation using perspective projection. The mapping of color elements values to the display color values is limited either by the range of values produced by the data acquisition system or by the range of color perceptible by the human eye. The mapping of elevation values

to display elements is handled by the perspective projection in which data elements or samples are represented as occupying a position in 3D space.

The second manifestation of how hardware limitations make difficult the completely express of the dataset accurately is related to the dataset size and the storage capacity available in the system. Visualization systems usually operate with large datasets consisting of millions or billions of data elements. [2] defines large datasets as those that are considerably larger than the main memory capacity of desktop computer. More specifically:

$D >> 10 * M_d$

where $D$ is the size of dataset of interest and $M_d$ is the main memory capacity of the desktop computer. Even larger dataset may also exceed the disk storage capacity of current consumer-level desktop computers. In such scenarios, the hardware will be incapable of displaying all the elements of the dataset at the same time[1] . The memory capacity limitations of the hardware are still a concern in terrain visualization applications since advances in data acquisition make larger and larger datasets possible.

### 1.1.2 Interactive frame-rate

The second performance goals of interactive terrain visualization is sustaining interactive rendering frame-rate, that is, being able to draw or render the 3D visualization at a rate in which interactive visualization is possible, without noticeable latency being present between the user-driven action and the visual update in the visualization screen. As mentioned earlier, in order to render terrain data, every

[1] In many cases it may not be desirable to present the full range of the dataset, even when the hardware is not a limiting factor. First, for considerably large datasets, this approach may result in information overload, where there is simply too much information to be visually digested. Second, in many cases not all the information is different or significant enough in terms of either data variability or number data elements to be useful.

vertex in the terrain geometry must be transformed (translated, rotated or scaled), shaded depending on light position and finally perspective projected, transforming the vertex information to the corresponding pixel in the visualization screen. The result of this process is a rendered frame of animation. Achieving interactive frame-rates is affected by two hardware-related factors:

- The hardware input/output capabilities.
- The hardware computational capabilities.

The I/O capabilities of the hardware limit how much times it takes to transfer the data between memory storage components in the interactive visualization system. As the dataset size increases, more elements need to be transferred as part of the rendering of an specific frame of animation.

The hardware computation capabilities limit how much time it takes to perform all the vertex transformations and light calculations required to render all the geometry in a frame of animation. Rendering a frame is computationally intensive and is usually performed several times per second. In general, the larger the number of frames rendered per second the smoother the animations is perceived by the user. Rendering systems typically aim for a target frame-rate between an average of 10 to 60 frames per second.

## 1.2 Performance Problems

Building an interactive terrain visualization tool present a series of challenges related to the performance and scalability of the application. Performing interactive rendering implies that user interaction should provide immediate changes in the visualization. In addition, we want to provide an accurate representation of the geographic region being visualized. The more accurate the representation is, the more complex and detailed the digital representation will be, which increases the data set storage space requirement. As the size of the dataset increases, the more difficult it becomes to be able to render the it interactively. The size and complexity

of the dataset may be too high for the CPU and the GPU to process and render at an interactive frame-rate. The terrain dataset may be too large to store in conventional memory or even in hard-disk and the required data transfer may stress the I/O handling capabilities of the computer to the limit. High-capacity storage servers can be used to alleviate this problem, however, it introduces both additional complexity in terms of data management and network communications overhead. Relying solely on hardware improvements to solve the previously mentioned performance problems is costly and generally insufficient, since the hardware may not necessarily scale well for interactive visualization applications [2].

## 1.3 Previous Solutions

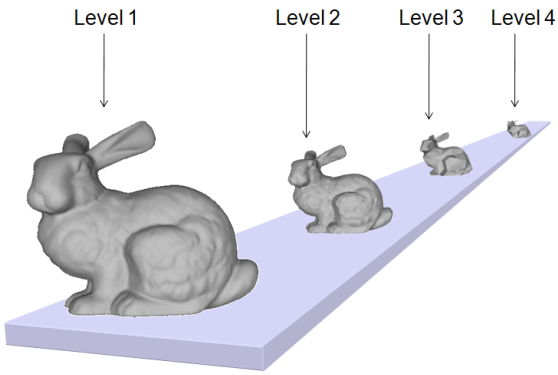
Performance optimization related to the interactive 3D visualization of both arbitrary 3D geometry and terrain geometry have been an active area of research for several decades [4]. As a result, several solutions to the various performance bottlenecks observed in specific parts of the rendering systems have been presented in the in the literature. We categorize these solutions as:

- graphic processing unit optimizations (GPU)
- multi-resolution or level-of-detail rendering algorithms (LOD)
- external memory management algorithms (EMM)
- data streaming techniques (DS)

In the following section we briefly introduce each approach. A more in-depth review of the most significant techniques related to each approach is presented in the Chapter 2.

### 1.3.1 Graphic Hardware Optimizations

Current graphic processing units (GPU) provide flexible programmable capabilities for graphics rendering which allow more control over the rendering pipeline than what was available using high-level graphic APIs. As consequence, terrain visualization tools can take advantage of capabilities provided by the GPU to improve




| Level | # Polygons | Image |
|---|---|---|
| 1 | 69,451 | |
| 2 | 2,502 | |
| 3 | 251 | |
| 4 | 76 | |

Figure 1–4: Four level-of-detail representations of the same model.

rendering performance. GPU rendering optimization range from off-loading part of the data management computations to the GPU, adapting existing data management techniques for GPU-based implementations, or devising new GPU-based techniques specifically tailored around the strengths and limitations of the available GPU architectures.

### 1.3.2 Level-of-Detail

Rendering level-of-detail (LOD) algorithms regulate the amount of detail or resolution of the terrain geometry to improve rendering performance, while satisfying a geometric and/or visual fidelity error metric in order to maintain a faithful representation. The basic concept behind LOD [5] is to use less detail for small, distant portions of the scene to be rendered (see Figure 1–4). This insight is based on the observation that it is inherently inefficient to use many polygons to render object that will only contribute to a few pixels of the rendered scene. For clarity, we will refer to a level of detail management algorithm as a LOD algorithms, and we will refer to the polygonal mesh of a given detail resolution produced by the LOD algorithm as a LOD.

### 1.3.3 External Memory Management

In order to handle datasets that exceed the size of available main memory, an out-of-core or external memory management (EMM) scheme must be employed. Conceptually similar to the paging mechanism employed at the OS-level, the EMM must allow the rendering system to operate in memory only with the subset of the terrain data required for the desired LOD at a given time. As the view position changes, data is loaded into memory as required.

### 1.3.4 Data Streaming

In many cases, the dataset size may also exceed the storage capacity of available disk storage, or the available disk space quota allocated for the user running the visualization application. To overcome this limitation, the external memory manager can be extended to support data streaming from remote data storage servers. Using a client/server pattern, the visualization application requests multiresolution terrain data from the servers, then stores it on a disk cache. The terrain data transmission is generally performed in a coarse-to-grain fashion, so a simple low-resolution model can be quickly displayed while more detailed data is been received. To optimize network transmission, a data compression scheme can be employed as well as data prefetching based on geographic spatial locality.

## 1.4 Problem Statement

Understanding the performance aspects related to data management and processing is a crucial aspect for the development of effective PC-based interactive 3D terrain visualization systems for scientific research. However, previous works have been focused on a specific performance bottlenecks and do not address the general recurrent performance problem of the timely movement of data throughout the complete rendering system (see Figure 1–5). Earliest efforts related to rendering performance optimization were focused on minimizing the number of triangles to be

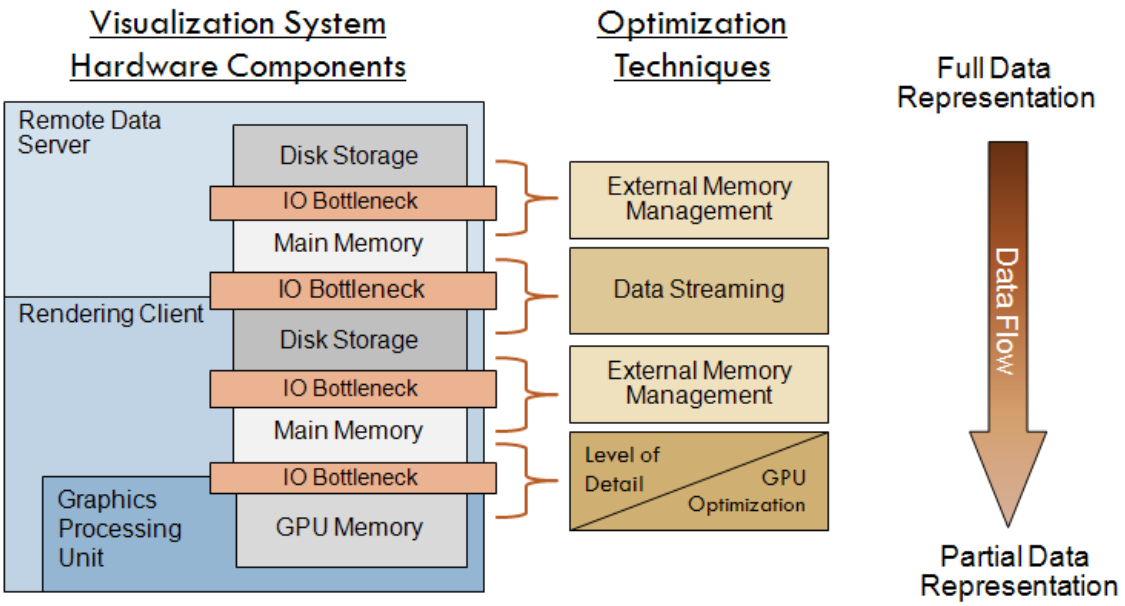



Figure 1–5: Hardware components, rendering optimizations techniques and data flow in interactive visualization systems

processed either by the CPU or the graphic hardware by performing selective geometric simplification during an off-line operation, at run-time or via a combination of both. To scale the rendering capabilities beyond the limitations of main memory, rendering algorithms were extended to incorporate external memory management capabilities in order to load data from secondary disk storage into main memory as required. Subsequently, improvements in networking technology opened the door for data streaming support in rendering algorithms to take advantage of remote, high-capacity data servers. More recently, graphics processing hardware optimization has reemerged as a focus of research due to both the improved processing capabilities, and the flexible programming capabilities of commodity graphic processing units. While extensive work has been done in each particular optimization area, and much work has also been done in various combined areas (GPU-LOD, LOD-EMM, and

EMM-DS), there has been very little research in the performance behavior of the complete data processing pipeline for interactive visualization systems[2] .

The existing situation presents a difficult scenario for system architects and engineers. In order to effectively design and develop efficient interactive terrain visualization systems, a general methodology for identifying the performance characteristics of the software and hardware components required to achieve a desired performance goal. As far as we are aware, such a methodology for the performance analysis of interactive terrain visualization systems has not been presented in the literature. To address this problem, we are proposing the use of a model for characterizing the performance of the interactive visualization system. The model should characterize the system as a data processing pipeline composed of multiple hardware component with specific performance parameters. Using the proposed model we can identify the effect of each potential bottleneck in the total performance of the system.

## 1.5 Goals and Objectives

The primary goal of this research was to develop a model for the performance analysis of interactive terrain visualization systems. The proposed model should help system architects and engineers to easily estimate the expected performance of different hardware and software components under considerations as part of the design, development and implementation of interactive terrain visualization systems.

---

[2] We must clarify the use of the term pipeline in the context of this work. Previously we referred to the "3D rendering pipeline", the sequence of steps performed either in software or hardware to transform the 3D data primitives and associated raster images into the final perspective projected image to be displayed. In this work we will use the term "data processing pipeline" to refer to the end-to-end terrain data management system which processes the terrain data from a complete, full resolution dataset at one end of the rendering system, to a partial and/or reduced resolution data subset to be consumed at the other end.

In addition, the proposed model could serve as a foundation for the future development of techniques for the automated selection of the ideal hardware and software components required for achieving an specific performance goal in an interactive terrain visualization systems. To that effect, we developed our proposed model with the following objectives:

- Integrate memory related aspects of terrain visualization that so far have been studied in isolation to obtain a holistic view of the performance of the system from end to end from which performance estimations can be performed.
- Identify which components related to the terrain data management and processing have the largest performance impact interactive visualization systems.

## 1.6 Proposed Solution

In this work we present our model for the performance estimation of interactive terrain visualization systems. Our model is developed as a particularization of the hierarchical memory model commonly used in computer architecture field for characterizing the memory subsystems in computing systems. In particular, we characterize each hardware components in terms of its memory storage and transfer capacity. Based on this characterization, we model the data transfer cost through the interactive visualization system as a function of the aggregated transfer cost of each component. The data unit employed in the model is based on a combination of terrain geometry and raster images. Data replacement between levels is determined by geographic spatial locality and view position. Optimizations techniques such as GPU, LOD, EMM,and DS are characterized as modifiers associated with a memory stage. Based on this model we estimate the total performance of the system and analyzed the impact of each performance bottleneck across the complete rendering system. We validate our model with a test case study of the WALSAIP Visual Terrain Explorer(VTE), a terrain visualization tool for exploration of georeferenced

data, which we developed as part of the Wide-Area, Large Scale, Automated Information Processing project. We selected various hardware platforms as well as a series of different datasets in order to study the performance of the VTE applications in different scenarios. Based on the hardware components of the hardware platforms selected, and using the model developed, we produce a performance estimated. The results obtained show that the model can characterize the general performance behavior of the VTE application running on the different hardware platform, however, the performance estimates is higher than the measured performance and diverges even further as the dataset size increases. We analyzed the results and presented various hypothesis as to why the model diverges from the measurements.

## 1.7 Contributions

In this research we present the following contributions:

- Developed a model for the analysis of interactive terrain visualization systems.
- Developed a methodology for mapping hardware/software components of interactive terrain visualization systems into the model.
- Developed an interactive terrain visualization applications called Visual Terrain Explorer.
  - Implemented in Java and OpenGL for cross-platform application development and deployment.
  - Supporting Level-of-detail, External memory management, and Data streaming.

In this chapter we have presented an overview of the different aspects related to the performance of interactive terrain visualization systems. In Chapter 2 we present a in-depth review of the literature related to the different techniques employed to improve the performance of interactive terrain visualization system. We also present an overview of existing models which have been used to characterize performance in computing systems. In Chapter 3 we present our proposed model

for the performance analysis of interactive terrain visualization. In Chapter 4 we describe the visual terrain application called *WALSAIP Visual Terrain Explorer* which we use to validate our model, as well as the results obtained. In Chapter 5 we present our conclusions and propose various potential future works.

# CHAPTER 2
# LITERATURE REVIEW

This chapter presents a literature review of the terrain rendering techniques previously mentioned in Section 1.3, as well as a review of data processing models. These two aspects are the basis of our proposed model.

## 2.1 Terrain rendering optimizations

### 2.1.1 Multi-resolution or Level-of-detail Rendering Algorithms

An excellent in-depth overview of multi-resolution or level-of-detail (LOD) rendering can be found in [4]. In this section we will summarize the most relevant aspects and considerations related to LOD rendering for terrain visualization.

A LOD algorithm is responsible for performing geometric simplification operation to eliminate redundant information where appropriate while satisfying both performance and visual accuracy constrains. The basic components of a LOD algorithms are the initial mesh, which consist of representation of the terrain at either the minimum resolution level or the maximum resolution level, and a set of updates operations, that, when applied to the initial mesh, produce a corresponding mesh of different resolution. Historically, the mesh with the minimum number of polygons required to approximate the full resolution terrain has been called the base mesh. We will refer to the full resolution representation of a terrain as the full mesh. Even though a LOD algorithm can begin with a base mesh (simplest representation) and progressively add detail, the overall process is still one of geometric simplification with respect to the original full mesh, since the resulting mesh is a simplified version

of the original. We will now describe the various characteristics which differentiate existing LOD algorithms.

**Transition Granularity between LODs**

A LOD algorithm can be characterized as either discrete or continuous depending on the transition granularity between successive LODs produced by the algorithm.

- **Discrete LOD:** In discrete LOD algorithms, multiple individual models of different resolutions are generated from the terrain data during an off-line processing operation, and the appropriate model is selected at runtime. Since the bulk of the geometric simplification is performed off-line, discrete LOD algorithms are simpler and their runtime overhead is usually limited to evaluating the selection criteria and handling transitions between LODs.
- **Continuous LOD:** In continuous LOD algorithms, a continuous spectrum of detail for the terrain is encoded in a data structure from which a model for a desired level of detail can be extracted at runtime. Continuous LOD algorithms are more complex than discrete LOD algorithms and also incur in considerable more runtime overhead. However, Continuous LOD algorithms are, in theory, more efficient since they provided more granularity between LODs, we should allow for more efficient resource utilization since the algorithm should only use as many polygons as required to achieve the desired LOD.

**Per-LOD Detail Distribution**

Depending on the distribution of geometric detail across the output mesh, LOD algorithms can be characterized as performing either a uniform or a non-uniform geometric simplification.

- **Uniform simplification:** In uniform simplification algorithms, the level of detail of a resulting mesh is uniformly distributed across the surface geometry. Uniform simplification is usually employed in discrete LODs generation which is performed

off-line when no view-dependent information is available. Uniform meshes are also generated by LOD algorithms that aim to optimize mesh layout for GPU processing.

- **Non-uniform simplification:** Non-uniform simplification is generally employed to produce meshes where the detail is selectively distributed across the surface geometry.It is generally performed for further reducing the number of vertices in the output mesh by only maintaining detail where required in the context of a specific LOD mesh and for supporting view-dependent simplification algorithms, where the level of details varies with respect to the view direction and the region of the mesh contained inside the view frustum.

**LOD Processing Direction**

As mentioned before, the geometric simplification operation performed by a LOD algorithm begins with using either the base mesh (simplest representation) or the full mesh (most complex representation) as the initial mesh. As a result, the use initial mesh selection allow us to classify the algorithms in terms of the direction in which the update operation introduces complexity into the resulting mesh.

- A top-down LOD algorithm (also referred to as refinement or subdivision algorithm) begins with a base mesh and then proceed to progressively add vertices, incrementing complexity, until the desired resolution is achieved. Top-down algorithms are ideally suitable for run-time operations since they complement view-dependant simplification algorithms by supporting view culling *i.e.*, discarding polygons that lie outside the view frustum.
- A bottom-up LOD algorithm (simplification or decimation) begins with a full mesh and proceed to progressively remove vertices (decreasing complexity) until the target resolution is reached. Bottom-up algorithms have higher memory and computational cost since they start with the full mesh, but are able to find the minimum

number of polygons for a given accuracy level. As a consequence, bottom-up algorithms are generally used during offline discreet uniform simplification operations.

Figure 2–1 illustrate the relation between the terrain refinement/simplification process and the processing direction.

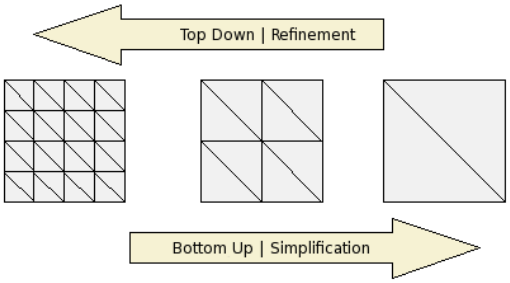



Figure 2–1: Top-down vs. Bottom-up processing direction.

**Terrain data structure**

LOD algorithms are also differentiated by the structure employed to represent the terrain. The basic two representations are height fields and triangulated irregular networks (TINs) [6].

- Height fields - Array of height values at regularly spaced x and y coordinates. In terms of management, height fields are easier to work due to their simple spatial organization. However, height fields are considered inefficient in terms of data size, since they store redundant information in regions with no change in elevation.
- TINS - In TINs, height values are irregularly spaced. TINs can use the minimum number of elevation samples to approximate a terrain by using few samples to describe large flat areas while using many samples on areas of larger terrain complexity.

**LOD data structure**

In order to implement view-dependent multi-resolution rendering a hierarchical structure capable of representing different parts of the terrain at different resolutions is required. The most commonly used structures for this purpose are the quadtree and the bintree [7]. In a quadtree structure, a rectangular region is recursively

subdivided into four uniform quadrants. More formally, [8] defines a quadtree for the set of points $P$ in a square $S = [x1 : x2] \times [y1 : y2]$ as follows:

- If $|P| <= 1$, then the quadtree is a single leaf where $S$ and $P$ are stored.
- Otherwise, let $Q_{NE}$, $Q_{NW}$, $Q_{SW}$, and $Q_{SE}$ denote the four quadtrees. Let $x_{mid} = (x1 + x2)/2$ and $y_{mid} = (y1 + y2)/2$, and define:
  - $P_{NE} = \{p \in P : p_x > x_{mid} \wedge p_y > y_{mid}\}$
  - $P_{NW} = \{p \in P : p_x \leq x_{mid} \wedge p_y > y_{mid}\}$
  - $P_{SW} = \{p \in P : p_x \leq x_{mid} \wedge p_y \leq y_{mid}\}$
  - $P_{SE} = \{p \in P : p_x > x_{mid} \wedge p_y \leq y_{mid}\}$

The quadtree contains a root node $v$, which contains $S$. Where $v$ has four children, and the X-child is the root of the quadtree of the set $P_X$, where $X|X \in \{NE, NW, SW, SE\}$.

A binary triangle tree structure (or bintree) use the same strategy but subdivide the initial rectangular region into two triangles. A bintree is in essence a kd-tree. A kd-tree is a binary tree that recursively subdivides a space such that a $k$-dimensional kd-tree divides a $k$-dimensional space with a $(k-1)$-dimensional plane [9].

The main advantages of bintrees over quadtrees are that it is easier to work with LOD transition artifacts [4]. Multi-triangulation [10] is an extremely general TIN data structure which stores a base mesh and the update operation required to refine it. Dependencies between update operations are represented by a direct acyclic graph which influence when simplification or refinement is performed. Another data structure commonly used is the pyramid structure (used by TerraVision [11, 12]), which is an adaptation of the texture mipmap technique [13] for geometry. Different LODs of a terrain geometry or texture are stored as different levels of a pyramid; the highest LOD at the bottom of the pyramid and each subsequent LOD is half the resolution of the previous. Figure 2–2 show the main data structures previously described: a) a quadtree of three levels. b) a bintree of four levels. c) and d)

alternate representations of a pyramidal structure where each pattern represents a different resolution.

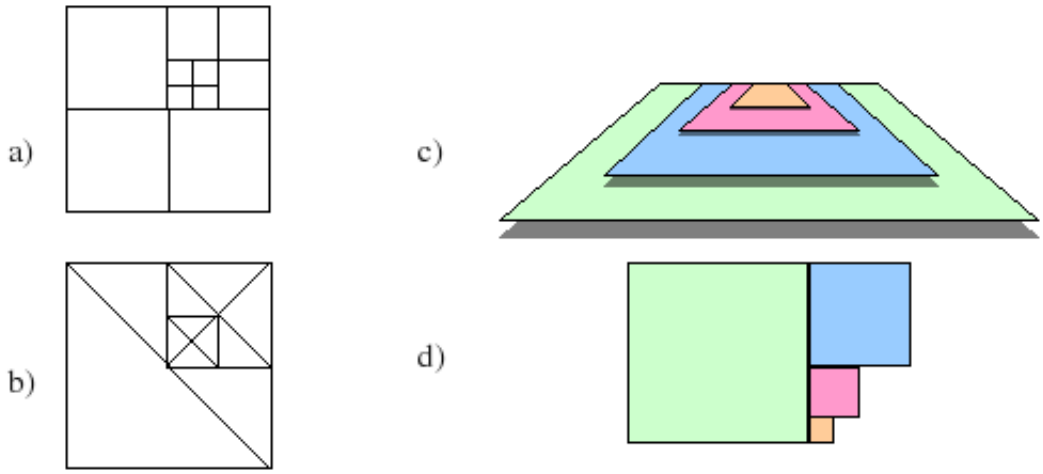



Figure 2–2: Hierarchical data structures.

**LOD selection**

Since the basic LOD premise states that less detail is required for small or distant elements of the scene, one of the most important aspects in a LOD algorithm is the criteria used to characterize an element as being small or distant, and how that criteria is used to select an appropriate LOD at runtime. The simplest criteria is distance, if an object is at a specific distance of the view position, it will be rendered using a particular LOD. An alternative criteria is the size of the particular element when projected into the screen, which in general is a more accurate way of selecting the LOD, but which is also more costly in terms of computation. Other parameters used in LOD selection may include view orientation and/or surface roughness. A more sophisticated approach proposed in [14] involves defining a rendering benefit and cost functions for each LOD type (discrete or continuous) and perform the selection on the benefit-to-cost ratio.

**LOD transition discontinuity (the popping effect)**

The most common problem faced by all LOD techniques is minimizing the temporal discontinuity that occurs as geometric complexity suddenly changes when

a LOD transition occurs. This effect is commonly referred to as the popping effect since it is particularly evident when a transition from a lower to a higher LOD causes more polygons to suddenly pop into view. The reverse effect is also common, as surface detail abruptly disappear when a higher to lower LOD transition occurs. The two most common approaches to this problem are examples of the inherent trade-off between performance and complexity present in LOD techniques. The popping effect can be easily eliminated by a) geomorphing (geometrical interpolation) between the two LODs as the view changes, maintaining visual continuity at the cost of the extra computation required to do so, or b) moving the LOD transition threshold farther away so that any potentially abrupt geometry change will occur before they can be perceived, due to the perspective projection. While this solution does not require extra computation, it will require maintaining geometric complexity beyond the point where it is perceivable, which goes against the basic premise of LOD.

**Terrain spatial discontinuity (surface cracks and tears)**

A problem generally faced when using quadtrees or any tile-based LOD approaches is that it is possible to introduce artifacts such as cracks and tears in the edges between LODs caused when a polygon from a higher LOD does not share a vertex or lies on an edge of a polygon in a lower LOD. Common strategies to eliminate this kind of artifact include: modifying the polygon involved by either adding a vertex at the lower LOD triangle or adjusting the vertex of the higher LOD triangle, introducing new polygons to fill the gap o subdivided both polygon to produce a more continuous transition at the cost of adding geometrical complexity, or just preventing simplification of vertices that lie on the LOD boundaries.

**Frame-to-frame coherence**

In general, continuous LODs algorithms try to minimize as much as possible the computation required during a simplification or refinement process. Fortunately, for interactive applications where no drastic change in the view position or orientation

occurs, it is possible to exploits the fact that for a particular region of the terrain, the LOD required during the next frame of animation will be either slightly lower (when moving away) or slightly larger when moving forward. This is known as frame-to-frame or temporal coherence. For a LOD algorithm to take advantage of frame-to-frame coherence, it must be capable of encoding the update operations (and the dependencies between each operation) in order to allow incremental updates to the last LOD instead of recomputing the new LOD from scratch.

**Review of LOD algorithms**

A review of the classic LOD techniques can be found in [4], here we summarize the most relevant classic works as well as new approaches found in recent literature. Lindstrom *et al.*[15] presented one of the earliest real-time continuous LOD algorithms. During an off-line operation, the original mesh is broken into a quadtree of height field blocks which contain discrete LODs. At runtime, an incremental top-down (coarse to fine) refinement is performed by traversing down the quadtree, followed by bottom-up simplification at the block level until the screen-space error criteria is reached. To exploit temporal coherence, an active cut of blocks is used to keep track of the current LOD. Extending his previous work on progressive meshes [16, 17], Hoppe presented a TIN-based, view-dependent terrain LOD algorithm [18]. In this technique, continuous LOD is produced by adding or removing triangles from off-line generated blocks of terrain. LOD selection is done based on view frustum, surface orientation, and screen-space geometric error. This technique uses memory mapping for out-of-core support.

The Geometrical MipMapping technique [19] adapts the texture mipmap [13] technique for geometric data. It employs height fields blocks of different LODs which are stored on disk for out-of-core operation and quad-tree of bounding boxes for view-frustum culling and block selection. Upon block selection, the vertex data is read from disk. To minimize CPU calculations, LOD transition are selected based

on minimum and maximum viewing distance which are precomputed with the worst-case camera angle (the camera angle from which geometric error is more evident).

CABTT [20] extends the bintree-based LOD approach to work with clusters of aggregated triangles. This reduces CPU overhead by performing view culling per cluster. In addition, since clusters stay fixed over several frames, they may be cached on the video card memory. QuickVDR [21] provide a general approach based on a cluster hierarchy of progressive meshes (CHPM) where each level of the hierarchy tree represents a different LOD. The hierarchy structure is used for visibility culling and each node or cluster consists of a progressive mesh which can be refined as required. The cluster itself is composed of a few thousand triangles with an associated bounding box which is built off-line. At runtime, clusters are split or merged as required. Zhu [22] presents a hybrid technique where irregular meshing is used to construct an input mesh for a uniform simplification process. This technique exploits the flexibility of an irregular mesh to produce better approximations of the original mesh, and the ability of uniform simplification to produce regularly-connected meshes which are ideal for creating triangle patches optimized for graphic hardware processing.

The Geometry Clipmap [23] presents a LOD algorithm based on the texture mapping technique, but implemented on the GPU. The basic algorithm works by building a multi-resolution mesh from the combination of nested concentric grids center about the view position, each grid from different level of LOD pyramid and sorted by decreasing resolution. As the view position changes, each nested grid is shifted to maintain the terrain LOD uniformly distributed around the view position and new data is paged into memory to fill the update region where LOD transition occurs. This technique integrates geometry and texture LOD and also supports terrain compression and synthesis. A different GPU-based approach is presented in [24], based on the progressive streaming of discrete mesh elements to the GPU. A

nested mesh hierarchy of discrete LODs is built off-line. At run-time, view frustum culling is performed on the CPU and tiles identified as visible are sent to the GPU where they are interpolated in order to obtain a continuous LOD. A specialized memory manager is used to keep track of which data is on the GPU memory and is used to prevent unnecessary data transfer.

### 2.1.2 External Memory Management or Out-of-core Rendering Algorithms

The general approach used for external memory management in terrain rendering applications is to subdivide the terrain geometry into blocks or tiles which are loaded when required. One of the simplest approaches is to exploit OS based system calls for mapping disk files to memory (memory mapping) in order to access terrain data. This approach delegates all the paging control to the OS, which is presumably more robust and efficient, but requires optimizing the terrain data on-disk layout (usually in coarse-to-fine order) and/or devising an algorithm to efficiently map terrain coordinates to the vertex data locations on disk. Other approaches depend on maintaining a mapping of nodes from a quadtrees or bintrees, or levels in a pyramidal data structure, to terrain tiles on disk. More specialized techniques may incorporate space location and LOD constrained access patterns into the data manager design for a more efficient mapping of multi-resolution terrain data to external memory.

Hoppe presented a technique [18] were terrain data being used as part of a progressive refinement process is partitioned and clustered into square tiles which are loaded using OS provided services. Lindstrom *et al.*[25] follow a different approach by optimizing data layout for memory coherence. The multiresolution terrain data is linearized in disk and accessed via memory-mapped file mechanism provided by the OS. Arguing that OS based services are suboptimal for paging large multiresolution terrain data, [26] presents a clustering technique based on object space and

LOD constrained access patterns which aims to minimize page faults and reduce loading time. The technique presented by Cignoni *et al.*[27] is based on a hierarchical geometric partitioning using a global indexed representation of the original huge mesh. A mesh portion is represented in core memory with indexed lists containing only the loaded vertices and faces. Partial data load/update/write-back operation are supported via automatic on the fly re-indexing of the loaded data portion. Vertex duplication between adjacent triangles is eliminated due to the global indexing scheme where a given vertex is referenced by only one index.

### 2.1.3 Terrain Data Streaming

Supporting terrain data streaming in terrain visualization aims to take advantage of remote high-capacity storage servers while minimizing the data transferred between the client and the server to compensate for the general performance issues related to network latency and bandwidth. Several works [28–30] has been done related to the compression of 3D geometry for multiresolution rendering. [31] address the neighbor dependency constrain in multi-resolution rendering techniques and proposes and scheme for the selective and progressive terrain data transmission by minimizing neighbor dependencies. In this scheme, the server sends the base mesh nodes first, and progressively transmit the other nodes depending on view parameters and error metric constrains. Kalaiah *et al.*[32] propose using statistical representation of the geometry in order to improve the geometry bandwidth bottleneck. Their approach is based on a hierarchical partitioning of the geometry into compact nodes constructed via clustering-based hierarchical principal component analysis (PCA) of the point geometry. Geometry decoding is performed by using a quasi-random sampling based on the probability distribution derived from the PCA analysis. [33] presents an overview of a proposed client-server architecture for terrain data streaming. Data transfer to the client is reduced by visibility culling and data caching based on the Least Potentially Visible scheme. Clientside prediction is used

to drive data prefetching. Terrain data representation is similar to [23]. The transmission scheme allows for the transmission of deltas between different resolution levels for low-end clients scenarios.

### 2.1.4 GPU Optimizations

The improvements in GPU technology has allowed for the re-evaluation of the performance optimizations traditionally required as part of interactive 3D rendering, particularly so, for the case of terrain visualization. One of the earliest works to incorporate modern GPU characteristics into terrain rendering problem is the the GeoMipMap technique [19]. The basic difference from earlier LOD techniques is the change in emphasis from finding the "perfect set" of triangles for a particular resolution to minimizing CPU utilization and maximizing GPU throughput. Thus, the algorithm sends denser mesh geometry to the GPU (compared to previous algorithms), requiring less calculation on the CPU, while keeping the GPU working as much as possible. The GPU optimization approach adopted by Yoon *et al.*[21] is to cache triangle patches or clusters in the GPU memory. Using *edge collapse* as triangle simplification allows their algorithm to produce a simplified mesh which is a subset of the original mesh. This procedure can be performed by manipulating the vertex connectivity of the geometry already in GPU memory, minimizing the data transferred from main memory. Geometry data is only transferred to GPU memory when the view parameters changes and new geometry regions become relevant or to fix cracks or tears introduced during the simplification operation.

Losasso *et al.*[23] present a GPU LOD technique that abandons any view-dependent re-meshing in order to minimize CPU utilization. The technique uses concentric, uniformly tessellated, square patches (called geometry clipmaps) centered around the camera and dropping exponentially in resolution as the distance increases. During run time, geometry is fetched from a toroidal buffer residing on the GPU, which is updated by the CPU when required. In [34], terrain mesh is

decimated and stored in a nested mesh hierarchy where new vertices from finer resolutions only have to be progressively transmitted and morphed in height onto the previous coarser level already available in GPU memory.

The major limitation of graphics hardware is the lack of geometry generation. GPUs are designed to efficiently rasterize huge polygons lists sent through the bus by the CPU, but are not capable of generating new polygons. This effectively have forced GPU LOD algorithms to operate by means of mesh simplification operations, which require sending geometry at full resolution through the bus, and simplified once it is already stored in the GPU memory. LOD algorithms based on mesh refinement, should provide the additional advantage of minimizing data transfer even further, since such algorithms start from a course level and progressively add vertices to achieve the desired mesh resolution.

Boubekeur *et al.*[35] present a GPU based approach in which a uniform mesh is sent to the GPU memory and manipulated by vertex displacement using vertex shaders. This approach allows for the arbitrary relocation of vertices (and thus, simulate vertex insertions). However, while this approach minimizes the memory footprint of mesh geometry in GPU memory, the actual overhead incurred while transferring the vertices into the desired final mesh make this technique generally slower than LOD techniques based on mesh simplification. It also less intuitive and more complex than other GPU-based approaches.

### 2.1.5 End-to-end Performance Analysis and Optimization

The study of the performance impact of data management from end-to-end in remote visualization systems is a new area of research. A recent work by Sisneros *et al.*[36] tackles the problem of dynamically identifying the optimal cache configuration which minimize absolute latency in remote visualization systems where multiple caches nodes are linearly located between the rendering client and the data server. The approach presented in [36] is based on the notion of an *incremental*,

which is the portion of data to be delivered or pulled through a chain of networked caching nodes. Each incremental is identified by a *compound key* composed of an arbitrary number of indices corresponding to the client request (view parameters, mesh resolution, etc.). Each compound key is encoded as a string to simplify the management of incremental requests by means of hash tables. Cache node configuration was define by three parameters: cache size (defined in number of bytes), prefetch size and delete size (both defined in number of incrementals). For model validation, 200 configuration were selected based on the following guidelines: cache size were distributed between 8 kilobytes and 128 megabytes to match real-world scenarios. Deletion size were selected as a random percentage of the cache size. Prefetch size was limited to a range between 0 and 6 to minimize data transfer cost. Each configuration was tested with 60 random requests. Based on the analysis of the configuration parameter space, the authors opted for numerical minimization as a heuristic search for better cache configurations. They used two procedural algorithms, Steepest Descend and Conjugate Gradient, to determine the direction of the minimization, moving from a cache configuration to another as long as the absolute latency is reduced in the process.

## 2.2 Data Processing Models

In terms of using current hardware for interactive visualization applications, the primary problem is still one of memory management [2]. To understand performance aspects related to data movement through the different memory components in a visualization system, we can incorporate two closely related models which have been previously used to analyze the performance or search for optimal configurations in multistage systems. These models are the *pipeline model* and the *hierarchical memory model*.

### 2.2.1 Pipeline Model

The optimization of multistage pipelines have been a focus of extensive research for scientists and engineers[37].

The classical problem is that of a simple, straight line transmission pipeline (also referred to as a gun-barrel pipeline) with $n$ compressor stations and a specified flow. Different pressures can be specified for $N$ points, which are located at the beginning of the pipeline, between each compressors and at the end of the pipeline. Figure 2–3 illustrates a pipeline consisting of $n$ stages (squares) and $N = n + 1$ decision points (diamonds). These $N$ pressure set points are the decision variables, and with a discrete set of $m$ possible values. Thus, the number of possible configurations for this pipeline system is $m^N$, which can be a huge number (for instance if $m = 20$ and $n = 10$, there are over $2 * 10^{14}$ possible configurations).

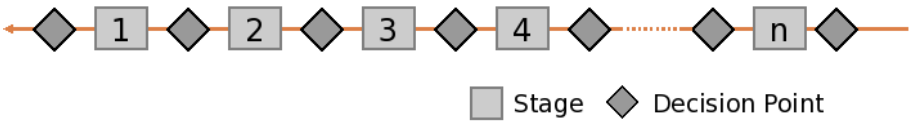



Figure 2–3: Multi-stage pipeline.

Identifying the optimal configuration is a combinatorial optimization problem, which can be attacked via meta-heuristics such as local search, simulated annealing, swarm intelligence, tabu search, genetic algorithms, neural networks [38].

### 2.2.2 Multilevel Memory Hierarchies or Hierarchical Memory Model

We turn to the concept of memory hierarchies whose motivation was clearly presented in the following quote by Burks, Goldstine and von Neumann [39]:

> "Ideally one would desire an indefinitely large memory capacity such that any particular... word would be immediately available... It does not seem possible to physically achieve such capacity. We are therefore forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible."

A hierarchical memory model (HMM) describes a random access machine whose memory access time is non-constant and determined by the non-decreasing access cost function $f(x)$. A typical access cost function is $f(x) = \log x$. Hierarchical memory can be visualized as a set of discrete levels (see Figure 2–4) where each level is defined as a continuous array of memory elements $A_m ... A_n$ such that $f(A_i) = f(A_{i+l})$, for $m \leq i < n$, and $f(A_{m-1}) < f(A_m)$ and $f(A_n) < f(A_{n+l})$. For $k \geq 1$, each level $k$ contains the $2^{k-1}$ locations at addresses $2^{k-1}$, $2^{k-1} + 1$, . . ., $2^k - 1$. For example, if $f(x) = \log x$, access to any location on level $k$ takes time $\approx k$. This model mimics the behavior of memory hierarchies consisting of increasingly larger amounts of slower memory.

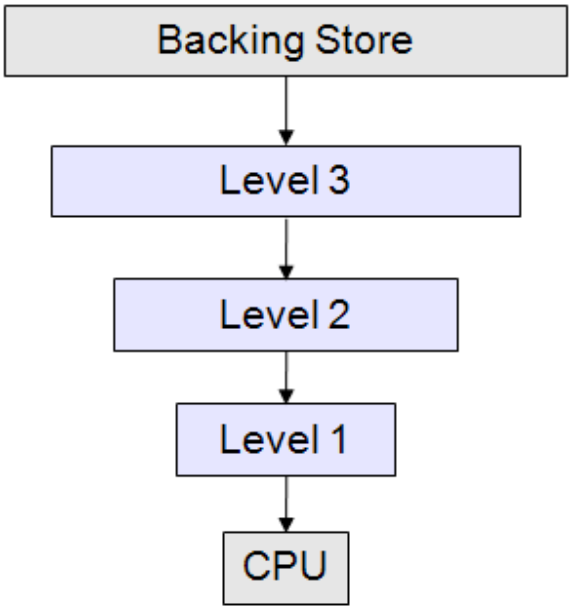



Figure 2–4: Example of five-level memory hierarchy.

In general terms, the HMM is a specialized pipeline model where the set of possible values for each decision variable increases at each subsequent stages. In addition, the HMM is selective in terms of data flow, *i.e.*, which data elements are transferred between each level. In contrast, the traditional pipeline model does not consider individual elements in the pipeline flow, only the total movement of identical elements between endpoint. Since both of these characteristics are pertinent for the

problem of data management in interactive terrain visualization systems, we choose to based our model primarily on the HMM, while reusing the conceptual form of the pipeline model in some cases due to its convenience for expressing composition and decomposition of stages.

In this chapter we have presented a review of the most significant techniques and approaches related to performance optimization of interactive terrain visualization systems. We also presented an overview of a previous performance study focused terrain visualization systems with cache nodes between the visualization client and the remote server. Finally we reviewed existing models which can be used study aspects related to performance in computing systems. In the Chapter 3 we incorporate all these aspects into our proposed model.

# CHAPTER 3
# HIERARCHICAL MEMORY MODEL FOR TERRAIN DATA MANAGEMENT

So far we have reviewed different aspects related to the performance of interactive terrain visualization systems, previous work related to performance optimizations, and existing models which have been used to characterize the performance of data processing systems. In this chapter we describe our proposed model for characterizing the performance of interactive terrain visualization systems. In Section 3.1 we describe the goal of the model. In Section 3.2 we present how interactive terrain visualization systems differ from traditional computing systems in the context of the proposed model. Section 3.3.1 describe two different representations employed for the conceptual presentation of the model. In Section 3.3 we describe the proposed model in detail.

## 3.1 Model Goals

Our goal is to estimate the performance of different components in an interactive visualization system and how the each component impacts the performance of the whole system. As described earlier, the most critical performance problem of interactive terrain visualization systems is related to the processing of the terrain data. As a result, our model must be capable of characterizing the data processing behavior of the system and in corporate the system characteristics most relevant to the performance.

## 3.2 Extending the HHM for the terrain visualization problem

Extending the hierarchical memory model for the analysis of interactive terrain visualization systems requires adapting the original model to incorporate details that are particular to the terrain visualization problem domain. First we must describe how an interactive terrain visualization system differs from a general computing system. The differences between an interactive terrain visualization system and a general computing system are primarily related to the characteristics and access pattern of data being processed.

### 3.2.1 Data Characteristics

In a general purpose computing system, a memory access is either a read operation or a write operation, in which a data unit consisting of a number of bits is transferred from or to memory. The write operation requires two inputs, a) the memory address to write to, and b) the data unit to be written. The read operation (see diagram a) in Figure 3–1) requires one input, a) the memory address to be read, and b) one output, the data unit read from the input memory address. At the conceptual level, the definition of a data unit can be considered an implementation detail. However, in concrete systems, it is usually specified at the Instruction Set Architecture level by the instruction being used, and is typically a multiple of the basic width of the system memory banks (common sizes are 8, 16, 32 and 64 bits).

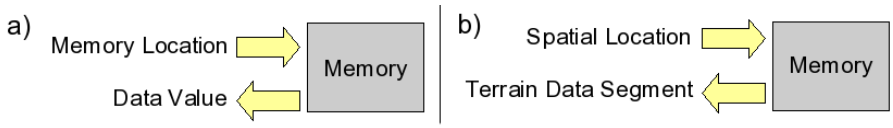



Figure 3–1: Read operations in a) general-purpose computing systems and b) terrain visualization systems.

Interactive terrain visualization systems operate over digital terrain geometry, raster images, and auxiliary data such as surface normals and texture coordinates. Access to the terrain data is dominated by read operations. Write operations to terrain data exists primarily in two particular cases: supporting interactive manipulation of terrain geometry, and systems which alter the terrain geometry for

performance reasons such as algorithms performing mesh refinement or geomorphing. In this research we choose to consider only the read operations. The first case of write operations is still fairly uncommon in highly interactive visualization scenarios, and the second case is only temporary in nature and tied to a particular stage of the rendering system. Read operations (see diagram b) in Figure 3–1) of terrain data consist of one input, the spatial position, and one output, the terrain data unit for the specified spatial position. The definition of a data unit can vary greatly between terrain visualization systems. The two primary components of a terrain data unit are geometry and images. The geometry data unit is generally a set of one or more vertices, which represent the terrain geometry precisely at (in the case of just one vertex), or around the spatial position requested (in the case of multiple vertices). The image data unit consists of raster images composed of $n \times m$ pixels of 24-bits (RGB bands of 8 bits each). The sizes of the image data can vary independently of the spatial range of the geometry. That is, raster images of different sizes can be used as texture for the same surface geometry. While images of larger resolution (which are larger in number of pixels) are preferable due to their improved visual quality, no clear criteria exist for selecting the image size for a particular geometry. For this reason we cannot generalize beyond the fact that the size of a raster image pixel is 24 bits.

### 3.2.2 Data Access Pattern

In terms of data access patterns, general computing system are designed for random data access, but three memory access patterns related to locality of reference are common due two the nature of how programs are structured, stored, and executed from memory. The three access patterns are: a) sequential locality, which dictates that memory locations accessed tend to follow a sequential order; b) spatial locality, which dictates that memory locations which are close to a previously

accessed memory location will probably be accessed in the future; and c) temporal locality, which dictates that memory locations previously accessed tend to be requested again in the future.

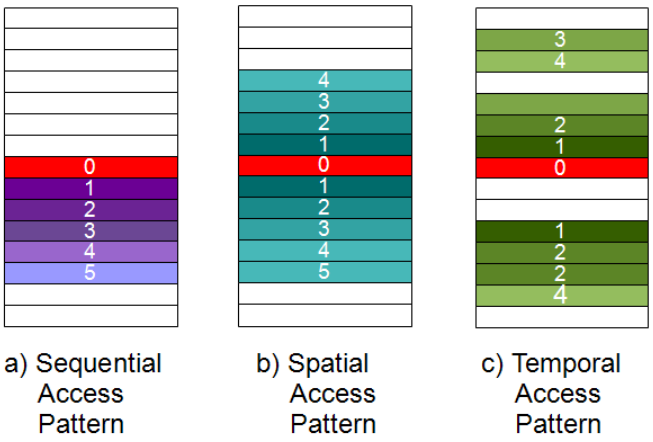



Figure 3–2: Data Access Pattern - General Computing System

Figure 3–2 illustrates each access pattern. In each diagram, different shades represent the expected sequence of future memory access, where darker shades occur before lighter ones.

In relation to the terrain data being managed, terrain visualization systems follow a data access pattern of geographic spatial locality. In other words, if a spatial location was accessed recently, other spatial adjacent locations will be requested in the future.

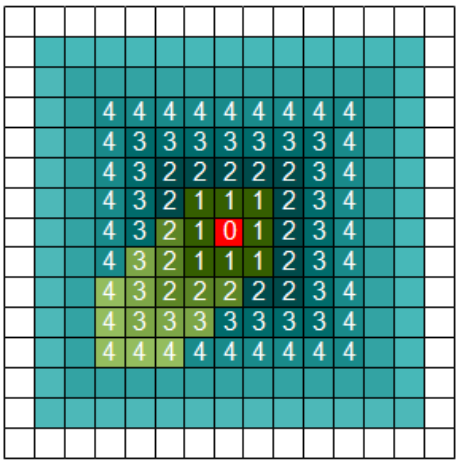



Figure 3–3: Data access Pattern - Terrain Visualization System

Geographic spatial locality is illustrated in Figure 3–3, in which green shades represent previous data request and blue shades represent potential future requests. A secondary access pattern occurs when dealing with multiresolution data. If the data for spatial position at a particular resolution was accessed recently, data of a level higher or a level lower of resolution may be requested in the future.

## 3.3 Model Description

### 3.3.1 Conceptual Representations

As part of the descriptions of the model we employ two different visual representations:

- Component representation.
- Data transfer representation.

These two representations allows us to focus in different aspects of the system as part of the model development process.

**Component Perspective**

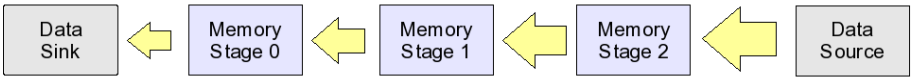



Figure 3–4: Visualization System - Component Representation

When using the component representation (see Figure 3–4), we focus on the dependencies and ordering of the components in a terrain visualization system. We define components as a hardware element in a visualization system which is capable of storing, transferring, and optionally, processing data to various degrees. Each component has a set of characteristics which affect the data transfer performance of the component and the performance of the system as a whole. Graphically, we present the components organization similarly to a hierarchical memory model, but presented in horizontal fashion similar to pipeline diagrams. This representation allows us to focus on the decomposition of the system, which is naturally oriented around components.

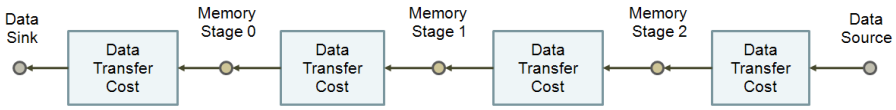



Figure 3–5: Visualization System - Data Transfer Representation

**Data Transfer Representation**

The goal of this representation is to represent the modeling of the data transfer in the interactive visualization system in terms of the component characteristics (see Figure 3–5). The total data transfer of the system is a function of the transfer cost between every two adjacent components in the system. The data transfer cost between two adjacent components is a function of the hardware characteristics of both components. We describe the modeling of transfer cost formally in sections 3.3.3 and 3.3.4. The data transfer representation allows us to focus on the performance of the system in terms of the data transfer between components and the characteristics of each components.

**Element Ordering Constraint**

In general, different stages or levels in a system will be organized according to the following general rule. Let:

- $E$ be the ordered set of elements of the system.
- $|E| = N$ be the number of elements in set E.
- $Z_N = 0, 1, 2, ..., N-1$ be the set of indices to the elements in set E.
- $C$ be the set of performance characteristics associated to an element $e_i$ where $e \in E$ and $i \in Z_N$.
- $v_i$ be the value of a performance characteristic $c$ of element $e_i$ be given by function $v_i = F(c, e_i)$ where $c \in C$, $e_i \in E$ and $i \in Z_N$.

then:

- $F(e_i, c) < F(e_{i+1}, c)$ where $c \in C$, $e_i \in E$ and $i \in Z_N$.

In other words, between two adjacent elements in the system, the lower element has a lower value for each of the performance characteristics than the next, higher

element. In both, the component and data transfer representation, leftmost elements correspond to lowest values, rightmost elements correspond to highest values. Note, that this constraint is defined only in terms of the numeric values of a performance characteristic, not on the interpretation of the value as being high or low in terms of performance. Lower transfer cost can be considered to give higher performance, while lower memory capacity may limit performance. In the context of this work, this translates to the interpretation that lower elements have lower data transfer cost and memory capacity and higher level has higher data transfer cost and memory capacity.

### 3.3.2 Data Unit Characterization

We define a terrain data unit as the basic unit of terrain data to be transferred through the interactive terrain visualization system. As mentioned earlier, terrain data consists of geometry and raster images. As a result, the basic terrain data unit should incorporate both geometry and image components corresponding to the same spatial region in the terrain dataset. In terms of geometry data, the most attractive choice for basic data unit would be a set of polygons, since they are the most commonly used type of primitive used in 3D rendering. However, since the actual polygon patch construction can vary due to different implementation choices (triangles *vs.* quadrilaterals, or strip layout *vs.* fan layout), the choice of polygon patch as basic terrain geometry data unit would bind the model to a particular implementation detail. In order to allow for flexibility in terms of the geometry primitives used to render the terrain, we define the basic terrain geometry data unit it in terms of terrain height samples. Similarly, for the case of raster image data, we define our basic image data unit in terms of image sample points or pixels.

Let $R$ be the rectangular spatial region of terrain with width $W_R$ and height $H_R$ described by geometry dataset $D_G$ and by raster image dataset $D_I$. Let $D_G$ be the set of terrain geometry points. Let $V$ be a 3D vertex or point in $D_G$, and $S_V$ be

the size in bytes of $V$. Let $D_I$ be the set of raster image samples. Let $P$ be a sample or pixel in $D_I$, and $S_P$ be the size in bytes of $P$. Let $B$ be the smallest rectangular subregion of $R$ to be transferred through the interactive terrain visualization system. Let $W_B$ and $H_B$ be the width and height of $B$, respectively. Let function $GD(B, D_G)$ define the set of $V$ in $D_G$ used to represent the terrain subregion $B$. Let function $ID(B, D_I)$ define the set of $P$ in $D_I$ used to represent the same terrain subregion $B$. Then the combined terrain geometry and image data used to represent region $B$ is $TD = GD(B, D_G) + ID(B, D_I)$, and the total size in bytes of the terrain data unit is $TDU = (|GD(B, D_G)| \cdot S_V) + (|ID(B, D_I)| \cdot S_P)$.

### 3.3.3 Hardware Component Characterization

At the core of our model is the hardware component characterization. In the traditional HMM, each memory components is characterized as a memory level or stage with an associated memory capacity and data transfer cost. We extend this approach to consider any hardware component capable of data storage transfer as a memory stage ($MS$). Similarly to the HMM we model each stage in terms of the memory capacity and data transfer, however, in order to support more flexibility in modeling the data transfer, we divide the data transfer cost into input transfer cost and output transfer cost.

Formally, we define the a memory stage as the set MS of performance related characteristics associated to a particular hardware component in the interactive visualization system. The primary characteristics taken into consideration in this work are:

- Memory capacity
- Read or output transfer rate
- Write or input transfer rate

However, we do not place restrictions on the type of characteristics that may included in the characterization. For example, OS related aspects that can affect the

effective performance of the hardware component could be taken into consideration by including additional characteristics as part of the memory stage definition. A more concrete example would be to incorporate information related to the filesystem for memory stages characteristics corresponding the disk storages units.

For performance estimation, we model a memory stage $MS_i$, in terms of the:

- Available memory capacity function $MC(MS_i)$ (TDU)
- Input transfer cost function $ITC(MS_i)$ (time per TDU)
- Output transfer cost function $OTC(MS_i)$ (time per TDU)

Figure 3–6 and 3–7 illustrate the memory stage in the component representation and data transfer representation, respectively

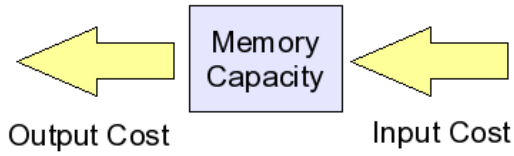



Figure 3–6: Memory Stage - Component Representation

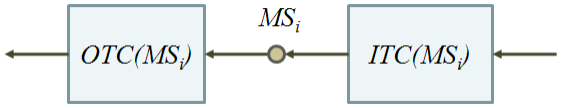



Figure 3–7: Memory Stage - Data Transfer Representation

### 3.3.4 Data Transfer Characterization

In this section we describe the modeling of the data transfer cost for the interactive visualization system based on the decomposition of such system into our hierarchical memory model. As mentioned earlier, transfer cost is defined in terms of time per terrain data unit transferred.

**Transfer Cost Between Stages**

The data transfer cost between two adjacent stages $MS_i$ and $MS_{i+1}$ is modeled as a function of the transfer cost associated with reading the data out from the higher stage ($MS_{i+1}$) and the transfer cost associated with writing the data into the lower stage ($MS_i$). Formally, the transfer cost function $TC(TransferCost_{in}, TransferCost_{out})$ where $TransferCost_{in} = OTC(MS_{i+1})$ and $TransferCost_{out} = ITC(MS_i)$.

That is, the transfer cost between memory stages is a function of the output cost of the higher stage and the input cost at the lower stage.

**Total System Transfer Cost**

The data transfer cost for the complete system is a function of the data transfer cost associate with transferring data between every adjacent stage in the system. If $S$ is the order set of memory stages,$|S| = N$ is the number of elements in $S$, and $Z_N = 0, 1, 2, 3, ..., N-1$ is the set of indices to the elements in $S$, then

$$TotalTransferCost = TTC(S) = \sum_{i=0}^{N-1} TC(MS_i, MS_{i+1})$$

### 3.3.5 Optimization Technique Characterization

We characterize the different optimization techniques described in Section 2.1 as modifiers applied to a particular memory stage. Stage modifiers are categorized as following:

- Data Unit Size Reduction
- Available Memory Capacity Increment
- Transfer Cost Reduction Between Stages

Depending on the optimization technique being consider, one or all the modifiers may be applied to the elements of the model.

### 3.3.6 System Characterization

In this section we describe the processes of characterizing the interactive terrain visualization system in terms of the hierarchical memory model. Looking at the system from the component perspective, we must subdivide the system into components with a set of performance related characteristics for each component. This requires a) to identify all the relevant hardware components of common rendering systems in a way that is meaningful for performance analysis, and b) identifying the most relevant hardware characteristics of each hardware components.

**Ideal System Characterization**

We define the Ideal System (see Figure3–8) as:

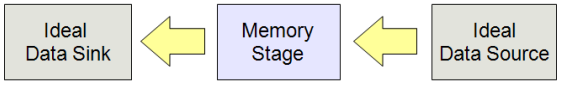



Figure 3–8: Ideal System

- A system with the minimum number of components.
- A system with an Ideal Data Source and an Ideal Data Sink.

The minimum number of components correspond to a data sink, a single stage, and a data source. In this ideal system, the user input determines which data is read from the data source, transferred trough the single memory stage and consumed by the data sink. We define the Ideal Data Source as $IdealDataSource = D_{src}$ if:

- $MC(D_{src}) = \infty$
- $ITC(D_{src}) = \infty$
- $OTC(D_{src}) = \infty$

and the Ideal Data Sink as $IdealDataSink = D_{sink}$ if

- $MC(D_{sink}) = 0$
- $ITC(D_{sink}) = 0$
- $OTC(D_{sink}) = 0$

That is, the ideal data sink has zero memory capacity and zero transfer cost, and the ideal data source has infinite memory capacity and transfer cost. In other words, the performance of this Ideal System is completely determined by the data processing characteristics of the single memory stage.

### 3.3.7 System Decomposition Methodology

In order to populate the model with all the relevant components, we start by representing the system as the previously described Ideal System. Then we identify the real system components that more closely resemble the ideal data sink and source. Then we follow a top-down decomposition approach starting at the highest level and recursively decomposing higher-level components into lower-level components until the all relevant components are incorporated into the model.

We begin the decomposition by identifying the endpoints of the systems, the data source and data sink. Identifying the data sink is easier in visualization systems, since the video card or graphic processing unit at the visualization workstation is the final consumer of terrain data. In addition, it is also the component with the lowest transfer cost. We select a remote server as end-point since it is the component with both the largest theoretical memory capacity (storage). The initial single stage in the high-level decomposition is the Visualization Workstation. This stage acts as an intermediary between the consumer (GPU) and the producer (Remote Server). We refer to these components as High-level System Components. These are the components that more closely approximate the Ideal System.

In more complex scenarios, additional considerations can aid in the identification of the High-level System Components. In particular, the decision to explicitly include a hardware component in the model characterization of the system can be made depending on:

- Natural hardware boundaries - Physical hardware boundaries generally map to increased transfer cost and memory isolation, both of which can be mapped into the stage model characteristics.
- Large changes in transfer cost compared to adjacent stages - Abrupt changes in data transfer cost between stages is significant enough for it to be considered as a stage.
- High computational capacity - The computational capacity of a particular hardware component makes it possible to implement more sophisticated data replacement algorithms, which can potentially improve the performance of the component, therefore it is convenient to consider the component as an independent element in the model.

### 3.3.8 High-level System Components

Once the high-level components are identified, we proceed to identify the most significant hardware characteristics and map them into model characteristics. In the following sections we describe each of the High-level System Components and the factors that affect their performance in terms of the performance characteristics of interest in the context of the model presented in this work.

**GPU**

Graphic processing units are hardware devices dedicated to the processing of graphics data in common computer systems. GPUs are connected to the computer motherboard via a Accelerated Graphics Port (AGP) or Peripheral Component Interconnect (PCI) bus. GPU components vary in features and capabilities. Many GPUs support programmability either for graphic processing or for general purpose processing. GPU programmability means that more sophisticated optimization techniques are available for this component. The fastest rendering is possible for data stored in video memory, local to the graphics processor. We can treat this memory as another level of cache for our geometry data and manage it much as we do the core memory cache. The following are the main performance characteristics of interest in this work and the factors that generally affect their value for the case of GPU components:

- Input cost: Affected by the memory bandwidth to the GPU.
- Memory capacity: Affected by the internal memory of the GPU.
- Output cost: Affected by the triangle fill rate of the GPU.

**Visualization Workstation**

The Visualization Workstation is basically a personal computer in which the user interaction with the visualization system is performed. For the purpose of this work, we consider consumer-level laptops and desktops computers as Visualization Workstations. The following are the main performance characteristics of interest

in this work and the factors that generally affect their value for the case of GPU components:

- Input cost: Affected by the disk write speed, and memory write speed and the network bandwidth.
- Memory capacity: Affected by the disk storage and the main memory.
- Output cost: Affected by the front-side bus, disk read speed, memory read speed, and network bandwidth.

**Remote Server**

A remote server is basically a computer system which is used for data serving. Actual remote server may consists of multiple CPUs or cores, multiple disk drives in RAID configuration. For the purpose of this work, we treat Remote Server as single CPU, single disk systems. Similarly to the case of the Visualization Workstation, the performance characteristics and the factors that affect them are:

- Input cost: Affected by the disk write speed, and memory write speed and the network bandwidth.
- Memory capacity: Affected by the disk storage and the main memory.
- Output cost: Affected by the front-side bus, disk read speed, memory read speed, and network bandwidth.

### 3.3.9 Further System Decomposition

After the High-level System components are identified, we proceed to recursively decompose each of them to identify memory stages of interest. The depth of the decomposition process will be determined by the precision required for the performance estimation. Generally, the subcomponents of each High-level System Components fall into one of the following categories:

**Random Access Memory Component**

Random Access Memory (RAM) components found in common computer are a type of volatile, writable memory storage based on solid state semiconductors

technology. The term "random access" means that any data element stored in the RAM memory component can be accessed at a constant time independently of previous data requests or of where the data is physically stored in the hardware of the component. RAM components are are commonly used as main or core memory in: laptops, desktops and server computers; cache memory in CPUs; data buffer in disk drives and network cards; and video memory in GPUs. RAM components typically have less memory capacity and transfer cost than other memory components in the system[1] . The performance characteristics and related factors for the RAM components are:

- Input cost: Affected by the memory write speed and front-side bus speed.
- Memory capacity: Affected by hardware component capacity and the operating system (OS) memory management.
- Output cost: Affected by the memory write speed and front-side bus speed.

**Flash Memory Components**

Flash memory components are non-volatile writable memory components. Like RAM component, they are also based on solid stage semiconductor technology. Flash memory is commonly used as removable storage devices connectable via USB interface; as removable memory cards in digital cameras, PDAs and cellular phones; and as secondary memory or memory storage in many types of embedded systems. Flash memory is generally slower than RAM memory components. In terms of memory capacity, current Flash memory support larger capacity than RAM memory, although the applications for large capacity Flash memory are few due to its higher price. In contrast to RAM memory components, data storage in Flash memory is usually

[1] The only exception are registers which are smaller and faster, but these are not considered in this work.

structured following a filesystem (FS) organization. The performance characteristics and related factors for the Flash memory components are:

- Input cost: Affected by Flash memory write speed, the connection BUS speed, and the OS and FS in use.
- Memory capacity: Affected by the hardware component capacity, and the OS and FS in use.
- Output cost: Affected by the Flash memory read speed, connection BUS speed, and the OS and FS in use.

**Disk Storage**

Disk storage components (also referred to as secondary or out-of-core memory in the literature) are non-volatile storage devices in which data is stored on the surface of magnetic disks platters. In contrast to RAM and Flash memory components, Disk storage components employ mechanical components which spin the disk platters and move the read-and-write heads. Disk storage component are commonly found in laptops, desktops and server computers (in internal and external, removable configurations) as well as in many embedded devices. Similarly to Flash memory components, Disk Storage generally use filesystem organization. The performance characteristics and related factors for the Disk storage components are:

- Input cost: Affected by the disk write speed, the connection BUS speed, and the OS and FS in use.
- Memory capacity: Affected by the component capacity, and the OS and FS in use.
- Output cost: Affected by the disk read speed, the connection BUS speed, and the OS and FS in use.

### 3.3.10 Model Evaluation

After decomposing the interactive terrain visualization system into the most relevant hardware components and identifing the most significant performance characteristics of each components, we can characterize each component as a memory

stage and evaluate the model in order to determine the expected performance in terms of the cost of transferring data from end-to-end, *i.e.*, from the Data Source to the Data Sink. To do so, we must:

- Select the data unit to be used in the performance estimation.
- Select how to model the performance of each memory stage $MS$ in terms of the selected performance characteristics by formally defining each of the performance modeling functions $MC(MS)$, $ITC(MS)$, and $OTC(MS)$ for each component.
- Select the transfer cost function $TC(TransferCost_{in}, TransferCost_{out})$ to model the data transfer cost between two adjacent memory stages $MS_i$ and $MS_{i+1}$, where $TransferCost_{in} = OTC(MS_{i+1})$ and $TransferCost_{out} = ITC(MS_i)$.

## 3.4 Model Validation

In the next chapter we validate our model via a test case study of interactive terrain visualization system.

# CHAPTER 4
# TEST CASE

In this chapter we describe the validation of our model via a test case study. In Section 4.1 we describe the system used for the test case study, namely, the WALSAIP Visual Terrain Explorer. In Section 4.2 we describe the validation methodology. Finally, in Section 4.3 we presents the model validation results.

## 4.1 Test Case System Description

### 4.1.1 WALSAIP Visual Terrain Explorer

The Wide Area Large Scale Automated Information Processing (WALSAIP) project [40] is concerned with the automated processing of signal-based information obtained from physical sensors. The main goal of the project is to support environmental monitoring applications by providing a framework capable of performing arbitrary computations on data received from sensors located on dispersed geographical locations. The nature of the computation being conducted is determined by the domain of the research being performed. The automated information processing framework is designed to operate in a time-space distributed manner, meaning that data storage and computation is not necessarily localized to a specific computing/storage server facility at a particular time. Data from sensors in multiple locations may be stored in geographically dispersed storage servers. Computation may be performed in a distributed manner in multiple computing nodes, which may operate in sequence or in parallel as required. Raw or processed data can be presented by different Information Rendering Systems (IRS), depending on the nature of either the data or the processing performed or being performed on it.

The WALSAIP Visual Terrain Explorer (VTE) is a terrain visualization tool which aims to provide an integrated visualization system for environmental monitoring applications. The main goal of the VTE application is to combine diverse georeferenced data in an interactive terrain visualization. In the context of the WALSAIP project , the VTE applications serves as an IRS which provide an user interface for the interactive, 3D visualization and exploration of georeferenced information belonging to a particular geographic area.

### 4.1.2 Implementation Technology

#### Java 2 Platform

The VTE application was developed using the Java 2 Standard Edition. Although Java is not necessarily a language traditional used for 3D visualization, it presents considerable advantages due to its cross-platform capabilities and due to the rich library of classes it provides, which facilitate building complex and versatile applications. In terms of the application architecture, the object-oriented nature of the language simplifies designing and building modular and extensible applications.

#### Eclipse Rich-Client Platform

The goal of improving the application capabilities not related to visualization respond to the need to provide a solid, sophisticated, and productive interaction experience for users working with the application. Previous development efforts had been mainly focused on developing the rendering infrastructure and had only provided a rudimentary graphical user interface (GUI) implemented using Java Swing Toolkit. Recent development efforts have focused on improving the GUI and to incorporate additional application functionality not related to the visualization, such as infrastructure for remote installation and update. To fulfill these goals, it was decided to port the VTE application to the Eclipse RCP [41] which provides an extensible framework for building Java application. The Eclipse RCP simplifies building sophisticated Java applications by providing a bare-bones skeleton which

already incorporates common functionality related to GUI management, local and remote file management, and application update operations. The actual application logic is programmed in a series of plug-in modules.

**OpenGL**

The VTE application uses OpenGL [42] as low-level graphic rendering library. OpenGL is a cross-platform, cross-language, general-purpose graphic library. Hardware accelerated execution of the OpenGL API is currently supported by most GPUs. Integration between Java and OpenGL is provided by the Java OpenGL binding library [43].

**Specialized Java Libraries**

The VTE implementation also uses the following specialized java libraries:

- Java NIO - Provides improved support for performance critical I/O operations in Java applications. Support working with ByteBuffer objects which provide a flexible operation for reading and writing to files, network sockets. Also provides support for memory mapping.
- Java Advanced Imaging - Provide a simple, high-level programming model for image processing operations.
- Java ImageIO - Provides a pluggable architecture for working with images stored in different formats.

### 4.1.3 Application Design

The application design follows the object-oriented and design pattern methodologies [44]. The initial design process was focused on providing a modular framework for constructing rendering pipelines. The pipeline was composed of three stages or layer:

- I/O Layer - Provide abstraction over different data sources and formats.
- Data Reduction Layer - Provides abstraction over different geometric simplification algorithms.

- Rendering Layer - Provides abstraction over different rendering techniques.

The pipeline was constructed at run-time depending on the inputs files by selecting the appropriate modules for each part of the pipeline. Variability in terms of performance is provided by using different Module objects which are responsible for implementing a particular stage of the pipeline.

The original design of the VTE application aimed to provide a flexible, modular architecture, but during the implementation process it became evident that the design was not flexible enough for our purposes. While the pipeline model seemed like a natural choice for the terrain rendering problem, the design of the classes used to represent the actual terrain data (the SurfaceData class) became more complicated. In particular, how to encapsulate and when to perform the program logic for handling different data formats presented several problems. The main problem was related to the fact the implementation detail of which internal representation was used by the SurfaceData class had different performance impact at different layer. Changing the internal implementation of the SurfaceData to improve performance in a layer, resulted in performance reduction in other layers. This is the result of the fact that different data processing algorithms required specific data layouts in order to perform adequately. Similarly, different rendering techniques required different data layout, which resulted in having to maintain different terrain representation inside different renderer modules, which added an undocumented data transformation operation to pipeline. In addition the rigid separation between layers was also too limiting for implementing complex data simplification algorithms which, in many cases, crossed the boundaries between layers. In particular, view-dependant LOD algorithms require data loading and simplification controlled by the view parameters. The original design allow communication between modules in the pipeline, but

data processing was performed only once, at the initialization phase of the visualization. Changing data reduction or filtering parameter required re-initializing the complete pipeline.

After a more in-depth study of LOD algorithms, rendering system architectures and a reevaluation of the desired application capabilities, the following the following design decisions were made in order to improve the design of the application:

- Move Multi-format support and data reduction and filtering support to an offline pre-processing phase.
- Simplify I/O to handle different data sources and only one common data format.
- Improve inter-module communication support via a MessageBus communication system.
- Simplify rendering layer to minimize the need for maintaining internal data representations.

### 4.1.4 Architecture

The VTE application was then built around the following general subsystems.

- The Module Subsystem
- The Module Factories
- The Message Bus
- The Graphical User Interface

In the following sections we describe each subsystem in more detail.

#### The Module Interface

The Module interface provides a common base type from which different specialized interfaces are extended. The main goal of the Module is to support information hiding and modularity by providing common interfaces to the various classes employed as part of the core logic of the application. This allows to minimize dependency on the implementation details of each class. The main sub-interfaces used in the application are the following:

- FileLoader - Data source access abstraction. The read() method is invoke to load a file from a particular data source into a DataSegment object.
- Renderer - Graphic rendering subsystem abstraction. Terrain data elements are sent for rendering by calling different draw() methods on a Renderer module. Internally, different Renderer modules may use different rendering techniques or hardware capabilities to render the data.
- Camera - Abstraction for different camera objects which provide different camera optical simulations and movement behaviors. The main method invoked by clients using this interface are the init() draw() and update() methods. The Camera object collaborates with the TerrainModel module (for LOD management) and with the Renderer module (for view parameters) in order to produce the final 3D rendering.
- TerrainModel - Abstraction which encapsulates different terrain data management implementations. It employs the FileLoader to read data the actual data, query the Camera module for view-dependent LOD managemnet, and sends data to be drawn to the Renderer module.

**The Module Factories**

Each different Module sub-interface has an associated ModuleFactory which is responsible for instantiating the desired Module implementation. All clients using the different Modules and the ModulesFactories are only aware of the public Interfaces mentioned above. All details related to which actual class implements any of the Interfaces is only known to the appropriate factory. Since the routines related to Module management and instantiation provided by all the Factories are exactly the same, each Factory delegates the actual work to a Java Generic-enabled ModuleFactory which each Factory create and use internally.

**Message Bus**

In order to provide a simple mechanism to provide communication paths between different parts of the application, a MessageBus system was developed. The

MessageBus is a Singleton Mediator class implemented as a Java static class. By registering as a MessageReceiver with the MessageBus, each object can receive messages from others objects of interest. All object interested in generating messages must only invoke MessageBus.sendMessage() with new Message object. The MessageBus takes care of passing the Message object to the Receivers.

**Graphical User Interface**

The Graphical User Interface (GUI) of the VTE application is based on a Model-View-Controller architecture [45]. In order to support different levels of sophistication at the GUI level, two different implementation of the application GUI where developed. The first implementation was based on the Java Swing Toolkit. In this implementation, a ViewFrame (a subclass of a Swing JFrame) contains a ViewPanel (a subclass of a JOGL GLJPanel). Each ViewPanel contains a TerrainModel object which encapsulates the terrain data management and rendering modules. The second implementation is based on the Eclipse Rich-Client Platform. In this implantation the TerrainModel object is bundled as an RCP plugin. The two GUI version share the same core classes and messaging infrastructure.

### 4.1.5 Java Performance Study

To successfully use Java technology for performance critical applications as in the case of interactive terrain visualization, it is crucial to first understand the general aspects related to the performance of Java applications. To support platform-independent execution, Java application source code is compiled to byte-code, which is binary format design to execute in a virtual architecture call the Java Virtual Machine (JVM) [46]. The JVM itself is implemented natively in different, real computer architectures. Once a JVM is ported to a particular architecture, any Java program can run on that architecture. The JVM performs many services that were traditionally performed by the Operating System such as thread management and memory

management. JVM implement automatic memory management via Garbage Collection (GC) [46], which periodically reclaims unused memory.

Performance of Java applications running on early JVMs was severely limited due to the interpreted execution of the byte-code. Java performance was substantially improved with the development of JIT compilers [47] in which the byte-code of Java applications is compiled into native machine code "on the fly" during application startup, and then executed. However, JIT compilation introduced other performance problems related to the application startup time, during which the JIT compilation was performed [48]. More recent JVMs introduced the Java HotSpot performance engine [48]. Instead of using the JIT approach, the HotSpot engine runs the Java program immediately using an interpreter, analyzes the run-time execution of the program, detects critical performance "hot spots", and performs optimized native-code compilation on the hot-spots. This approach avoids wasting time compiling infrequently executed code, allows focusing on the critical parts of the program, minimizes compilation time, and allows for continued dynamic optimization of the program based on the actual run-time metrics.

In order to identify how identify potential performance problems caused by the use of Java technology, we performed a performance study of the VTE application. For this performance study we used the Eclipse Test and Performance Tools Platform Project (TPTP) [49]. Eclipse TPTP is an open source, collaborative integrated testing, tracing, profiling and monitoring platform which encompasses everything from data collectors to a data model and viewers, all integrated within the Eclipse Rich Client Platform. TPTP provides a ready-to-use, but extensible, solution for data collection and testing development built on a common framework which provides the opportunity for different products to be used seamlessly alongside each others and allowing them to make their data collector available to other tools as well.

Table 4–1: Performance Metric provided by Eclipse TPTP.

| Memory Usage | |
|---|---|
| Total Instances | Number of instances that had been created. |
| Live Instances | Number of instances (not garbage collected). |
| Collected Instances | Number of instances that were garbage collected. |
| Total Size | Size in bytes of all created instances. |
| Active Size | Total size of all live instances combined. |
| Execution Time | |
| Base time | Time taken to execute the invocation, excluding the time spent in other methods that were called during the invocation. |
| Cumulative Time | Time taken to execute all methods called from an invocation. |
| Calls | Number of calls made by a selected method. |

**Performance Metrics**

Table 4–1 describes the performance metrics measured during the performance study of the VTE application using the Eclipse TPTP.

**Results**

The results of the performance profiling of the VTE applications are presented in the Appendix A. Tables A–1 presents the results for the total memory usage profiling; A–2 presents the results for memory usage per object instance; A–3 presents the results for the base execution time profiling; A–4 presents the results for the cumulative execution time profiling; and A–5 present the results of the execution time per method call profiling.

**Analysis**

The results obtained give us a clear indicator of which are the most critical aspects of the application in terms of performance. The most critical aspects were the data loading and processing and the level-of-detail implementation. In particular, we observe performance problems associated with: a) the use of Java NIO classes; b) object creation and method invocation inside critical loops; and c) the size of terrain data structures maintained in memory. The Java NIO classes play a crucial role during the data loading and conversion process. Java NIO provides classes

and methods for high-performance IO operation in Java. However, as observed in the results for the number of calls metric, it is possible to use the NIO classes in a sub-optimal way. In this case, the application invokes buffer.put(floatValue) method inside a loop. This approach results in unnecessary number of method invocations. The second observation is that, classes and interface that will be used in performance critical methods must be carefully designed in order to minimized any potentially unnecessary overhead. Such is the case of the use of the getters/setters method for minElevation and maxElevation in the BoundingBox class. During data loading phase, each new sample read is compared to determine the minimum and maximum elevation values. However, do to a naive implementation of the BoundingBox class, each comparison incurred in over six getter/setter operations. The third important observation extracted from the results is that, during run-time, the most critical part of the application is the LOD implementation. The LOD implementation builds various quadtrees where each leaf is a block of the original data, which means that even though the algorithm may be able to allow us better interactive rendering, its implementation may require more memory to contain the quadtrees and images.

**Performance Optimizations**

In order to improve the performance of the application during loading time, the following stratagems were adopted:

- Use arrays instead of single values when using NIO Buffers in order to minimize method invocations.
- Move terrain data processing operations to a offline preprocessing stage.
- Minimize method invocation and object creation inside critical loops.

## 4.2 Model Validation Methodology

We employ the following methodology to validate our model via test case study.

- Construct model of test case system.

  - Map components into model elements.
- Evaluate model for each test platform.
  - Measure actual component performance.
  - Calculate estimated performance based on the model.
- Benchmark each test platform.
  - Measure actual performance.
- Validate benchmark results against model results.

### 4.2.1 Test Case Model Evaluation Scope

Here we describe the scope of the test case in terms of components considered in the study, data unit characterization, as well as datasets, and test systems used to conduct the study.

**System Decomposition**

For the purpose of this test case study, we only consider the main components obtain from the decomposition of the visualization workstation component in the complete interactive visualization system. Figure 4–1 illustrates the system decomposition process and the final set of components selected for evaluation. The components are:

- GPU memory
- Main memory
- Local disk storage

**Data unit characterization**

For the purpose of this test case study, we selected as data unit as combination of a geometry mesh constructed from a 64x64 regular height field and a raster image of 64x64 pixels. The geometry mesh is constructed as a triangle strip where each height samples is represented by two triangles. Each vertex is composed of 4 floats (x,y,z,w), and each pixel is composed of 4 bytes (r,g,b,a), for a total of 147456 bytes per data unit.

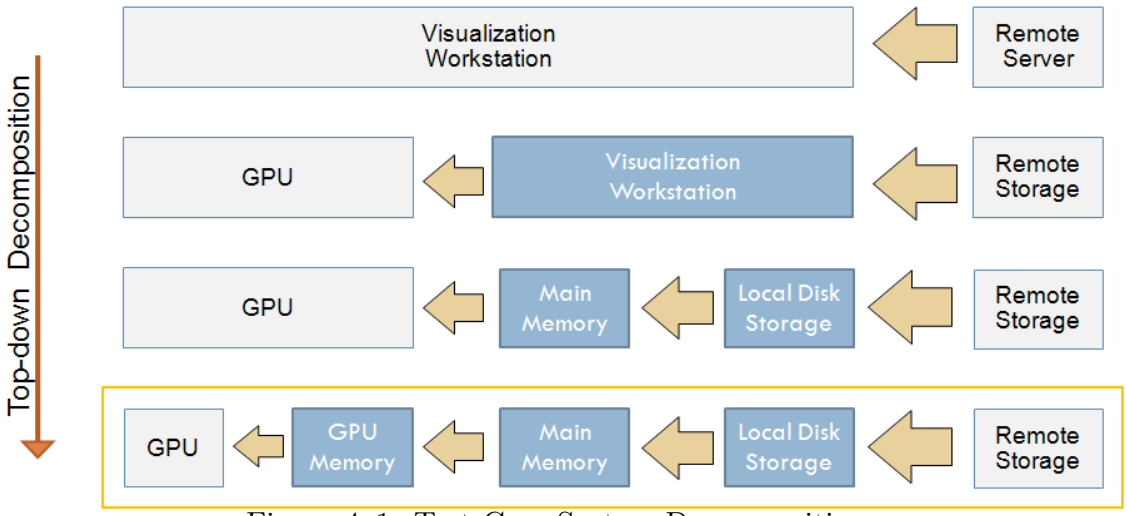



Figure 4–1: Test Case System Decomposition

In this work, we restrict both function $GD(B, D_G)$ and function $ID(B, D_I)$ so that:

- $GD(B, D_G)$ defines a rectangular array of terrain geometry points with width $W_{GD}$ and height $H_{GD}$.
- $ID(B, D_I)$ defines a rectangular array of terrain image pixels with width $W_{ID}$ and height $H_{ID}$.
- $W_{GD} = W_{ID} = W_B$
- $H_{GD} = H_{ID} = H_B$

In other words there is a one-to-one correspondence between geometry points, pixel and spatial points in the terrain region being transferred. We use $W_B = H_B = 64$, so every data unit consists of 64x64 points and 64x64 pixels. We define $V = (x, y, z, w)$, where each value is stored as single-precession binary floating point number, therefore $S_V = |V| \times$ 4 bytes = 16 bytes. We define $P = (r, g, b, a)$, where each value is a byte, therefore $S_P = |P| \times$ 1 byte = 4 bytes.

Then $TDU = (|GD(B, D_G)| \times S_V) + (|ID(B, D_I)| \times S_P) = ((W_{GD} \times H_{GD}) \times S_V) + ((W_{ID} \times H_{ID}) \times S_V) = ((64 \times 64) \times 16) + ((64 \times 64) \times 4) = 81920$ bytes.

**Memory Stage Modeling**

For the purpose of the validation of our model, we choose to model the memory capacity, input transfer cost and output transfer cost of each Memory Stage using constant functions. For each component, we conducted a performance measurement to identify each of the characteristics of interest for each system under the same test conditions, that is, with the same conditions in which we start every performance measurement for the formal performance study of the VTE application. The actual values used for the modeling of each component is presented in Table 4–4.

**Between-Stages Transfer Cost Modeling**

To model the transfer cost between stages $MS_i$ and $MS_{i+1}$ where:

- $TransferCost_{in} = OTC(MS_{i+1})$
- $TransferCost_{out} = ITC(MS_i)$

we use the transfer cost function:

- $TC(TransferCost_{in}, TransferCost_{out}) = max(TransferCost_{in}, TransferCost_{out})$.

**Datasets**

For this study, we used four different datasets of 4 different geographic areas:

- Island of Hawaii - The terrain dataset consists of an island geography dominated by the Mauna Loa and Mauna Kea volcanoes. Figure B–1 shows a screenshot of the visualization of this dataset.
- Grand Canyon National Park - This dataset is a generally flat rectangle, with high surface detail around the gorge formed by the Colorado River. Figure B–2 shows a screenshot of the visualization of this dataset.
- Municipality of Guanica, Puerto Rico - This dataset exhibits more surface variability, since the geographic region features coastlines, mountains, valleys and rivers. Figure B–3 shows a screenshot of the visualization of this dataset.

- Jobos Bay Reserve, Puerto Rico - While generally similar to the Guanica datasets, this dataset has more pronounced mountain areas and less elevation in the shore region. Figure B–4 shows a screenshot of the visualization of this dataset.

Table 4–2 describe the size of each dataset in terms of number of pixels/samples, bytes, and data units.

Table 4–2: Dataset characteristics

| Area | Dimensions (pixels) | Size (bytes) | | Size (Data Units) |
|---|---|---|---|---|
| | | Terrain | Image | |
| Hawaii-1 | 1024 × 1024 | 16,777,216 | 4,194,304 | 256 |
| Hawaii-2 | 2048 × 2048 | 67,108,864 | 16,777,216 | 1024 |
| Hawaii-3 | 4096 × 4096 | 268,435,456 | 67,108,864 | 4096 |
| Hawaii-4 | 8192 × 8192 | 1,073,741,824 | 268,435,456 | 16384 |
| GrandCanyon-1 | 512 × 256 | 2,097,152 | 524,288 | 32 |
| GrandCanyon-2 | 1024 × 512 | 8,388,608 | 2,097,152 | 128 |
| GrandCanyon-3 | 2048 × 1024 | 33,554,432 | 8,388,608 | 512 |
| GrandCanyon-4 | 4096 × 2048 | 134,217,728 | 33,554,432 | 2048 |
| Guanica-1 | 1024 × 1216 | 19,922,944 | 4,980,736 | 304 |
| Guanica-2 | 2048 × 2432 | 79,691,776 | 19,922,944 | 1216 |
| Guanica-3 | 4096 × 4864 | 318,767,104 | 79,691,776 | 4864 |
| Guanica-4 | 8192 × 9728 | 1,275,068,416 | 318,767,104 | 19456 |
| JobosBay-1 | 1024 × 690 | 11,304,960 | 2,826,240 | 172.5 |
| JobosBay-2 | 2048 × 1380 | 45,219,840 | 11,304,960 | 690 |
| JobosBay-3 | 4096 × 2760 | 180,879,360 | 45,219,840 | 2760 |
| JobosBay-4 | 8192 × 5520 | 723,517,440 | 180,879,360 | 11040 |

### Test Systems

All test were performed on three different system configurations. Table 4–3 presents the vendor specifications for the components selected for the test case study. In order to analyze the performance of each system based on the hardware performance parameters that approximate more closely the actual performance available during the execution of interactive visualization system, we conducted actual performance measurements for each component. Table 4–4 presents the performance values for each hardware components. The memory capacity values are in data units. The transfer cost units are in seconds per data unit transferred.

Table 4–3: Test Systems Specifications

| Component | Parameter | System 1 | System 2 | System 3 |
|---|---|---|---|---|
| GPU | Memory Capacity | 256 MB | 256 MB | 32 MB |
| | Memory Bandwidth | 32 GB/s | 19.4 GB/s | 6.4 GB/s |
| Main Memory | Memory Capacity | 2 GB | 2 GB | 1 GB |
| | Memory Bandwidth | 5336 MB/s | 4264 MB/s | 2133 MB/s |
| Disk Storage | Storage Capacity | 80 GB | 80 GB | 80 GB |
| | Transfer Rate | 300 MB/s | 59.4 MB/s | 43.8 MB/s |

Table 4–4: Test Systems Actual Parameters

| Component | Parameter | System 1 | System 2 | System 3 |
|---|---|---|---|---|
| GPU | Memory Capacity | 227.56 | 227.56 | 28.44 |
| | Output Cost | 0.1088 | 0.1805 | 0.3599 |
| | Input Cost | 0.1223 | 0.2028 | 0.4930 |
| Main Memory | Memory Capacity | 1338.03 | 1274.31 | 0.5941 |
| | Output Cost | 0.2482 | 0.4293 | 0.5941 |
| | Input Cost | 0.4431 | 0.8586 | 1.1209 |
| Disk Storage | Memory Capacity | 67811.56 | 67838.86 | 67838.86 |
| | Output Cost | 25.3950 | 37.0066 | 53.8278 |
| | Input Cost | 36.2786 | 45.6871 | 78.0112 |

### 4.2.2 Performance Measurement Tests Description

In order to measure the performance of the test case system, VTE, running on each of the test platform, the VTE application was enhanced with instrumentation to obtain the pertinent performance metrics. These are:

- Average frames per second.
- Average triangles per second.
- Average data units per second.
- Total Transfer Cost.

Due to the nature of the HotSpot compiler that is part of the modern JVM, special attention had to be taken in the implementation of the instrumentation code in the VTE. When the JVM loads a Java class, the initial execution performed via the JVM built-in interpreter. During these initial phase, the HotSpot compiler performs analyzes the performance of the code and performs optimizations on-the-fly before the Java code runs at optimal speed. Performance measurements obtained

during this initial phase will be inaccurate and not indicative of the true performance of the the application. Therefore, before beginning rendering performance measurements, the instrumentation code waits for a fixed time period to allow the HotSpot compiler to stabilize. In addition, to avoid measuring too frequently (which would also hamper rendering performance), the instrumentation code only compute the metrics at fixed time intervals. The default sampling duration is 10 seconds. After waiting for the initial HotSpot optimization phase, the instrumentation code performs a calibration routine, where it computes the number of frames rendered during the sampling period. After doing this calibration, the instrumentation code reports the measurements after these many frames are rendered.

In order to obtain a performance measurement representative of real usage patterns while minimizing variability, the test performed consists of a pre-recorded set of viewing operations. These operations form a virtual flight path in which the complete terrain surface is explored by moving the virtual camera over the complete terrain at a predefined set of distances and angles. The same predefined flight path is performed 5 times for each dataset.

### 4.3 Model Validation Results

The complete listing of the test case results are presented in Appendix C. All results are presented in terms of data units. The results of the model performance estimates for each of the data sets are presented in Table 4–5.

The results of the performance measurement for the Hawaii dataset are presented in Tables C–1, C–2, and C–3. Figures D–1, D–2 and D–3 present a comparison between the model results and the performance measurements.

The results of the performance measurements for the Grand Canyon dataset are presented in Tables C–4, C–5, and C–6. Figures D–4, D–5 and D–6 present a comparison between the model results and the performance measurements.

Table 4–5: Model Results - Transfer Cost

| **Dataset** | **Hawaii 1** | **Hawaii 2** | **Hawaii 3** | **Hawaii 4** |
|---|---|---|---|---|
| **# Data Units** | **256** | **1024** | **4096** | **16384** |
| System 1 | 0.8206 | 3.282328016 | 13.12931206 | 52.51724826 |
| System 2 | 1.1979 | 4.791791353 | 19.16716541 | 76.66866166 |
| System 3 | 1.7415 | 6.965990971 | 27.86396388 | 111.4558555 |
| **Dataset** | **Grand Canyon 1** | **Grand Canyon 2** | **Grand Canyon 3** | **Grand Canyon 4** |
| **# Data Units** | **32** | **128** | **512** | **2048** |
| System 1 | 0.102572751 | 0.410291002 | 1.641164008 | 6.5647 |
| System 2 | 0.14974348 | 0.598973919 | 2.395895677 | 9.5836 |
| System 3 | 0.217687218 | 0.870748871 | 3.482995485 | 13.9320 |
| **Dataset** | **Guanica 1** | **Guanica 2** | **Guanica 3** | **Guanica 4** |
| **# Data Units** | **304** | **1216** | **4864** | **19456** |
| System 1 | 0.97444113 | 3.8978 | 15.59105808 | 62.36423231 |
| System 2 | 1.422563058 | 5.6903 | 22.76100893 | 91.04403572 |
| System 3 | 2.068028569 | 8.2721 | 33.08845711 | 132.3538284 |
| **Dataset** | **Jobos Bay 1** | **Jobos Bay 2** | **Jobos Bay 3** | **Jobos Bay 4** |
| **# Data Units** | **173** | **690** | **2760** | **11040** |
| System 1 | 0.552931233 | 2.211724933 | 8.846899731 | 35.38759892 |
| System 2 | 0.807210946 | 3.228843783 | 12.91537513 | 51.66150053 |
| System 3 | 1.173470159 | 4.693880635 | 18.77552254 | 75.10209015 |

The results of the performance measurements for the Guanica dataset are presented in Tables C–7,C–8, and C–9. Figures D–7, D–8 and D–9 present a comparison between the model results and the performance measurements.

The results of the performance measurements for the Jobos Bay dataset are presented in Tables C–10,C–11, and C–12. Figures D–10, D–11 and D–12 present a comparison between the model results and the performance measurements.

### 4.3.1 Discussion

Observing the results obtained, we notice that in general, both the model results and the test results follow a similar pattern of exponential increase as the dataset

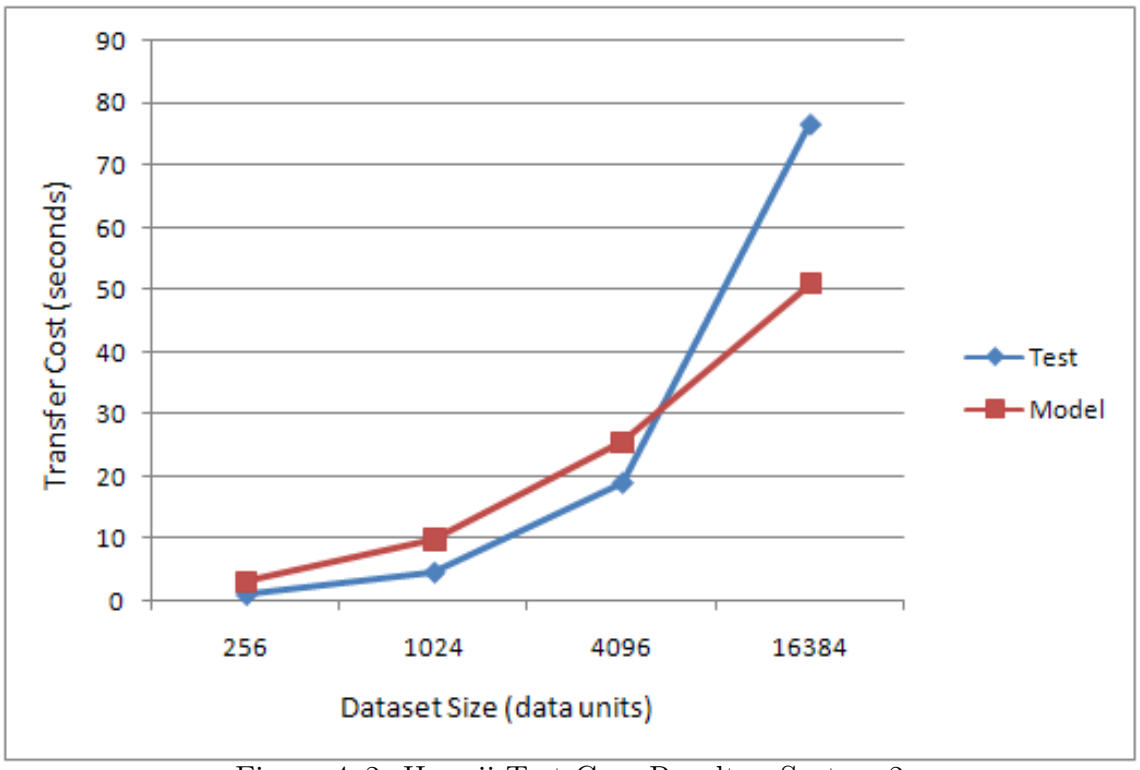



Figure 4–2: Hawaii Test Case Results - System 2

size increases. The only exceptions are the results obtained for the Hawaii Dataset for System 2 and 3 (see Figures 4–2 and D–3) were the model estimated larger transfer cost than what was actually measured . However, we also observed that the curves for the model and the test results diverge as the dataset size increases.

With the exception of the last test (which uses the largest dataset), the model results for the Hawaii set on System 1 (see Figure 4–4) were the ones that most

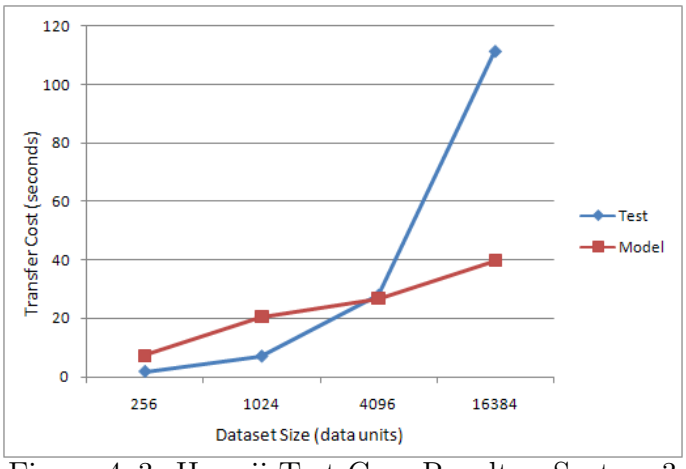



Figure 4–3: Hawaii Test Case Results - System 3

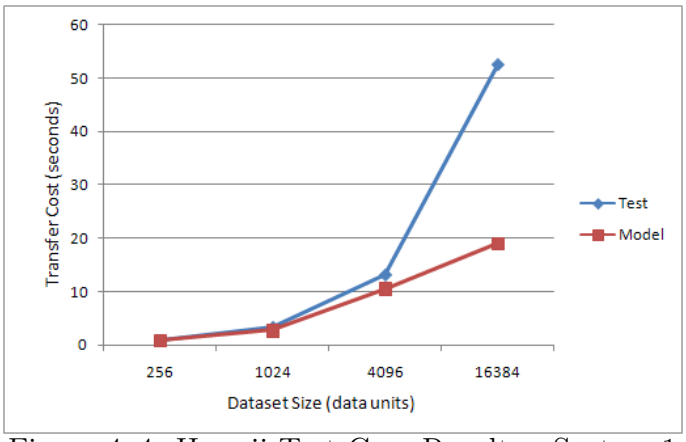



Figure 4–4: Hawaii Test Case Results - System 1

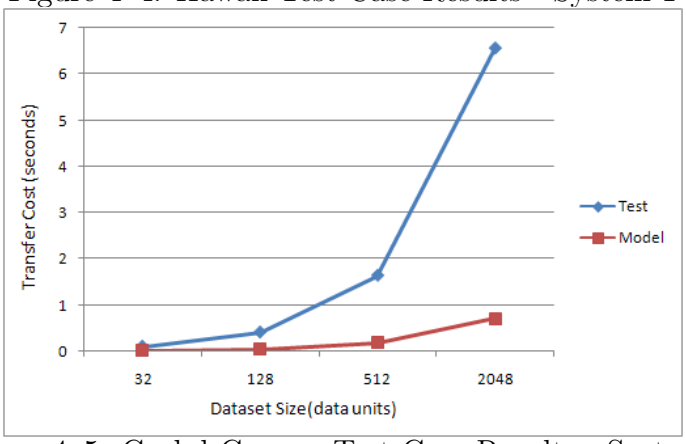



Figure 4–5: Grabd Canyon Test Case Results - System 2

closely approximated the actual performance. On the other hand, the results for the last test of the Grand Canyon set on System 1 (which consists of the smallest datasets) produced the results with the smallest difference between the estimated and the actual performance (see Figure 4–5). All other results showed the same general behavior of divergence between the estimated and the actual performance results.

# CHAPTER 5
# CONCLUSIONS AND FUTURE WORK

## 5.1 Conclusions

In this work we have presented:

- A hierarchical memory model for the performance estimation of interactive terrain visualization systems.
- A methodology for mapping hardware components in an interactive terrain visualization system into memory stages in the hierarchical memory model.
- A Java-based cross-platform interactive terrain visualization tool called Visual Terrain Explorer.

Our model can be successfully used to estimate the performance of the system for certain dataset sizes. When the dataset size increases beyond certain point, the performance estimations obtain using the model diverge considerably from the performance measurements obtained for our test case.

The divergence observed in the results can be attributed to several factors:

- Choice of modeling function:
  - The use of constant functions to model the performance characteristics of the memory stages.
  - The maximum transfer cost as modeling function for the transfer cost between adjacent stages.
- Software-related characteristics that affect hardware component performance:
  - OS overhead related to memory management and filesystem management.

– Overhead caused by the choice of Java as implementation technology, and the effect of the automatic memory management by the Java Virtual Machine.

The model presented in this research serves as a foundation for characterizing the the hardware components in interactive visualization systems. By incorporating terrain data characteristic into the hierarchical memory model we were able to construct an improved picture of visualization system in which the most critical performance aspects can be identified. Understanding which components of the visualization system are the most critical performance bottlenecks can aid system designer in the selection of both hardware components and software techniques for terrain data management.

## 5.2 Closing Remarks

After more than 30 years of research and enormous advances in computer hardware technology, the management of terrain data is still the biggest challenge faced by the designer of interactive terrain visualization system. As technology continue to advance, larger and larger datasets become available due to improved data acquisition systems and larger data storage technology. Current technology trends point toward a future of ubiquitous computing; a great variety of computing platforms such as desktop, laptops, tablets, smart-phones, PDAs, and game consoles, interconnected via the Internet. This future scenario of highly heterogeneous computing systems may limit the effectiveness of previous PC-oriented techniques for terrain data management. Future visualizations systems, for both research and commercial purposes, should be capable of providing a consistent and productive user experience. Understanding the performance capabilities required by a visualization system will continue to be critical for the design and implementation of interactive terrain visualization systems.

## 5.3 Future Work

We propose the following areas as potential future extensions to the work presented here:

### 5.3.1 Model Enhancements

The model presented in this research, can be extended to take into consideration system configurations with parallel components. In particular:

- Parallel GPUs.
- Parallel Disk[1] .
- Parallel Remote Servers.

### 5.3.2 System Studies

In terms of system configurations used as part of the performance study, several future research direction are possible.

**System Implementations**

In order to further validate the model, a performance study should be performed focusing on interactive terrain visualization systems implemented in other programming languages or technologies other than Java.

**Additional System Components**

Extend the depth of the performance study by taking into account more components in the visualization system, in particular:

- CPU cache.
- Remote server data layout and indexing.

**Different High-level System Configurations**

Performing studies using visualization systems view different composition than the configurations considered in this work. For example:

[1] Parallel disks are considered in the HMM-related literature.

- Mobile and embedded devices.
- Game consoles.
- Data streaming with multiple cache nodes [2] .

### 5.3.3 Problem Domain Studies

Finally, the model presented can be extended or adapted to incorporate data characteristics and access patterns for other problem domains, such as:

- Visualization of arbitrary 3D geometry.
- General multimedia systems.

[2] Incorporating the work previously presented in [36].

# APPENDICES

# APPENDIX A
# VTE JAVA PERFORMANCE PROFILING RESULTS

Table A–1: Memory Usage - Total Size.

| Class | Total Size |
|---|---|
| java.lang.Class | 33408 |
| vte.model.geomipmap.TerrainBlock | 1152 |
| java.lang.Thread | 672 |
| java.util.TimerThread | 416 |
| vte.datatypes.Vector | 240 |
| vte.model.geomipmap.Patch | 200 |
| vte.model.geomipmap.QuadTreeNode | 176 |
| vte.datatypes.Position | 160 |
| vte.model.geomipmap.TerrainBlock | 128 |
| vte.messagebus.Message | 120 |
| vte.datatypes.Light | 96 |
| vte.datatypes.Vertex | 96 |
| vte.module.data.SurfaceData | 96 |
| vte.datatypes.Angle | 96 |
| vte.model.LODTerrainModel | 80 |
| vte.model.geomipmap.GeoMMLandscape | 72 |
| vte.datatypes.BoundingBox | 72 |
| vte.ModuleFactory | 48 |

Table A–2: Memory Usage - Total Instances.

| Class | Total Instances |
|---|---|
| java.lang.Class | 348 |
| vte.model.geomipmap.TerrainBlock | 16 |
| vte.datatypes.Vector | 15 |
| java.lang.Thread | 7 |
| vte.datatypes.Angle | 6 |
| vte.datatypes.Vertex | 6 |
| vte.model.geomipmap.QuadTreeNode | 6 |
| vte.datatypes.Position | 5 |
| vte.messagebus.Message | 5 |
| vte.model.geomipmap.Patch | 5 |
| java.util.TimerThread | 4 |
| vte.model.geomipmap.TerrainBlock | 4 |
| vte.data.DataType | 3 |
| vte.datatypes.BoundingBox | 3 |
| vte.data.source.loader.file.FileLoaderFactory | 2 |
| vte.data.SurfaceData | 2 |
| vte.ModuleFactory | 2 |
| vte.datatypes.Light | 2 |
| vte.camera.Fully3DCamera | 1 |

Table A–3: Execution Time - Base.

| Class | Method | Time |
|---|---|---|
| java.nio.DirectFloatBufferU | put(int, float) | 17.12 |
| vte.model.geomipmap.TerrainBlock | setVertexAndShading(int, int, int) | 6.95 |
| sun.misc.Unsafe | putFloat(long, float) | 4.98 |
| java.nio.DirectFloatBufferU | ix(int) | 3.71 |
| java.nio.Buffer | checkIndex(int) | 3.7 |
| opengl.impl.mipmap.Mipmap | gluBuild2DMipmaps(GL) | 2.07 |
| vte.model.geomipmap.HeightMap2Df | get(int, int) | 1.94 |
| vte.model.geomipmap.TerrainBlock | biLinearInterp(int[]) | 1.55 |
| vte.model.geomipmap.TerrainBlock | calcDn2(float) | 1.24 |
| imageio.plugins.png.PNGImageReader | read(int,ImageReadParam) | 0.88 |
| vte.model.geomipmap.TerrainBlock | renderBlock(GL) | 0.77 |
| java.util.Random | next(int) | 0.72 |
| vte.model.geomipmap.GeoMMLandscape | access(GeoMMLandscape) | 0.65 |
| java.lang.Math | max(float, float) | 0.29 |
| vte.model.LODTerrainModel | setupTextures(GL) | 0.25 |
| java.lang.Math | abs(float) | 0.25 |
| vte.data.SurfaceData | putSample(int, int, float) | 0.21 |
| vte.model.geomipmap.TerrainBlock | findBlockCenter() | 0.17 |

Table A–4: Execution Time - Cumulative.

| Class | Method | Time |
|---|---|---|
| vte.gui.ViewPanel | display(GLAutoDrawable) | 38.78 |
| vte.model.LODTerrainModel | draw(GL) | 38.78 |
| vte.model.geomipmap.GeoMMLandscape | render(GL) | 38.77 |
| vte.model.geomipmap.Patch | render(GL) | 38.77 |
| vte.model.geomipmap.TerrainBlock | render(GL) | 38.77 |
| vte.model.geomipmap.TerrainBlock | renderBlock(GL) | 38.77 |
| vte.model.geomipmap.TerrainBlock | setVertexAndShading(int, int, int) | 37.78 |
| java.nio.DirectFloatBufferU | put(int, float) | 29.51 |
| vte.gui.ViewPanel | init(GLAutoDrawable) | 11.84 |
| vte.model.LODTerrainModel | init(GL) | 11.78 |
| sun.misc.Unsafe | putFloat(long, float) | 4.98 |
| vte.model.geomipmap.GeoMMLandscape | GeoMMLandscape(HeightMap) | 4.82 |
| vte.model.geomipmap.GeoMMLandscape | initPatches(int, int, int, float) | 4.82 |
| vte.model.geomipmap.TerrainBlock | TerrainBlock(GeoMMLandscape) | 4.82 |
| vte.gui.MultiFrameGUI | actionPerformed(ActionEvent) | 4.67 |
| vte.gui.MultiFrameGUI | access(MultiFrameGUI, ActionEvent) | 4.67 |
| vte.gui.MultiFrameGUI | openActionPerformed(ActionEvent) | 4.67 |
| vte.model.geomipmap.TerrainBlock | calcDn2(float) | 4.51 |

Table A–5: Execution Time - Calls

| Class | Method | Calls |
|---|---|---|
| java.nio.DirectFloatBufferU | put(int, float) | 5591040 |
| java.nio.Buffer | checkIndex(int) | 5591040 |
| java.nio.DirectFloatBufferU | ix(int) | 5591040 |
| sun.misc.Unsafe | putFloat(long, float) | 5591040 |
| vte.model.geomipmap.HeightMap2Df | get(int, int) | 2854256 |
| vte.model.geomipmap.TerrainBlock | setVertexAndShading(int, int, int) | 931840 |
| vte.model.geomipmap.GeoMMLandscape | access(GeoMMLandscape) | 931840 |
| java.lang.Math | max(float, float) | 436912 |
| java.lang.Math | abs(float) | 371377 |
| vte.model.geomipmap.TerrainBlock | biLinearInterp(int[]) | 371376 |
| sun.awt.image.ByteInterleavedRaster | getSample(int, int, int) | 196608 |
| java.util.Random | next(int) | 131072 |
| vte.datatypes.BoundingBox | getMaxElevation() | 66050 |
| vte.datatypes.BoundingBox | getMinElevation() | 66050 |
| java.awt.image.BandedSampleModel | setSample(int, int, int, DataBuffer) | 66049 |
| sun.awt.image.SunWritableRaster | setSample(int, int, int, float) | 66049 |
| vte.data.SurfaceData | putSample(int, int, float) | 66049 |
| java.awt.image.DataBuffer | getElemFloat(int, int) | 66049 |

# APPENDIX B
# TERRAIN DATASETS

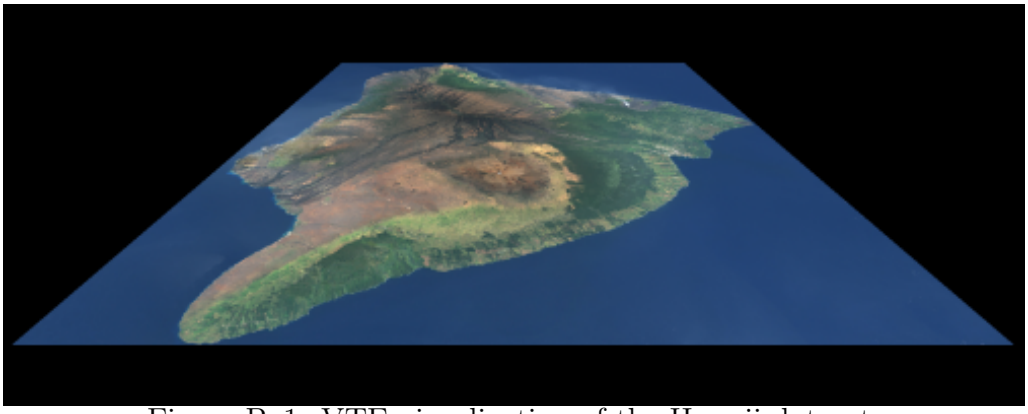


Figure B–1: VTE visualization of the Hawaii dataset.

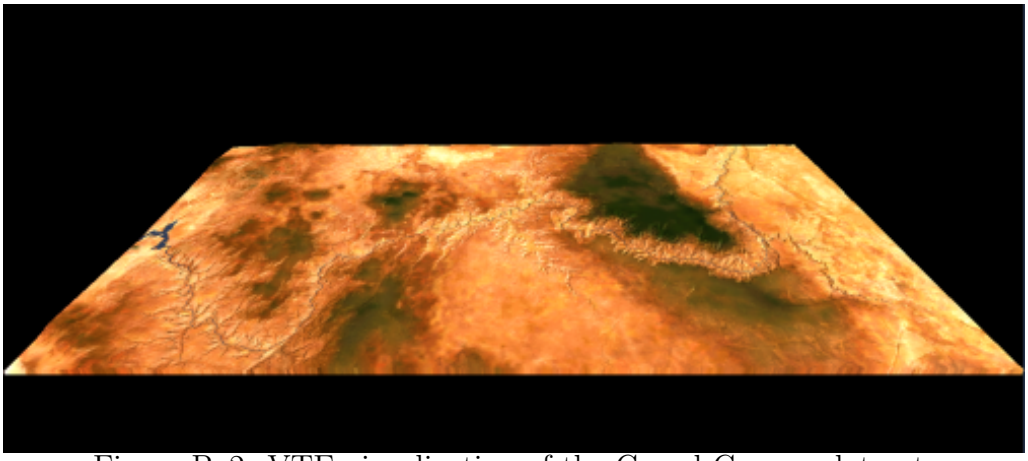


Figure B–2: VTE visualization of the Grand Canyon dataset.

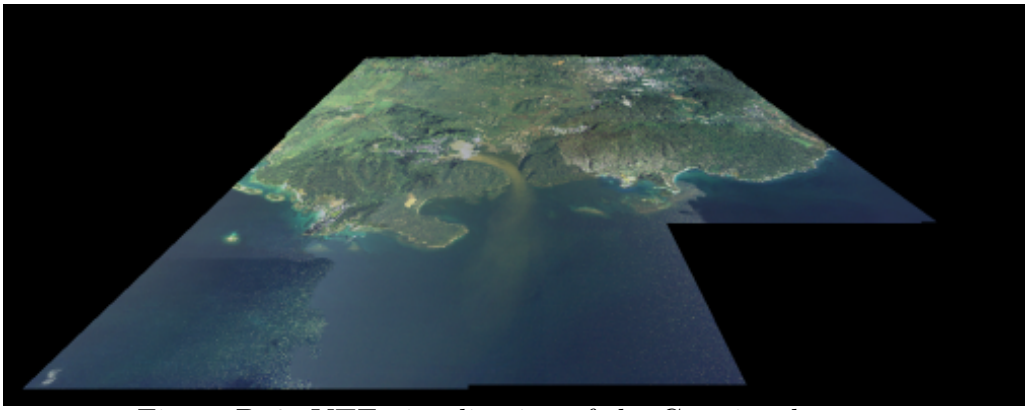


Figure B–3: VTE visualization of the Guanica dataset.

Figure B–4: VTE visualization of the Jobos Bay dataset.

# APPENDIX C
# TEST CASE PERFORMANCE MEASUREMENTS RESULTS

Table C–1: Hawaii - System 1

| | Average Frame Rate | Average Triangle Rate | Average Data Units Rate | Average Transfer Cost |
|---|---|---|---|---|
| Hawaii-1 | 108.4390 | 1,614,530.3000 | 394.1724 | 0.6495 |
| | 112.2770 | 1,048,619.6230 | 256.0107 | 1.0000 |
| | 112.7430 | 1,134,087.9670 | 276.8769 | 0.9246 |
| | 113.5780 | 1,199,308.3130 | 292.7999 | 0.8743 |
| | 113.4900 | 1,252,764.7260 | 305.8508 | 0.8370 |
| Average | 112.1054 | 1,249,862.1858 | 305.1421 | 0.8390 |
| Hawaii-2 | 108.8030 | 1,924,497.9320 | 469.8481 | 2.1794 |
| | 104.0620 | 1,541,919.7230 | 376.4452 | 2.7202 |
| | 102.7470 | 1,507,382.2810 | 368.0133 | 2.7825 |
| | 102.3450 | 1,419,731.1260 | 346.6140 | 2.9543 |
| | 102.2790 | 1,432,093.8350 | 349.6323 | 2.9288 |
| Average | 104.0472 | 1,565,124.9794 | 382.1106 | 2.6799 |
| Hawaii-3 | 87.0770 | 1,611,057.6000 | 393.3246 | 10.4138 |
| | 87.9210 | 1,603,754.6910 | 391.5417 | 10.4612 |
| | 87.7230 | 1,565,076.9360 | 382.0989 | 10.7197 |
| | 87.7080 | 1,579,994.0800 | 385.7407 | 10.6185 |
| | 87.3960 | 1,645,885.5140 | 401.8275 | 10.1934 |
| Average | 87.5650 | 1,601,153.7642 | 390.9067 | 10.4782 |
| Hawaii-4 | 48.0490 | 2,767,060.8480 | 675.5520 | 24.2528 |
| | 24.9590 | 3,572,880.5100 | 872.2853 | 18.7828 |
| | 28.4700 | 3,742,864.0320 | 913.7852 | 17.9298 |
| | 31.0780 | 3,797,736.6620 | 927.1818 | 17.6708 |
| | 33.1030 | 3,829,660.1370 | 934.9756 | 17.5235 |
| Average | 33.1318 | 3,542,040.4378 | 864.7560 | 18.9464 |

Table C–2: Hawaii - System 2

| | Average Frame Rate | Average Triangle Rate | Average Data Units Rate | Average Transfer Cost |
|---|---|---|---|---|
| Hawaii-1 | 71.2270 | 283,343.9540 | 69.1758 | 3.7007 |
| | 71.7260 | 210,395.8420 | 51.3662 | 4.9838 |
| | 71.9220 | 196,835.4030 | 48.0555 | 5.3272 |
| | 71.2440 | 487,058.5960 | 118.9108 | 2.1529 |
| | 71.5330 | 411,361.8130 | 100.4301 | 2.5490 |
| Average | 71.5304 | 317,799.1216 | 77.5877 | 3.2995 |
| Hawaii-2 | 69.3670 | 379,982.4230 | 92.7691 | 11.0382 |
| | 69.6150 | 391,904.3380 | 95.6798 | 10.7024 |
| | 69.0600 | 416,665.1610 | 101.7249 | 10.0664 |
| | 67.8050 | 436,696.7290 | 106.6154 | 9.6046 |
| | 66.7980 | 461,557.0730 | 112.6848 | 9.0873 |
| Average | 68.5290 | 417,361.1448 | 101.8948 | 10.0496 |
| Hawaii-3 | 45.3610 | 408,571.0360 | 99.7488 | 41.0632 |
| | 45.5870 | 345,206.4490 | 84.2789 | 48.6005 |
| | 45.6470 | 505,290.5940 | 123.3620 | 33.2031 |
| | 45.6640 | 860,070.4920 | 209.9781 | 19.5068 |
| | 45.6130 | 1,150,606.4480 | 280.9098 | 14.5812 |
| Average | 45.5744 | 653,949.0038 | 159.6555 | 25.6552 |
| Hawaii-4 | 34.1870 | 1,148,740.6580 | 280.4543 | 58.4195 |
| | 36.2350 | 1,217,566.3600 | 297.2574 | 55.1172 |
| | 36.4020 | 1,342,634.6860 | 327.7917 | 49.9830 |
| | 36.6930 | 1,383,341.2270 | 337.7298 | 48.5122 |
| | 36.5960 | 1,488,690.3150 | 363.4498 | 45.0791 |
| Average | 36.0226 | 1,316,194.6492 | 321.3366 | 50.9870 |

Table C–3: Hawaii - System 3

| | Average Frame Rate | Average Triangle Rate | Average Data Units Rate | Average Transfer Cost |
|---|---|---|---|---|
| Hawaii-1 | 19.7580 | 118,323.8900 | 28.8877 | 8.8619 |
| | 19.7570 | 150,692.7640 | 36.7902 | 6.9584 |
| | 19.8280 | 126,818.3000 | 30.9615 | 8.2683 |
| | 19.8640 | 164,556.7210 | 40.1750 | 6.3721 |
| | 19.8310 | 143,370.4510 | 35.0026 | 7.3138 |
| Average | 19.8076 | 140,752.4252 | 34.3634 | 7.4498 |
| Hawaii-2 | 19.9080 | 199,221.3390 | 48.6380 | 21.0535 |
| | 19.5980 | 176,930.0820 | 43.1958 | 23.7060 |
| | 19.5220 | 185,181.8080 | 45.2104 | 22.6497 |
| | 19.3620 | 223,412.7450 | 54.5441 | 18.7738 |
| | 19.3060 | 229,222.1160 | 55.9624 | 18.2980 |
| Average | 19.5392 | 202,793.6180 | 49.5102 | 20.6826 |
| Hawaii-3 | 19.2160 | 437,830.8500 | 106.8923 | 38.3189 |
| | 18.9310 | 466,990.6120 | 114.0114 | 35.9262 |
| | 19.1570 | 591,549.3400 | 144.4212 | 28.3615 |
| | 19.1970 | 779,501.9520 | 190.3081 | 21.5230 |
| | 19.2480 | 853,732.9420 | 208.4309 | 19.6516 |
| Average | 19.1498 | 625,921.1392 | 152.8128 | 26.8040 |
| Hawaii-4 | 19.0660 | 518,728.2990 | 126.6427 | 129.3719 |
| | 18.9720 | 5,493,227.9300 | 1,341.1201 | 12.2167 |
| | 18.8510 | 649,912.8480 | 158.6701 | 103.2583 |
| | 18.8950 | 838,357.9760 | 204.6772 | 80.0480 |
| | 18.7230 | 939,584.6070 | 229.3908 | 71.4240 |
| Average | 18.9014 | 1,687,962.3320 | 412.1002 | 39.7573 |

Table C–4: Grand Canyon - System 1

| | Average Frame Rate | Average Triangle Rate | Average Data Units Rate | Average Transfer Cost |
|---|---|---|---|---|
| GrandCanyon-1 | 61.4970 | 9,710,174.5470 | 2,370.6481 | 0.0135 |
| | 63.8110 | 9,452,455.7330 | 2,307.7285 | 0.0139 |
| | 61.8050 | 9,822,218.3760 | 2,398.0025 | 0.0133 |
| | 61.8350 | 9,894,420.0310 | 2,415.6299 | 0.0132 |
| | 61.7950 | 9,960,051.1830 | 2,431.6531 | 0.0132 |
| Average | 62.1486 | 9,767,863.9740 | 2,384.7324 | 0.0134 |
| GrandCanyon-2 | 55.0210 | 10,632,766.5240 | 2,595.8903 | 0.0493 |
| | 52.6120 | 10,982,656.6060 | 2,681.3126 | 0.0477 |
| | 53.6270 | 10,878,158.0580 | 2,655.8003 | 0.0482 |
| | 53.5530 | 10,936,283.7790 | 2,669.9912 | 0.0479 |
| | 52.8010 | 11,061,772.0270 | 2,700.6279 | 0.0474 |
| Average | 53.5228 | 10,898,327.3988 | 2,660.7245 | 0.0481 |
| GrandCanyon-3 | 42.8420 | 11,859,543.8970 | 2,895.3965 | 0.1768 |
| | 41.7460 | 11,829,330.1440 | 2,888.0201 | 0.1773 |
| | 40.9310 | 11,911,592.8720 | 2,908.1037 | 0.1761 |
| | 40.6150 | 11,966,043.2080 | 2,921.3973 | 0.1753 |
| | 40.7040 | 11,972,784.8300 | 2,923.0432 | 0.1752 |
| Average | 41.3676 | 11,907,858.9902 | 2,907.1921 | 0.1761 |
| GrandCanyon-4 | 43.3760 | 11,171,963.7990 | 2,727.5302 | 0.7509 |
| | 40.5500 | 11,721,384.2120 | 2,861.6661 | 0.7157 |
| | 39.8680 | 11,873,108.9980 | 2,898.7083 | 0.7065 |
| | 38.7800 | 12,082,441.9270 | 2,949.8149 | 0.6943 |
| | 38.5390 | 12,142,887.6550 | 2,964.5722 | 0.6908 |
| Average | 40.2226 | 11,798,357.3182 | 2,880.4583 | 0.7110 |

Table C–5: Grand Canyon - System 2

| | Average Frame Rate | Average Triangle Rate | Average Data Units Rate | Average Transfer Cost |
|---|---|---|---|---|
| GrandCanyon-1 | 43.0330 | 7,897,260.6560 | 1,928.0422 | 0.0166 |
| | 43.2160 | 8,031,949.5470 | 1,960.9252 | 0.0163 |
| | 43.9380 | 7,905,686.8300 | 1,930.0993 | 0.0166 |
| | 45.6890 | 7,519,374.0300 | 1,835.7847 | 0.0174 |
| | 45.4050 | 7,631,244.6560 | 1,863.0968 | 0.0172 |
| Average | 44.2562 | 7,797,103.1438 | 1,903.5896 | 0.0168 |
| GrandCanyon-2 | 38.5590 | 8,525,297.4410 | 2,081.3714 | 0.0615 |
| | 38.6450 | 8,598,043.7370 | 2,099.1318 | 0.0610 |
| | 37.7230 | 8,912,036.2660 | 2,175.7901 | 0.0588 |
| | 37.4110 | 9,042,751.4640 | 2,207.7030 | 0.0580 |
| | 37.2500 | 9,137,023.6170 | 2,230.7187 | 0.0574 |
| Average | 37.9176 | 8,843,030.5050 | 2,158.9430 | 0.0593 |
| GrandCanyon-3 | 29.2990 | 10,317,839.4290 | 2,519.0038 | 0.2033 |
| | 30.4760 | 10,036,472.7640 | 2,450.3107 | 0.2090 |
| | 31.1800 | 9,936,631.8630 | 2,425.9355 | 0.2111 |
| | 31.9540 | 9,905,673.8900 | 2,418.3774 | 0.2117 |
| | 32.5290 | 9,864,055.1570 | 2,408.2166 | 0.2126 |
| Average | 31.0876 | 10,012,134.6206 | 2,444.3688 | 0.2095 |
| GrandCanyon-4 | 30.0830 | 10,105,542.7860 | 2,467.1735 | 0.8301 |
| | 29.5270 | 10,195,986.9320 | 2,489.2546 | 0.8227 |
| | 29.6990 | 10,123,939.5020 | 2,471.6649 | 0.8286 |
| | 29.8230 | 10,076,120.9600 | 2,459.9905 | 0.8325 |
| | 29.8150 | 10,062,648.5810 | 2,456.7013 | 0.8336 |
| Average | 29.7894 | 10,112,847.7522 | 2,468.9570 | 0.8295 |

Table C–6: Grand Canyon - System 3

| | Average Frame Rate | Average Triangle Rate | Average Data Units Rate | Average Transfer Cost |
|---|---|---|---|---|
| GrandCanyon-1 | 16.8150 | 2,999,472.2550 | 732.2930 | 0.0437 |
| | 16.9300 | 2,885,825.2910 | 704.5472 | 0.0454 |
| | 16.8300 | 2,974,266.2580 | 726.1392 | 0.0441 |
| | 16.7560 | 3,009,608.3870 | 734.7677 | 0.0436 |
| | 16.7420 | 3,037,671.1700 | 741.6189 | 0.0431 |
| Average | 16.8146 | 2,981,368.6722 | 727.8732 | 0.0440 |
| GrandCanyon-2 | 16.5990 | 3,103,467.9730 | 757.6826 | 0.1689 |
| | 16.5510 | 3,096,916.3690 | 756.0831 | 0.1693 |
| | 16.6900 | 3,032,959.5950 | 740.4687 | 0.1729 |
| | 16.8130 | 2,989,662.9960 | 729.8982 | 0.1754 |
| | 16.7370 | 3,052,770.6450 | 745.3053 | 0.1717 |
| Average | 16.6780 | 3,055,155.5156 | 745.8876 | 0.1716 |
| GrandCanyon-3 | 14.4570 | 4,801,723.2380 | 1,172.2957 | 0.4367 |
| | 14.3800 | 4,795,820.0510 | 1,170.8545 | 0.4373 |
| | 14.3800 | 4,811,523.9100 | 1,174.6885 | 0.4359 |
| | 14.3970 | 4,799,237.6290 | 1,171.6889 | 0.4370 |
| | 14.4170 | 4,778,963.9120 | 1,166.7392 | 0.4388 |
| Average | 14.4062 | 4,797,453.7480 | 1,171.2534 | 0.4371 |
| GrandCanyon-4 | 13.2290 | 5,722,011.9065 | 1,396.9756 | 1.4660 |
| | 12.9944 | 5,822,200.5377 | 1,421.4357 | 1.4408 |
| | 13.2334 | 5,688,176.3117 | 1,388.7149 | 1.4747 |
| | 13.3322 | 5,635,053.4169 | 1,375.7455 | 1.4886 |
| | 13.4028 | 5,606,487.2520 | 1,368.7713 | 1.4962 |
| Average | 13.2384 | 5,694,785.8850 | 1,390.3286 | 1.4730 |

Table C–7: Guanica - System 1

| | Average Frame Rate | Average Triangle Rate | Average Data Units Rate | Average Transfer Cost |
|---|---|---|---|---|
| Guanica-1 | 75.3310 | 7,128,390.9720 | 1,740.3298 | 0.1747 |
| | 75.8740 | 7,301,432.6260 | 1,782.5763 | 0.1705 |
| | 77.3680 | 7,184,737.9910 | 1,754.0864 | 0.1733 |
| | 77.9910 | 7,148,153.5120 | 1,745.1547 | 0.1742 |
| | 78.8120 | 7,086,939.8260 | 1,730.2099 | 0.1757 |
| Average | 77.0752 | 7,169,930.9854 | 1,750.4714 | 0.1737 |
| Guanica-2 | 59.7680 | 8,526,711.6800 | 2,081.7167 | 0.5841 |
| | 59.2480 | 8,760,429.2720 | 2,138.7767 | 0.5685 |
| | 59.2090 | 8,903,042.7010 | 2,173.5944 | 0.5594 |
| | 59.6640 | 8,927,000.9630 | 2,179.4436 | 0.5579 |
| | 60.7710 | 8,806,303.9690 | 2,149.9766 | 0.5656 |
| Average | 59.7320 | 8,784,697.7170 | 2,144.7016 | 0.5670 |
| Guanica-3 | 49.4170 | 7,966,159.4300 | 1,944.8631 | 2.5009 |
| | 44.2130 | 8,951,928.1000 | 2,185.5293 | 2.2255 |
| | 36.4510 | 10,571,582.1050 | 2,580.9527 | 1.8846 |
| | 31.0700 | 11,707,537.9210 | 2,858.2856 | 1.7017 |
| | 28.9570 | 12,102,677.8000 | 2,954.7553 | 1.6462 |
| Average | 38.0216 | 10,259,977.0712 | 2,504.8772 | 1.9418 |
| Guanica-4 | 22.6890 | 10,775,315.2730 | 2,630.6922 | 7.3958 |
| | 22.7130 | 10,786,638.8690 | 2,633.4568 | 7.3880 |
| | 22.7440 | 10,801,651.8430 | 2,637.1220 | 7.3777 |
| | 22.7420 | 10,800,735.2220 | 2,636.8982 | 7.3784 |
| | 22.0800 | 10,995,974.0830 | 2,684.5640 | 7.2474 |
| Average | 22.5936 | 10,832,063.0580 | 2,644.5466 | 7.3570 |

Table C–8: Guanica - System 2

| | Average Frame Rate | Average Triangle Rate | Average Data Units Rate | Average Transfer Cost |
|---|---|---|---|---|
| Guanica-1 | 50.1900 | 5,504,152.8740 | 1,343.7873 | 0.2262 |
| | 51.6690 | 5,289,301.1940 | 1,291.3333 | 0.2354 |
| | 51.8010 | 5,282,053.7530 | 1,289.5639 | 0.2357 |
| | 51.9490 | 5,258,734.5770 | 1,283.8707 | 0.2368 |
| | 52.3940 | 5,174,324.7020 | 1,263.2629 | 0.2406 |
| Average | 51.6006 | 5,301,713.4200 | 1,294.3636 | 0.2349 |
| Guanica-2 | 55.4790 | 3,928,543.6320 | 959.1171 | 1.2678 |
| | 49.5510 | 5,278,258.7750 | 1,288.6374 | 0.9436 |
| | 41.6760 | 7,066,865.2760 | 1,725.3089 | 0.7048 |
| | 37.8160 | 7,945,254.8530 | 1,939.7595 | 0.6269 |
| | 39.6910 | 7,532,168.3630 | 1,838.9083 | 0.6613 |
| Average | 44.8426 | 6,350,218.1798 | 1,550.3462 | 0.7843 |
| Guanica-3 | 40.9460 | 6,215,674.6660 | 1,517.4987 | 3.2053 |
| | 23.2470 | 9,536,924.6860 | 2,328.3508 | 2.0890 |
| | 16.3050 | 11,704,605.4220 | 2,857.5697 | 1.7021 |
| | 17.3700 | 12,490,403.1570 | 3,049.4148 | 1.5951 |
| | 18.3070 | 8,012,624.6420 | 1,956.2072 | 2.4864 |
| Average | 23.2350 | 9,592,046.5146 | 2,341.8082 | 2.0770 |
| Guanica-4 | 24.2970 | 5,372,408.7350 | 1,311.6232 | 14.8335 |
| | 17.6160 | 10,440,361.6940 | 2,548.9164 | 7.6330 |
| | 12.6060 | 12,412,518.1890 | 3,030.3999 | 6.4203 |
| | 10.8160 | 12,147,222.7500 | 2,965.6306 | 6.5605 |
| | 9.8600 | 11,916,103.2390 | 2,909.2049 | 6.6877 |
| Average | 15.0390 | 10,457,722.9214 | 2,553.1550 | 7.6204 |

Table C–9: Guanica - System 3

| | Average Frame Rate | Average Triangle Rate | Average Data Units Rate | Average Transfer Cost |
|---|---|---|---|---|
| Guanica-1 | 17.9790 | 1,651,789.0190 | 403.2688 | 0.7538 |
| | 14.4720 | 1,311,018.4800 | 320.0729 | 0.9498 |
| | 13.8170 | 1,145,729.3230 | 279.7191 | 1.0868 |
| | 14.3300 | 1,039,513.9720 | 253.7876 | 1.1979 |
| | 14.4130 | 1,066,665.4060 | 260.4164 | 1.1674 |
| Average | 15.0022 | 1,242,943.2400 | 303.4529 | 1.0018 |
| Guanica-2 | 17.7960 | 1,594,918.5410 | 389.3844 | 3.1229 |
| | 14.6160 | 1,959,006.1500 | 478.2730 | 2.5425 |
| | 14.0980 | 2,368,975.3330 | 578.3631 | 2.1025 |
| | 15.0760 | 2,114,299.2700 | 516.1863 | 2.3557 |
| | 15.4970 | 2,097,584.6340 | 512.1056 | 2.3745 |
| Average | 15.4166 | 2,026,956.7856 | 494.8625 | 2.4572 |
| Guanica-3 | 16.0160 | 1,855,491.8880 | 453.0009 | 10.7373 |
| | 13.9400 | 3,417,836.2020 | 834.4327 | 5.8291 |
| | 12.1950 | 4,670,416.3870 | 1,140.2384 | 4.2658 |
| | 12.5400 | 4,452,482.1440 | 1,087.0318 | 4.4746 |
| | 12.8710 | 4,244,947.1510 | 1,036.3641 | 4.6933 |
| Average | 13.5124 | 3,728,234.7544 | 910.2136 | 5.3438 |
| Guanica-4 | 14.4076 | 1,912,362.3660 | 466.8853 | 41.6719 |
| | 13.7960 | 4,065,823.8720 | 992.6328 | 19.6004 |
| | 11.9140 | 5,893,662.3970 | 1,438.8824 | 13.5216 |
| | 12.1702 | 5,527,267.4420 | 1,349.4305 | 14.4179 |
| | 11.7870 | 5,275,866.3790 | 1,288.0533 | 15.1050 |
| Average | 12.8150 | 4,534,996.4912 | 1,107.1769 | 17.5726 |

Table C–10: Jobos Bay - System 1

| | Average Frame Rate | Average Triangle Rate | Average Data Units Rate | Average Transfer Cost |
|---|---|---|---|---|
| JobosBay-1 | 84.6430 | 6,184,379.2140 | 1,509.8582 | 0.1142 |
| | 77.9450 | 7,142,400.9730 | 1,743.7502 | 0.0989 |
| | 79.7190 | 6,977,201.3500 | 1,703.4183 | 0.1013 |
| | 78.9590 | 7,102,733.0050 | 1,734.0657 | 0.0995 |
| | 77.8170 | 7,263,199.6450 | 1,773.2421 | 0.0973 |
| Average | 79.8166 | 6,933,982.8374 | 1,692.8669 | 0.1019 |
| JobosBay-2 | 76.5770 | 6,285,580.1620 | 1,534.5655 | 0.4496 |
| | 73.5460 | 6,495,802.6160 | 1,585.8893 | 0.4351 |
| | 73.2710 | 6,602,001.9750 | 1,611.8169 | 0.4281 |
| | 72.1510 | 6,736,625.5050 | 1,644.6840 | 0.4195 |
| | 71.0910 | 6,842,206.1690 | 1,670.4605 | 0.4131 |
| Average | 73.3272 | 6,592,443.2854 | 1,609.4832 | 0.4287 |
| JobosBay-3 | 83.5090 | 5,930,607.1000 | 1,447.9021 | 1.9062 |
| | 60.4260 | 5,713,688.2580 | 1,394.9434 | 1.9786 |
| | 53.7900 | 6,471,853.8740 | 1,580.0424 | 1.7468 |
| | 49.0150 | 6,621,814.6400 | 1,616.6540 | 1.7072 |
| | 46.1630 | 6,429,233.6120 | 1,569.6371 | 1.7584 |
| Average | 58.5806 | 6233439.497 | 1521.835815 | 1.8136 |
| JobosBay-4 | 38.8860 | 3,439,146.2990 | 839.6353 | 13.1486 |
| | 40.5220 | 7,259,869.1850 | 1,772.4290 | 6.2287 |
| | 37.0640 | 8,468,150.0230 | 2,067.4194 | 5.3400 |
| | 36.2390 | 9,303,622.9300 | 2,271.3923 | 4.8605 |
| | 37.1430 | 9,757,996.1870 | 2,382.3233 | 4.6341 |
| Average | 37.9708 | 7645756.925 | 1866.639874 | 5.9144 |

Table C–11: Jobos Bay - System 2

| | Average Frame Rate | Average Triangle Rate | Average Data Units Rate | Average Transfer Cost |
|---|---|---|---|---|
| JobosBay-1 | 71.9920 | 1,608,178.7800 | 392.6218 | 0.4394 |
| | 68.9420 | 1,629,166.5050 | 397.7457 | 0.4337 |
| | 61.6820 | 1,635,102.7210 | 399.1950 | 0.4321 |
| | 60.7210 | 1,640,426.9640 | 400.4949 | 0.4307 |
| | 60.3260 | 1,641,065.7380 | 400.6508 | 0.4305 |
| Average | 64.7326 | 1,630,788.1416 | 398.1416 | 0.4333 |
| JobosBay-2 | 53.1430 | 199,019.6580 | 48.5888 | 14.2008 |
| | 53.5810 | 1,436,996.5060 | 350.8292 | 1.9668 |
| | 54.0040 | 3,382,068.7230 | 825.7004 | 0.8357 |
| | 54.1910 | 3,722,827.9010 | 908.8935 | 0.7592 |
| | 54.2130 | 3,906,170.6760 | 953.6550 | 0.7235 |
| Average | 53.8264 | 2,529,416.6928 | 617.5334 | 1.1173 |
| JobosBay-3 | 49.0520 | 4,321,376.2150 | 1,055.0235 | 2.6161 |
| | 50.3510 | 4,360,097.4260 | 1,064.4769 | 2.5928 |
| | 46.2410 | 4,385,055.9800 | 1,070.5703 | 2.5781 |
| | 43.7910 | 4,396,349.4800 | 1,073.3275 | 2.5714 |
| | 43.6460 | 4,407,833.3000 | 1,076.1312 | 2.5647 |
| Average | 46.6162 | 4374142.48 | 1067.905879 | 2.5845 |
| JobosBay-4 | 35.9660 | 3,471,771.7990 | 847.6005 | 13.0250 |
| | 38.8630 | 3,471,082.5910 | 847.4323 | 13.0276 |
| | 40.0210 | 4,683,770.2680 | 1,143.4986 | 9.6546 |
| | 40.6810 | 5,423,847.3530 | 1,324.1815 | 8.3372 |
| | 41.0220 | 5,359,847.6170 | 1,308.5565 | 8.4368 |
| Average | 39.3106 | 4482063.926 | 1094.253888 | 10.0891 |

Table C–12: Jobos Bay - System 3

| | Average Frame Rate | Average Triangle Rate | Average Data Units Rate | Average Transfer Cost |
|---|---|---|---|---|
| JobosBay-1 | 20.1320 | 851,028.6610 | 207.7707 | 0.8302 |
| | 19.6950 | 789,567.8760 | 192.7656 | 0.8949 |
| | 19.2440 | 856,602.7030 | 209.1315 | 0.8248 |
| | 19.2680 | 838,145.4660 | 204.6254 | 0.8430 |
| | 19.3740 | 766,004.5140 | 187.0128 | 0.9224 |
| Average | 19.5426 | 820,269.8440 | 200.2612 | 0.8614 |
| JobosBay-2 | 18.0540 | 421,533.6870 | 102.9135 | 6.7047 |
| | 17.9480 | 815,673.5160 | 199.1390 | 3.4649 |
| | 17.9750 | 1,211,060.8120 | 295.6691 | 2.3337 |
| | 18.0990 | 1,223,078.5970 | 298.6032 | 2.3108 |
| | 18.2540 | 1,162,914.3270 | 283.9146 | 2.4303 |
| Average | 18.0660 | 966,852.1878 | 236.0479 | 2.9231 |
| JobosBay-3 | 18.4910 | 1,873,390.7200 | 457.3708 | 6.03 |
| | 18.5260 | 1,430,298.3220 | 349.1939 | 7.90 |
| | 16.8250 | 1,089,709.3350 | 266.0423 | 10.37 |
| | 15.0900 | 1,044,197.4780 | 254.9310 | 10.83 |
| | 15.0950 | 1,003,675.4010 | 245.0379 | 11.26 |
| Average | 16.8054 | 1,288,254.2512 | 314.5152 | 8.78 |
| JobosBay-4 | 16.4130 | 1,443,895.7460 | 352.5136 | 31.32 |
| | 16.7790 | 1,456,403.9620 | 355.5674 | 31.05 |
| | 15.5560 | 1,444,167.4440 | 352.5799 | 31.31 |
| | 13.9210 | 1,429,130.6090 | 348.9088 | 31.64 |
| | 13.9750 | 1,400,585.2140 | 341.9397 | 32.29 |
| Average | 15.3288 | 1,434,836.5950 | 350.3019 | 31.52 |

# APPENDIX D
# TEST CASE RESULTS COMPARISON

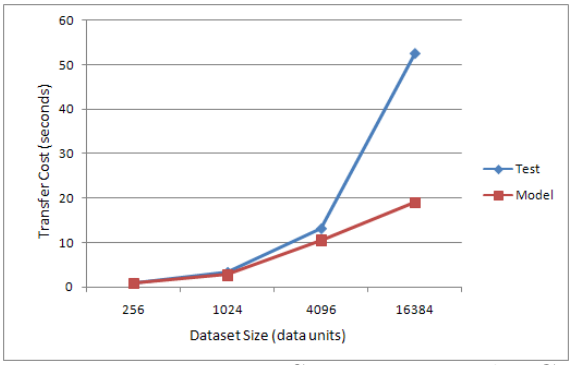


Figure D–1: Hawaii Dataset - System 1 Results Comparison

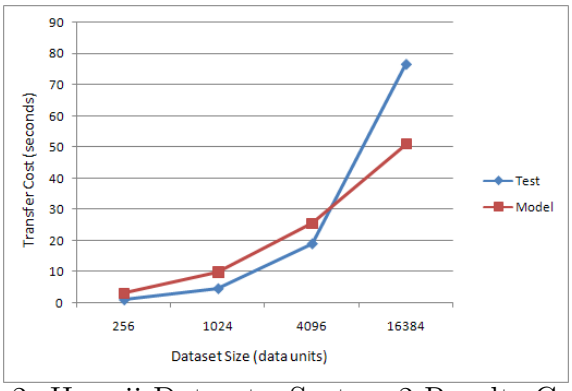


Figure D–2: Hawaii Dataset - System 2 Results Comparison

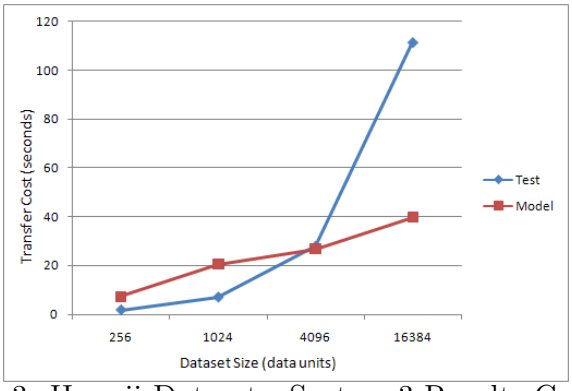


Figure D–3: Hawaii Dataset - System 3 Results Comparison

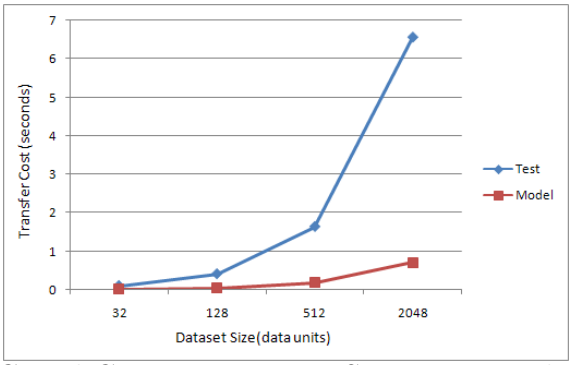



Figure D–4: Grand Canyon Dataset - System 1 Results Comparison

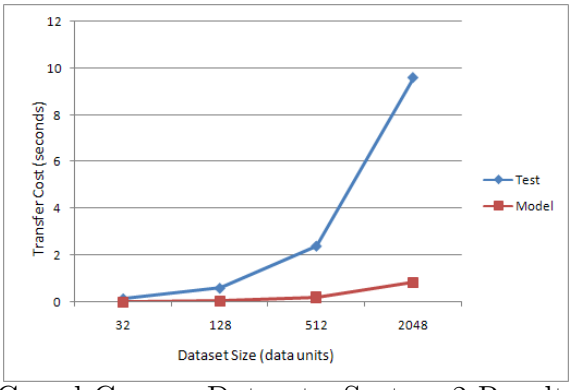



Figure D–5: Grand Canyon Dataset - System 2 Results Comparison

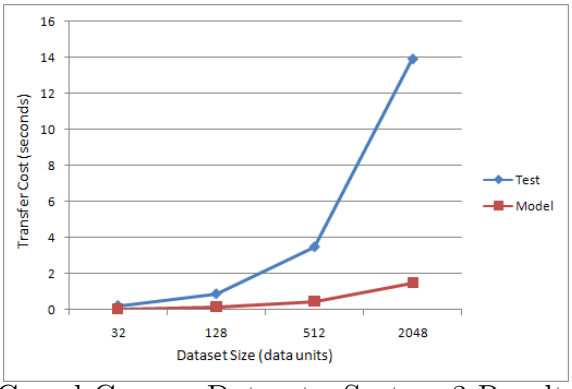



Figure D–6: Grand Canyon Dataset - System 3 Results Comparison

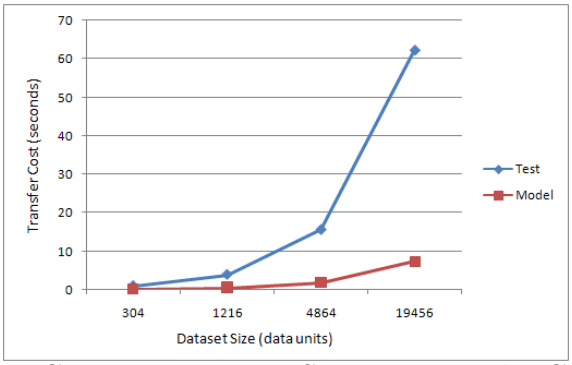



Figure D–7: Guanica Dataset - System 1 Results Comparison

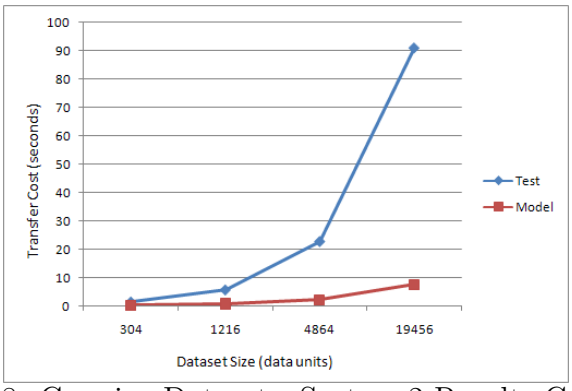



Figure D–8: Guanica Dataset - System 2 Results Comparison

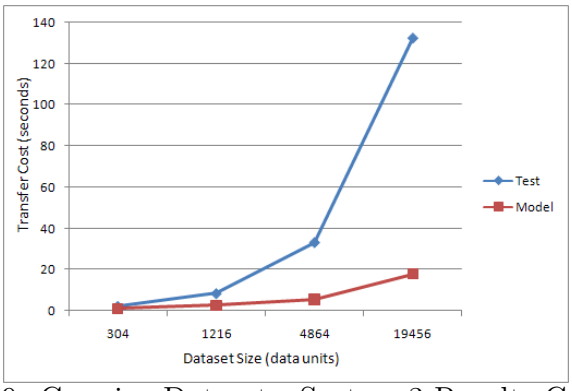



Figure D–9: Guanica Dataset - System 3 Results Comparison

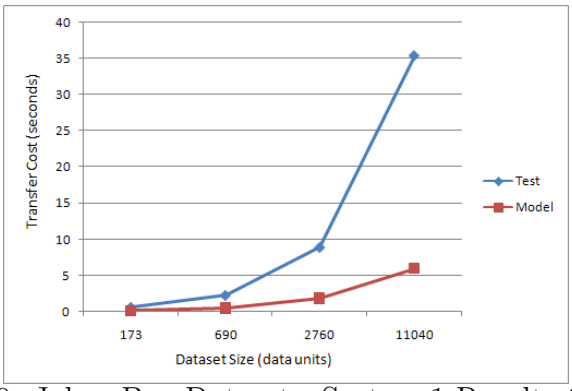



Figure D–10: Jobos Bay Dataset - System 1 Results Comparison

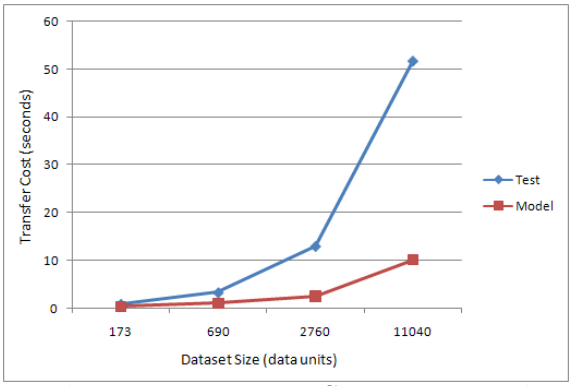



Figure D–11: Jobos Bay Dataset - System 2 Results Comparison

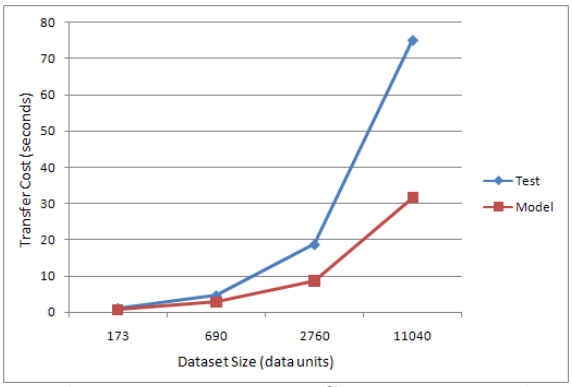



Figure D–12: Jobos Bay Dataset - System 3 Results Comparison

REFERENCE LIST


[1] A.H. Watt and M. Watt. *Advanced Animation and Rendering Techniques: Theory and Practice*. Addison-Wesley Pub., 1992.

[2] C.D. Hansen and C.R. Johnson. *The Visualization Handbook*. Elsevier, 2005.

[3] D.A. Patterson and J.L. Hennessy. *Computer organization & design: the hardware/software interface*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1993.

[4] David Luebke, Benjamin Watson, Jonathan D. Cohen, Martin Reddy, and Amitabh Varshney. *Level of Detail for 3D Graphics*. Elsevier Science Inc., New York, NY, USA, 2002.

[5] J.H. Clark. Hierarchical Geometric Models for Visible Surface Algorithms. *Communications of the ACM*, 19(10):547–554, 1976.

[6] R.J. Fowler and J.J. Little. Automatic Extraction of Irregular Network Digital Terrain Models. *Proceedings of the 6th annual conference on Computer graphics and interactive techniques*, pages 199–207, 1979.

[7] H. Samet. The Quadtree and Related Hierarchical Data Structures. *ACM Computing Surveys (CSUR)*, 16(2):187–260, 1984.

[8] E. Langetepe and G. Zachmann. *Geometric Data Structures for Computer Graphics*. A. K. Peters, Ltd., Natick, MA, USA, 2006.

[9] A. Ogren. Continuous Level of Detail In Real-Time Terrain Rendering. *Sweden: Sweden Umea University*, 2000.

[10] E. Puppo. Variable resolution triangulations. *Computational Geometry: Theory and Applications*, 11(3-4):219–238, 1998.

[11] Y.G. Leclerc and S.Q. Lau. TerraVision: A Terrain Visualization System. *SRI International, Technical Note*, 540, 1994.

[12] M. Reddy, Y. Leclerc, L. Iverson, and N. Bletter. TerraVision II: Visualizing Massive Terrain Databases in VRML. *Computer Graphics and Applications, IEEE*, 19(2):30–38, 1999.

[13] L. Williams. Pyramidal parametrics. *Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, pages 1–11, 1983.

[14] C. Zach, S. Mantler, and K. Karner. Time-critical rendering of discrete and continuous levels of detail. *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 1–8, 2002.

[15] P. Lindstrom, D. Koller, W. Ribarsky, L.F. Hodges, N. Faust, and G.A. Turner. *Real-time, continuous level of detail rendering of height fields*. ACM Press New York, NY, USA, 1996.

[16] H. Hoppe. Progressive meshes. *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 99–108, 1996.

[17] H. Hoppe. View-dependent Refinement of Progressive Meshes. *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 189–198, 1997.

[18] H. Hoppe. Smooth View-Dependent Level-of-Detail Control and Its Application to Terrain Rendering. *IEEE Visualization '98*, 31:35–42, 1998.

[19] W.H. de Boer. Fast Terrain Rendering Using Geometrical MipMapping. *Unpublished paper, available at http://www. flipcode. com/articles/article geomipmaps.pdf*, 2000.

[20] J. Levenberg. Fast view-dependent level-of-detail rendering using cached geometry. *Proceedings IEEE Visualization'02*, pages 259–266, 2002.

[21] S.E. Yoon, B. Salomon, R. Gayle, and D. Manocha. Quick-VDR: Interactive View-Dependent Rendering of Massive Models. *IEEE Visualization 2004*, pages

131–138, 2004.

[22] Y. Zhu. Uniform Remeshing with an Adaptive Domain: A New Scheme for View-Dependent Level-of-Detail Rendering of Meshes. *Visualization and Computer Graphics, IEEE Transactions on*, 11(3):306–316, 2005.

[23] F. Losasso and H. Hoppe. Geometry Clipmaps: Terrain Rendering Using Nested Regular Grids. *International Conference on Computer Graphics and Interactive Techniques*, pages 769–776, 2004.

[24] Anders Brodersen. Real-time visualization of large textured terrains. In *GRAPHITE '05: Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 439–442, 2005.

[25] P. Lindstrom and V. Pascucci. Visualization of Large Terrains Made Easy. *Proceedings of the conference on Visualization '01*, pages 363–371, 2001.

[26] X. Bao and R. Pajarola. LOD-based Clustering Techniques for Efficient Large-scale Terrain Storage and Visualization. *Proceedings SPIE Conference on Visualization and Data Analysis*, 235, 2003.

[27] P. Cignoni, C. Montani, C. Rocchini, and R. Scopigno. External Memory Management and Simplification of Huge Meshes. *IEEE Transactions on Visualization and Computer Graphics*, 9(4):525–537, 2003.

[28] O.G. Staadt, M.H. Gross, and R. Weber. Multiresolution compression and reconstruction. *Proceedings of IEEE Visualization 1997*, 1997.

[29] A. Khodakovsky, P. Schroder, and W. Sweldens. Progressive geometry compression. *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 278, 2000.

[30] PH Chou and TH Meng. Vertex data compression through vector quantization. *Visualization and Computer Graphics, IEEE Transactions on*, 8(4):373–382, 2002.

[31] D.S.P. To, R.W.H. Lau, and M. Green. A method for progressive and selective transmission of multi-resolution models. *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 88–95, 1999.

[32] A. Kalaiah and A. Varshney. Statistical geometry representation for efficient transmission and rendering. *ACM Transactions on Graphics (TOG)*, 24(2):348–373, 2005.

[33] S. Deb, PJ Narayanan, and S. Bhattacharjee. Streaming terrain rendering. *International Conference on Computer Graphics and Interactive Techniques*, 2006.

[34] J. Schneider and R. Westermann. GPU-Friendly High-Quality Terrain Rendering. *Journal of WSCG (S1213-6972)*, 14(1):49–56, 2006.

[35] T. Boubekeur and C. Schlick. Generic Mesh Refinement on GPU. *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 99–104, 2005.

[36] Robert Sisneros, Chad Jones, Jian Huang, Jinzhu Gao, Byung-Hoon Park, and Nagiza Samatova. A multi-level cache model for run-time optimization of remote visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(5):991–1003, 2007.

[37] R.G. Carter. Pipeline optimization: dynamic programming after 30 years. *Proceedings of the 30th PSIG Annual Meeting*, 1998.

[38] I.H. Osman. *Meta-Heuristics: Theory and Applications*. Springer, 1996.

[39] *Preliminary Discussion of the Logical Design of an Electronic Computing Instrument*. Inst. of Advanced Study, Princeton, N.J., 1946.

[40] The walsaip project. http://walsaip.uprm.edu/.

[41] J. McAffer and J.M. Lemieux. *Eclipse Rich Client Platform: Designing, Coding, and Packaging Java (TM) Applications*. Addison-Wesley Professional, 2005.

[42] Opengl - the industry standard for high performance graphics. http://www.opengl.org.

[43] Jogl: Java binding for opengl. https://jogl.dev.java.net.

[44] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1995.

[45] S. Burbeck. Applications Programming in Smalltalk-80 (TM): How to use Model-View-Controller (MVC). Technical report, Softsmarts Inc., 1987.

[46] B. Venners. *Inside the Java Virtual Machine*. McGraw-Hill Professional, 1999.

[47] W. Gu and N. Burns. Evolution of a high-performing Java virtual machine. *IBM Systems Journal*, 39(1):135–150, 2000.

[48] S. Microsystems. The Java Hotspot Performance Engine Architecture. *White paper (http://java. sun. com/products/hotspot/whitepaper. html), Sun Microsystems, April*, 2006.

[49] Eclipse test & performance tools platform project. http://www.eclipse.org/tptp/.

## BIOGRAPHICAL SKETCH

Ricardo Veguilla González was born in June 29, 1976 in Hato Rey, Puerto Rico. Ricardo is son of Eduardo Veguilla Zayas and Carmen S. González Flores. He received the Bachelors Degree in Computer Engineering from the University of Puerto Rico, Mayagüez Campus in May 2005 with specialization in the area of Computing Systems. He has worked as software developer, system administrator, and software tester. He is currently pursuing his M.S. degree in Computer Engineering at the University of Puerto Rico, Mayagüez Campus. His area of interest includes software engineering, design patterns, object-oriented programming, visualization, automated information processing, and distributed computing. He is particularly interested in the design of software architectures and middleware for the integration of diverse information coming from and being processed by distributed resources and how to exploit such technologies in order to easily provide people with the information they need. Mr. Veguilla is a member of the ACM.

# A HIERARCHICAL MEMORY MODEL FOR TERRAIN DATA MANAGEMENT IN INTERACTIVE TERRAIN VISUALIZATION

Ricardo Veguilla González
(787) 365-3288
Department of Electrical and Computer Engineering
Chair: Nayda G. Santiago
Degree: Master of Science
Graduation Date: December 2007