4-21-2010

Data Structures And Techniques For Visualization Of Large Volumetric Carbon Dioxide Datasets In A Real Time Experience

Jason B. Lambert
Purdue University, shotgunkiwi@gmail.com

Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Jason Lambert

Entitled Data Structures and Techniques for Visualization of Large Volumetric Carbon Dioxide Datasets in a Real Time Experience

For the degree of Master of Science

Is approved by the final examining committee:

Dr. Bedrich Benes
Chair

Dr. Kevin Gurney

Dr. James Mohler

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Dr. Bedrich Benes

Approved by: Dr. James Mohler 4-16-10
Head of the Graduate Program Date

Graduate School Form 20
(Revised 1/10)

PURDUE UNIVERSITY
GRADUATE SCHOOL

Research Integrity and Copyright Disclaimer

Title of Thesis/Dissertation:

Data Structures and techniques for Visualization of Large Volumetric Carbon Dioxide Datasets in a Real Time Experience

For the degree of Master of Science

I certify that in the preparation of this thesis, I have observed the provisions of *Purdue University Teaching, Research, and Outreach Policy on Research Misconduct (VIII.3.1)*, October 1, 2008.*

Further, I certify that this work is free of plagiarism and all materials appearing in this thesis/dissertation have been properly quoted and attributed.

I certify that all copyrighted material incorporated into this thesis/dissertation is in compliance with the United States' copyright law and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and save harmless Purdue University from any and all claims that may be asserted or that may arise from any copyright violation.

Jason Lambert

Printed Name and Signature of Candidate

4-19-2010

Date (month/day/year)

*Located at http://www.purdue.edu/policies/pages/teach_res_outreach/viii_3_1.html

DATA STRUCTURES AND TECHNIQUES FOR VISUALIZATION OF LARGE VOLUMETRIC CARBON DIOXIDE DATASETS IN A REAL TIME EXPERIENCE

A Thesis
Submitted to the Faculty
of
Purdue University
by
Jason B. Lambert

In Partial Fulfillment of the
Requirements for the Degree
of
Master of Science

May 2010
Purdue University
West Lafayette, Indiana

Dedicated to Grandpa Leonard, I just hope they print theses in Heaven.

ACKNOWLEDGMENTS

The author would like to thank his committee:

Dr. Bedrich Benes

Dr. Kevin Gurney

Dr. James Mohler

Additionally the author would like to thank all his family and friends that have made my time at graduate school so enjoyable.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

GLOSSARY

Within this document several terms are used that require definition; the definitions of these terms are as follows:

- CPU: Central Processing Unit. Processes all non-graphics graphics commands in traditional computer architecture.
- GPU: Graphics Processing Unit. The Computer Hardware used to render images to screen and perform graphics based calculations.
- RAMS: Regional Atmospheric Modelling System
- NACP: North American Carbon Program
- Voxel: A volume element part of a volumetric field - a “voxel” field – used to describe a complete volumetric space.
- C02 Viz: The application containing the implementation of this research and previous work by Nathan Andrysco
- Real-Time: A term taken in this research to define an interactive experience that allows user controlled change with near Instantaneous graphical feedback. Near instantaneous being defined at running with image generation speeds lower than 33ms.
- Dataset: The collection used throughout this research is combination of both geometric and 'real-world' data, that is, advected

measurements of carbon dioxide transmission within the earth's atmosphere.

- Offline Rendering: Previous attempts at visualization the NCAP dataset are considered 'offline' rendering methods, that is, that they do not provide immediate and interactive results. The controls and images are produced on demand and take a generation time of greater than 33ms for each image, and the resultant experience is less than 30fps.

- Pre-Process: Any function or program that performs its tasks prior to execution of the main rendering suite. A Pre-process is not considered part of rendering system of a real-time application.

- Pixel: An element of a frame-buffer containing color information.

- Normal: A vector defining a direction perpendicular to a surface.

- Vertex: A collection of coordinates defining a position in space.

- MB: (Megabytes) a measure of size often associated with data on disk.

- PPM: Parts per Million, a measure of concentration.

- Legacy Software: The previous visualization project completed in 2009 (Andrysco, Gurney, Benes, & Corbin, 2009).

- ASCII: American Standard Code for Information Interchange, used in this research to indicate data that is presented using regular characters and numbers that the data is readable to a human without machine decoding.

ABSTRACT

Lambert, Jason B. M.S., Purdue University, May, 2010. Data Structures and Techniques for Visualization of Large Volumetric Carbon Dioxide Datasets in a Real Time Experience. Major Professor: Bedrich Benes.

This thesis covers new research into real-time rendering of volumetric carbon dioxide data collected in the Vulcan project. The Vulcan project, a multi-disciplinary initiative to quantify carbon dioxide mass flux from residential, commercial and industrial sources headed by Gurney et al (Gurney, K. R., Mendoza, D. L., Zhou, Y., Fischer, M. L., Miller, C. C., Geethakumar, S., and de La Rue du Can, S. , 2009). The Vulcan datasets are a significant aid for policy makers, scientists and the general public alike as the collection was completed at a much finer space and time resolution than ever before.

A previous visualization attempt, completed in 2009 (Andrysco, Gurney, Benes, & Corbin, 2009) was able to visualize the data in an offline environment, noting constraints of data size and disk speed access as the most significant drawbacks for real-time visualization.

This thesis presents research towards a new real-time visualization suite in the areas of compression, data representation and simplification. The research hypothesizes that the use of these techniques will enable sufficient speed of rendering and loading to enable real-time data exploration.

The results show that a combination of techniques used in compression and the use of optimized indexed geometric structures allows the dataset to be explored and rendered in real time.

CHAPTER 1. INTRODUCTION

This chapter introduces the research by presenting the problem statement and associated research questions. The chapter concludes by defining the assumptions used as well the scope and significance of this particular research thesis.

1.1. Problem Statement

This research explores visualization and rendering data structures in order to determine the effect of key visualization algorithms and compression techniques when creating real-time visualizations the atmospheric carbon dioxide (CO_2) concentration fields, the Vulcan data. The Vulcan data is collected from commercial and federal data sources including airports, roads and domestic, commercial and industrial buildings is comprised of point, line and area mass flux data sources. The research will specifically explore the effects of compression and rendering algorithms on the size of the data on disk, and methods for loading and rendering the data at tune-time. Additionally this research helps scientific users to visualize one of the key carbon budget components of the North American Carbon Program (NACP); fossil fuel CO_2 emissions and their transport within the atmosphere as a 3D real-time experience.

The Vulcan dataset is 5 dimensional data of CO_2 concentration over time in volumetric space. With the complete dataset in the order of tens of gigabytes in size, there cannot be a real-time complete visualization of the true data source as indicated by Andrysco (2009). This is a problem traditionally solved with several strategies, data compression and blending between various discrete

levels of detail to simulate a continuous spectrum and changing the format of run-time rendering to a less data intensive format.

1.2. Research Question

This research focuses on a single primary question and breaks this down into several secondary research questions.

1.2.1. Primary Questions

- What are the most appropriate methods of compression, loading and rendering for real-time visualization of atmospheric Vulcan CO2 concentration data?

1.2.2. Secondary Questions

- What is the effect of data representation optimization?
- How can arbitrary surfaces be appropriately blended together to form a continuous spectrum across multiple ?
- How can memory be most efficiently managed to provide the smoothest loading of new data in real-time?
- What techniques offer an appropriate reduction in data size while maintaining integrity of the data
- What is the most appropriate rendering method given the previous work?

1.3. Scope

This research is limited to the Vulcan atmospheric CO_2 volumetric dataset used for previous research in 2009 (Andrysco, Gurney, Benes, & Corbin, 2009). However many of the concepts presented generalize into generic volumetric-over-time datasets. Additionally, the scope of this research is limited to creating a real-time visualization for the scientific community. As such, performance testing will be performed on a "high-end" computer work station, that is, a PC with advanced workstation hardware (for example a Quad-core CPU with more than 3GB of RAM and a G80 or higher GPU)

1.4. Significance

This research expands knowledge in the field of real-time volumetric rendering, offering methods for effective visualization of atmospheric carbon dioxide data. By providing a best-practices approach to volumetric rendering in real-time, this research serves as a guide to others attempting to bring a dataset traditionally viewed as "too large" into the real-time visualization domain. Additionally, by providing a novel interactive visualization of the Vulcan dataset, the research will allow scientific researchers to gain greater understanding of the more subtle, yet extremely important, nuances present within the dataset that were previously not visible in visualizations.

Not only is the graphics research significant, but the carbon dioxide data visualization is significant for furthering scientific knowledge and public awareness of the study of emissions. Politically, the study of emissions will drive a new host of environmental legislation, and the insight from visualizations produced by CO2 Viz will be an invaluable tool to new policy makers, scientists and consumers alike.

1.5. Assumptions

This research is performed and conclusions drawn using the following assumptions:

- The latest Vulcan dataset is representative of both past and future Vulcan datasets in its data format and size.
- Specific computer hardware and software used to implement solutions do not alter the generalization of the results unless specifically mentioned.

1.6. Delimitations

This research is performed acknowledging the following delimitations:

- The research will not be tested on every graphics card configuration to identify driver nuance effect on implementation.
- The pre-processing time is not included in the real-time performance evaluation.
- Other classes beyond the degree of the Vulcan volumetric datasets are not considered.
- The lighting and shading requirements for effective presentation to users of the software are not considered as variant.
- No other users apart from scientific professionals are considered, that is the usability of the testing software is outside the scope of this research.

1.7. Limitations

This research is limited by the following:

- The real-time rendering implementation uses pre-processed data and thus not all calculations must be performed at run-time.
- Performance is only considered on high-end consumer computer hardware.

1.8. Chapter Summary

This chapter introduced the research contained within this thesis, outlining the key research questions and variables. Additionally this chapter noted the limitations and delimitations of the chosen scope, and its contribution to the body of knowledge by explaining the significance of the research.

CHAPTER 2. LITERATURE REVIEW

This chapter provides a summary of recent research literature in the areas of volumetric visualization and the Vulcan project, providing both a base understanding of the methods in the subject area as well as motivation going forward to new methodology.

2.1. Introduction and Motivation

This thesis defines a suite of new rendering data structures alongside a new visualization application that extends a previous study (Andrysco, Gurney, Benes & Corbin, 2009) into the visualization of the Vulcan fossil fuel CO_2 emissions inventory as transformed by simulated atmospheric transport (Gurney, K. R., Mendoza, D. L., Zhou, Y., Fischer, M. L., Miller, C. C., Geethakumar, S., and de La Rue du Can, S. 2009)

The Vulcan project (Gurney, et al., 2009) accomplished quantification of fossil fuel CO_2 emissions over the U.S. in much greater space/time detail than previously achieved. The Vulcan emissions data product was input as the surface flux field to a *Regional Atmospheric Modeling System* (RAMS), a mesoscale atmospheric transport model developed at Colorado State University. Using RAMS, the project created a unique dataset (the “advected” data) of volumetric CO_2 emissions over the course of a full year in a much finer time-scale than ever achieved before. The atmospheric model showing the strong correlation of convection of the carbon dioxide along seasonal weather patterns raises important questions to many user-communities. As Andrysco (2009) states such interested user communities includes “educators, policymakers,

demographers and social scientists." The purpose of exposing user groups to this data is to aid in indentifying the total American carbon 'footprint', noting subtle changes in the atmospheric concentration over desired areas on the continental United States.

2.2. Previous Visualization project

Andrysco (2009) created a visualization suite, C02 Viz, for exploring the transport data created by Gurney (Gurney, et al., 2009). Taking into account the varied user base consisting of both scientific users and the general public, Andrysco transformed the 2D data points originating from the advected RAMS data into 3D visualizations through a newly developed computer application. The application developed chiefly by Andrysco, seen in fig. 2.1, presents users with the ability to create visualizations unique to their purpose by way of single location, time and viewing angle selection. However Andrysco notes significant limitations in the application's ability to run at interactive rates because of limiting computer architecture bottlenecks such as the high memory requirements of the large dataset and then subsequent slow memory and disk access rates. Andrysco states "the next step is to explore methods to facilitate interactivity" (p. 3) and this thesis presents a method of achieving this goal.

The most significant visualization present in the lagacy software for the purposes of continuing research into interactivity is the generation of an Iso-surface of the advected CO_2 in the atmosphere because of the increased insight into CO_2 transport and weather phenomena that this iso-surface shows. Andrysco's visualizations bring many of Gurney's (Gurney, et al., 2009; Gurney et al., 2005) conclusions to light, such as the strong summertime transport of Southern Californian air across the Pacific Ocean the Gulf of Mexico.

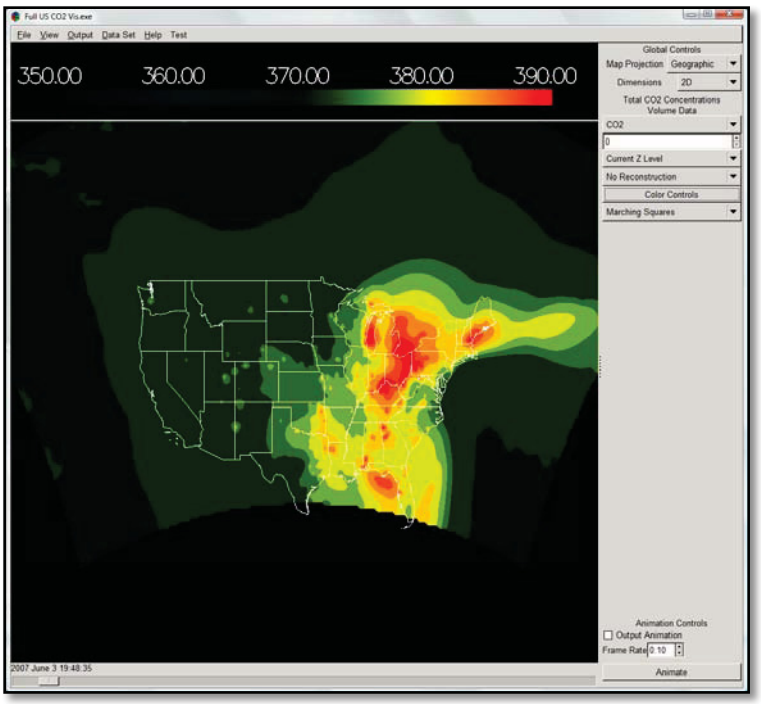

Figure .2.1 Legacy C02 Viz.

Andrysco's application allows the generation of offline, non-interactive content, one example being a highly watched video seen on YouTube (*'Revolutionary' CO2 maps zoom in on greenhouse gas sources*, 2008). Andrysco's application generates many images, with rendering times often taking 10 to 20 seconds per frame that may be linked to form a video. While this generated multimedia is a strong tool for the user-base and scientific community, the experience offered is far from the desired real-time exploration of this dataset. The videos generated are limited in information scope, lacking in visual fidelity, and take large amounts of time and processing power to produce. The geometric data produced by the legacy software is discarded after use, and any subsequent viewing of the data from a different viewing perspective requires re-generation of the visual representation. Thus a new real-time rendering system will discard re-generation of the legacy software's data by storing and utilizing the data from disk. The entire scheme of data flow from raw data through the various

systems to the new system is explained in fig. 2.2. The final new element, colored green, is the culmination of this research's contributions, the new C02 Viz and resides on the top of the data flow.

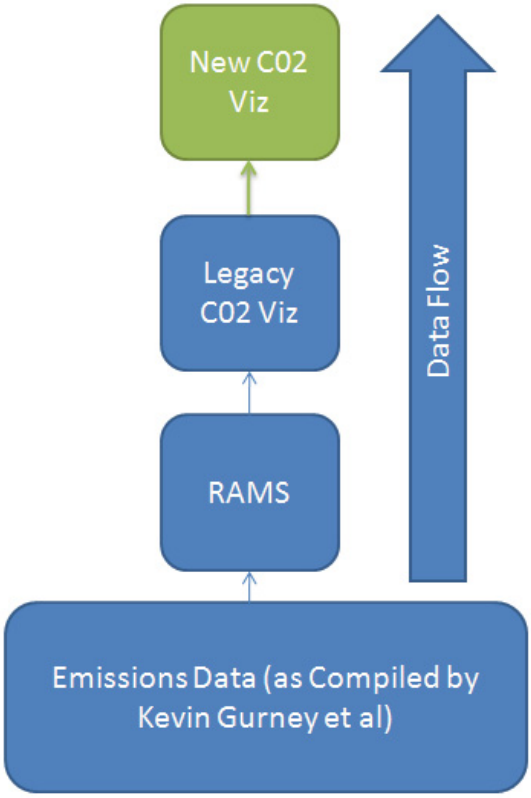

Figure 2.2 The Rendering system in context with previous systems.

2.3. User base

The results of the legacy software (Andrysco, et al., 2009) have already been circulated in the scientific community by releasing the visualizations on the internet via streaming video and Google Earth (Purdue University News, 2009). Distribution of the video across the internet and the incredible number of views indicates the wide interested user base of the data. In order to best service the audience of this data, both scientific professionals and the general user must be taken into account with the presentation and interface design.

Hadwiger's research in volumetric rendering (Hadwiger, Ljung, Salama, & Ropinski, 2008), compared with Marsalek's research in the same field (Marsalek, Hauber, & Slusallek, 2008) noted that the advanced techniques required to produce real-time performance demand a significant level of consumer hardware. At the time of writing, this level of hardware translated into consumer terms means a GPU equivalent to an *Nvidia* GPU of the 8xxx series or higher. While this hardware is easily available and priced at fewer than 200 dollars at the time of writing, it is not in use by everyone outside the scientific community. Because considering penetration of such consumer hardware is out of the scope of this research, an additional research effort is required to effectively distribute the new visualization to general consumers. Such an effort might continue to use video presentations and internet media. Beyond the technology required, additional consideration must be given in the application design and usability. Noting the goals of the user base, to disseminate knowledge from the dataset, the application must expose this data at the correct level.

2.4. Iso-surfaces

The first exploration into interactivity would be to re-use the once discarded geometric data generated from C02 Viz (Andrysco, et al., 2009). Because the geometric data present in a single frame is of a reasonable amount for real time rendering the entire set is not immediately suitable for real time. The data taken from C02 Viz must therefore be captured and stored using geometric compressing data structures. First published by Lorensen in 1987 (Lorensen, 2006; Lorensen & Cline, 1987) a method for creating hardware-renderable geometric data from volumetric datasets is introduced and called marching cubes. The algorithm, now out of its copyrighted period is free for use and Andrysco implemented the marching cubes algorithm for creation of the geometric representation of the advected Vulcan data. In order to compress the data from Lorensen’s algorithm, which voxelizes the advected dataset before

producing a surface polygonal patch, the voxelization parameters must be reconstructed and then the geometry may be recorded. Geometric correspondence produced by the marching cubes algorithm from frame to frame is unpredictable. Thus the methods of compressing the actual marching cube produced geometry (the “triangles”) and compression of the parametric data (the “indexed” data) are contrasted.

Knowing that the geometric data from Andrysco’s application is of a four-dimensional structure, covering volumetric space over time, Neophytou (Neophytou & Mueller, 2002) describes efficient methods for rendering such volumetric data. Utilizing point splatting 3D-data points over time, the GPU creates view dependant geometric data. Neophytou indicated that instead of storing geometry, a parametric point cloud dataset is stored; an efficient point based rendering technique can then be used to obtain an approximated 20% performance increase over polygonal (traditional) geometric renderer. Not only is the rendering speed increased, the parametric dataset is often a reduced dimensionality than geometry, and thus takes up far less data storage space. Neophtou’s point rendering method is effective, but the final images it produces do not have as high visual quality as desired by C02 Viz, and as such are not a candidate for the Vulcan CO2 Viz suite. However utilizing Neophytou’s ideas on data structures, better rendering solutions are discussed later.

2.5. Methods for Optimization

Correa (Correa, Klosowski, Morris, & Jackmann, 2007) details an entire visualization framework for distributed computing dissemination of large datasets in real time. Correa’s suit, while detailing important optimizations for rendering, applies them to a different subset of volumetric rendering than that within the research question of this thesis, that of rendering a large amount of *geometrical data* in a distributed environment. Andrysco’s (Andrysco et al., 2009) application produces a “reasonable” amount of geometry for a given time step in the data,

and thus the raw amount of geometrical data generated does not overwhelm a single CPU or GPU and thus require a distributed architecture. However many of the optimization techniques used on Correa's application suite can enhance the use of the data structures present in this thesis. Correa lists "spatialization, simplification, view-frustrum culling, occlusion culling, multithreading and prefetching" (p. 12) as being implemented within the suite. Correa explains each listed technique, and spatialization, simplification, multithreading and prefetching are applicable for use with the Vulcan Data compression and rendering.

2.5.1. Data Compression

Noting a significant limiting factor in the performance of the previous visualization was the "memory access time" (p. 11) (Andrysco, et al., 2009) and thus to overcome that performance bottleneck the amount memory used by the data must be reduced without compromising the information contained therein. A field of research that addresses this problem is data compression.

Fout (Fout, Ma, & Ahrens, 2005) postulates that while the analyst's ability to create large scale numeric simulations increases, the data being produced is often occupying "hundreds of gigabytes and often several terabytes" (p. 1). This severely degrades the performance of computer application attempting to use this data, as memory sizes are often orders of magnitude less than this size and data read time from memory is prohibitively slow. This claim is reinforced by Andrysco, as file sizes of the Vulcan volumetric data, as read from the Vulcan public website (Department of Earth and Atmospheric Sciences, 2007) average file-size is approximately 5 gigabytes. Fout exploits correlation in volumetric data to produce strong compression in time-variant data. The data structures used for the Vulcan project must therefore establish frame to frame correspondence.

Combining time-variant volumetric compression with traditional geometric compression LOD models would require significant pre-processing of the large

volume datasets. The computation time is a variable to minimize when considering increasing application performance. Khodakovsky's research into (Khodakovsky, Schröder, & Sweldens, 2000) progressive geometric compressions affords the final application high run-time efficiency and flexibility.

Care must be taken in compression however, as Neophytou (Neophytou & Mueller, 2002) adds in final remarks to his research that a significant drawback of improving compression is the extra work that is required at run-time to decompress the data.

2.5.2. Level of Detail

Song (Song, Bai, & Wang, 2006) describes a method of geometrical metamorphosis used for compression of an object between levels of detail (stages of change, morph targets, within the transformation) from an acquired dataset. Song's method is indicative of a class of methods that change a polygonal representation of a surface (such as that resultant from Andrysco's iso-surface generation) to an implicit surface representation, in this case, a Bezier-spline patch surface. With claimed data reduction of 90%, it would seem like an ideal technique for application with the Vulcan data. Song notes that finding geometrical correspondence, the process of identifying key characteristics of an object that give meaning to the surface between different representations, is very difficult on an arbitrary geometrical object. The geometrical data produced by the marching cubes algorithm used by Andrysco's (Andrysco et al., 2009) application is indeed an arbitrary object with no apparent "features" and varies greatly depending on the parameters of time and CO_2 atmospheric concentration. Song defines a unique method for finding correspondence in an arbitrary shape with no human input required, however the shapes and surfaces used an input into Song's method are 3D scans of real objects, which can be approximated by a continuous surface. As such the Vulcan data would require up to hundreds of such surface approximations. This exemplifies a common problem with

implementing an implicit surface reconstruction; the Vulcan data is too chaotic for "regular" or "orderly" techniques, as such techniques would fail to wrap the complete dataset. Thus, while Song notes that the results are "aesthetically pleasing", the expected error rate in use of Song's method or techniques of the same class on the chaotic Vulcan geometry would be too high.

To improve the performance of the CO2 Viz system to real-time standards (that of reproducing images at speeds greater than 30hz) discrete LOD methods such as those discussed by Luebke (Luebke, Reddy, Cohen, Varshney, Watson, Huebner, 2002) are used. Luebke notes that LOD details are done as a pre-process and that the users viewpoint into the data cannot be predicted, thus the Vulcan data must be uniformly reduced across the sample space (a desired parts-per-million). Luebke notes that this approach is the most amicable to modern graphics hardware as the rendering process and data simplification are decoupled, and the decoupled simplification process can then be performed in as much time as desired.

2.6. Terrain Visualization

The new Vulcan visualization requires not only direct rendering of the CO_2 data, but additionally the images that place the Vulcan data in context, satellite imagery of the United States. Andrysco (Andrysco et al., 2009) utilized Geographical Information System (GIS) data in his visualizations and in turn the same GIS data is of use to the new visualization. In Polack's book (Polack, 2002) he outlines several methods for real-time large scale terrain visualization, the most robust and effective of these methods is an improvement on an older algorithm entitled "Real-Time Optimally Adapting Mesh" (ROAM). ROAM 2.0, the improved version of the original ROAM algorithm was updated by Seamus McNally and is covered in research by Turner (Turner, 2000). McNally's improvements to the ROAM algorithm involve eliminating data redundancy by establishing coherence between render frames and only applying slight

modifications to the ROAM structures at any given time step. The effect of these changes reducing the need to recreate the data structure every frame significantly boosts the performance of the algorithm, to the point at which run-time use of the algorithm is almost unnoticeable on modern computing systems. ROAM has been used successfully by Hwa (Hwa, 2005) in his research regarding planetary terrain geometry and texturing, proving the algorithm is both usable and scalable for larger terrain sets like those used in Andrysco's application.

2.7. Rendering System

Two competing rendering system types are contrasted: Direct Volume Raycasting and Boundary Surface Representation by either points or polygons (triangles, quadrilaterals).

Using the compressed volumetric data across all levels of detail the advantages of a point based rendering system as described by Neophytou (Neophytou & Mueller, 2002). Neophytou's rendering system, however, is not interactive, as it does not use any GPU acceleration. Any rendering system wishing to be interactive would require GPU implementation given such a large dataset. Neophytou's point based rendering system can be contrasted against other leading volume rendering techniques as Hadwiger (Hadwiger et al., 2008) states that volume ray-casting is the "state of the art technique for interactive volume rendering." (p. 2). Hadwiger describes GPU implemented rendering techniques for volume ray casting that provide not only convincing images, but provide a basis for advanced illumination techniques such as ambient occlusion and soft shadowing. In Hadwigers 2008 course notes, he states that experts agree that direct volume ray-tracing (DVR) can be superior to polygonal boundary representation as any DVR algorithm does not have to calculate a surface representation. The results of Hadwiger prove to be superior to that of both polygonal surface representation and point based rendering systems yet

contain a much higher memory footprint of operation as the source data is required for rendering. This claim is backed by Marsalek's (Marsalek et al., 2008) research into high speed ray-casting using Nvidia's CUDA programming language. Marsalek's results show that using the latest graphics hardware, direct ray casting can be implemented with performance results equal to that of previous techniques of volumetric rendering but with greater visual fidelity. Storing all the data and performing the ray casting algorithm in parallel directly on GPU affords this increased performance. Volumetric ray-tracing as Hadwiger (Hadwiger et al., 2008) explains, contains two key elements: the volumetric data and the transfer function. The transfer function can be modified by the user in real time to create a unique visualization that is visually stunning and affords the user a far greater perception of the data than a surface polygonal representation ever could.

Yet the images that DVR techniques create are at the price of memory. As there is significant preprocessing (done at application run-time) and these techniques still operate directly into the source data-set. Comparatively; boundary surface rendering systems such as QSplat and traditional triangular based meshes require the least amount of memory per frame. As described in the research by Rusinkiewicz and Levoy (Rusinkiewicz, S. and Levoy, M. 2000) documenting *QSplat* (2000) the authors describe how with larger data, like that of the Vulcan dataset, the size per-surface can be greatly reduced with triangle based representation, and even further by point based representation and rendering. By ignoring connectivity information the data is reduced, theoretically, by 33%, and as described by Rusinkiewicz and Levoy by still retaining normal data for the points, the splats can be rotated to maintain boundary lines, and filling algorithms can be used to ensure close to full coverage in a particular image. The run-time rendering complexity of the system is explored and the results suggest suitability for application with the Vulcan system. However QSplat techniques as described by Rusinkiewicz are not implemented for time varying meshes/surfaces like the Vulcan data set and additional novel techniques will

need to be introduced to cover the full functionality required for real-time rendering of the full Vulcan dataset.

2.8. Lighting and Shading Techniques

Beyond the initial rendering system, there exists a class of algorithms dedicated to improving the visual quality of a single image, referred to in this thesis, as lighting and shading techniques. The implementation of key shading techniques will greatly enhance the final image, and thus the speed of information dissemination by the target user-base of the visualization. However additional processing time must be devoted to the techniques, and the additional processing time required by each new technique must be evaluated against the potential gain in quality.

Neophytou's (Neophytou & Mueller, 2002) research into point based rendering states that an important effect for blending between levels of detail of animated data is motion blur, blurring the transitional effects. By blurring the data with motion blur, each frame further emphasizes the time dimension of the data. An additional advantage of motion blur as Potmesil (Potmesil & Chakravarty, 1983) describes is that it removes the appearance of aliasing in animated transitions for any given object. Because the Vulcan data will operate on several levels of detail, and is inherently "data in motion", that is that the CO_2 is volumetric particles in motion in the atmosphere, motion blur is an important effect for inclusion. Potmesil describes a new method of motion blur synthesis by way of a camera synthesis model, using cascaded optical transfer function that can express time dependant changes of an object position and direction. Both position and direction play important roles in the meaning of the Vulcan advected data, and emphasizing these effects will enhance the experience.

To further help the user distinguish the nuances of the surfaces generated, there exist several advanced illumination algorithms that approximate the global illumination algorithms present in high fidelity low-performance offline rendering systems. Ambient occlusion is one such illumination algorithm that estimates the global visibility across the rendered surface. Hadwiger (Hadwiger et al., 2008) details a method for computing ambient occlusion with direct volume raytracing that includes semi-transparent samples and translucency. Hadwiger's methods simulate key visual features such as "direct lighting, shadows and interreflections" (p. 79). Hadwiger's algorithms are suitable for implementation with a direct volume rendering system and not for a boundary surface system.

2.9. Conclusions

The previous Vulcan visualization paved the way for this thesis, however many new steps and techniques will be applied to achieve interactivity. Utilizing methods used by Andrysco in data generation, a new compressed and optimized series of levels of detail of the Vulcan dataset must be created used principles found in Luebke (Luebke et al., 2002) and Neophytou (Neophytou & Mueller, 2002). Principles such as data simplification, compression and representation.

2.10. Chapter Summary

This chapter summarized existing literature on the subject of volumetric visualizations and the methods used by others for improving such visualizations for real time experience of large data-sets. The main rendering methods of rendering systems were compared and contrasted, as well as providing a brief summary of the suitability of other key components for a complete visualization of the Vulcan data set, such as terrain visualization. Additionally the chapter covered previous work in the Vulcan project, and how this provides a motivation for new research in the volumetric visualization research field.

CHAPTER 3. METHODOLOGY

This chapter will cover the implementation of the Vulcan real-time rendering system and also research framework, sample set and testing methodology used in this thesis.

3.1. New C02 Viz Rendering System Introduction

The combination of compression, rendering system data structures, layer interpolation, terrain rendering and lighting and shading techniques that are tested in this research creates a complete new real time rendering system. The data flow diagram of this system that relates these research areas is shown in fig. 3.1.

The new C02 Viz is a C++ application suite that is capable of testing rendering and compression algorithms and data structures for their suitability in rendering the Vulcan Data in real-time. The application suite consists of two independent C++ applications; a compression pre-process application and a real-time loading and rendering application.

3.2. *Triconverter*, A simplification and data sorting application

The first application in the C02 Viz suite is named Triconverter and is chiefly responsible for transforming the data from the format used in previous visualization attempts and compressing the data accordingly into a format that may be quickly read by the real-time loading and rendering application.

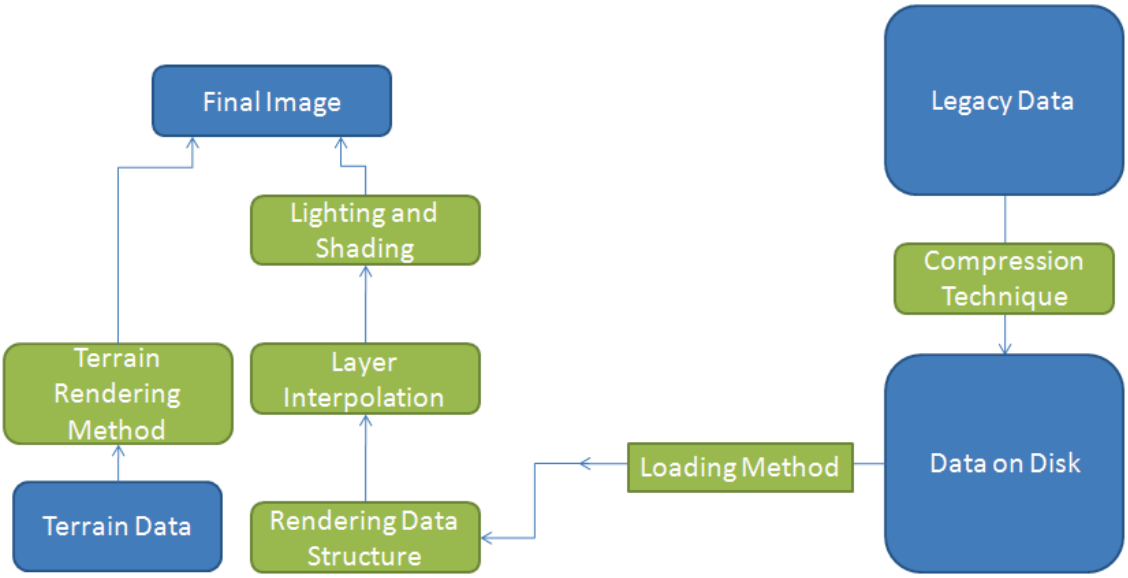

Figure 3.1 General Data Flow Diagram for a real-time C02 Viz.

Triconverter was developed iteratively, initially using a indexing method to reduce the data size, and outputting data in ASCII format for rendering testing. This method proved useful for getting initial prototypes of the rendering system running, but did not provide adequate compression ratios. Initial compression ratios were in the magnitude of only 10-25% with indexing alone. Once the data was converted into binary format (that is data unreadable to a human being but readable quickly in bulk into computer memory) and simplification algorithms employed the compression ratio became much more desirable, the results of this application are outlined in section 4.2.

The application is without a GUI and is run in the command line for increased speed and easier batching. The program first asks for a host directory, and then begins to scan this directory for all suitable geometric data files written by the legacy software.

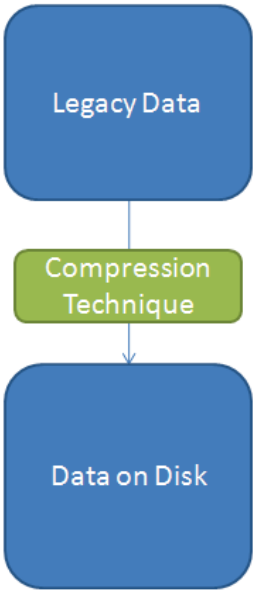

Figure 3.2 Triconverter Behavior.

To transform the data, as indicated by fig. 3.2 *Triconverter* employs a dynamic grid across the data volumetric space and then indexes this grid as a list of geometric primitives are read in from the input data. As a primitive is read, its constituent vertices are checked against the database of existing vertices in the grid, with duplicates being removed and final indices into the database stored. The simplification algorithm in pseudo code is shown in table 3.1.

To test the effectiveness of this algorithm, *Triconverter* would be executed twice on the data provided by the legacy software at varying PPM levels. *Triconverter* would be an invisible application to the final user of the software suite, and is only run a few times to create the compressed data sets used by the rendering application at a later time. *Triconverter* would only need to be run again once new Vulcan data sets are collected.

Table 3.1
Simplification Algorithm

Step	Description
1	Take Point p_1 from Data
2	Check Point p_1 against every other point p_x in the dictionary test using simplification threshold
3	If p_1 is unique, that is, is it greater than length dl from all p_x add it to dictionary : else discard p_1
4	If there are still uncategorized points, go to step 1 else finish

The algorithm is implemented using the *search* functions within the C++ standard templates library (STL) which allows easy change of the simplification tolerance, and maximized execution speed. In addition to the algorithm outlined in table 3.1, there are several other switches which may be toggled to change the final outcome (often for testing purposes). The switchable flags include; removal of degenerate triangles; process points only, points and normals or full polygonal representation. The output of the software is then up to 3 binary files containing geometric information and a single ASCII header file used to index the geometric data. The file types are summarized in table 3.2. The specific testing variables are discussed in section 3.4, 3.5 and 3.6.

3.3. Loading and Rendering Application

The final run-time, and program presented to general users is an application in the new C02 Viz suite that is responsible for loading the data generated by *Triconverter* and rendering it in real-time. This behavior is summarized in fig. 3.3.

Table 3.2
Triconverter File Types

Extension	Description
Head.ivh	ASCII header file containing index locations for each frame present in the data series
I.bin	Binary format containing polygon index information for all frames in a series
V.bin	Binary format containing vertex information for all frames in a series
N.bin	Binary format containing normal information for all frames in a series

This application would be chief entry point of the "users" of the application suite, and be the "face" of the Vulcan dataset as it has a graphical user interface (GUI) more suitable for user interaction than a rapid terminal.

3.3.1.1. Loading

The rendering application covers the bulk of the behaviors outlined in fig. 3.3. It is therefore most important that this application has satisfactory run-time performance that allows users to manipulate their view into the Vulcan dataset minimizing stuttering or delays. The results of the applications performance and the techniques used are covered in sections of 4.3, 4.4 and 4.5 of this chapter.

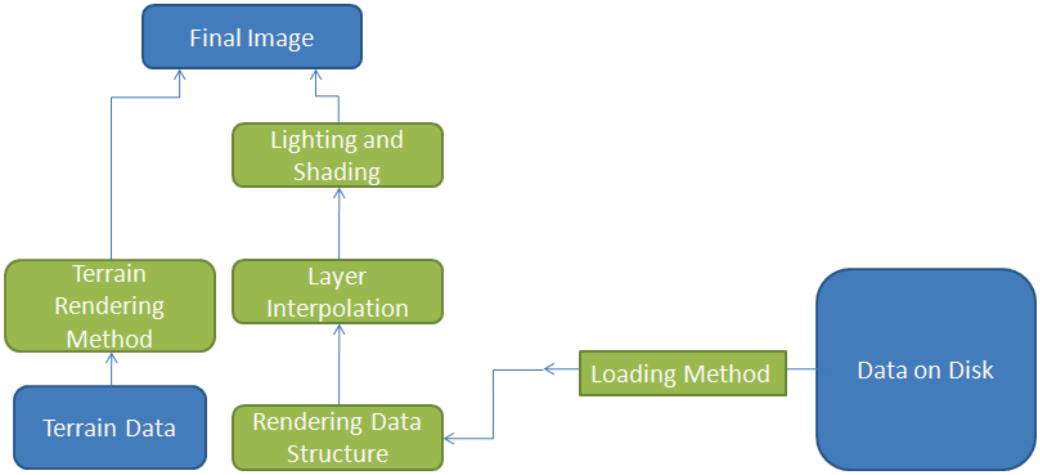

Figure 3.3 Real-Time Application Behavior.

Using the reverse operations of binary reading, the run-time application caches the entire header file into RAM, which is only a small file in the order of 1-2KB. Using the file index, the run-time application can load a set number of frames at once into a rendering container . The specific parameters of loading, frame number, data type and operating thread can be controlled for testing. All of this CPU code handling file operations is implemented in the C++.Net Framework, using managed C++ to control the GUI and event framework (such as buttons and folder browsers), while retaining elements of unmanaged C++ for lower level file operations and graphics API communication.

3.3.1.2. Lighting and Shading

To visualize the data once it is loaded into RAM, the data may be rendered using the GPU and the OpenGL Shading Language (GLSL) to control the final vertex and pixel positions. Depending on which geometric representation the data is stored in (point, point/normal or polygon) the rendering

method employs a different shader set. The top objectives of the shaders are as follows:

- Maximize data visibility with information available
- Maximize rendering speed
- Maximize data coverage when working with reduced data representations

For point based representation, the vertex shader controls the size of the point sprite based on the distance from the camera. As the point is found to be closer to the virtual camera the point size increases, thus pixel holes are mostly filled in this way.

For polygonal representation, the vertex shader does not use point sprites, but instead performs the standard transformations.

For all geometric representations, the fragment shader applies a ramp texture to the final result, varying the color by height, so as to maximize the data perception.

3.3.1.3. Layer Interpolation

Once the data is loaded in memory and ready to be rendered in a continuous blend from known data points to produce a full spectrum of information approximating the source dataset. This feature is a novel method that in psedo code is as follows in table 3.3 - this method is executed at tun time each frame.

This method does not require a particular input representation, but is sensitive to holes in the textures generated in step 1 - therefore is best used with Triangle based meshes or dense point clouds.

Table 3.3
Interpolation Algorithms

Step	Description
1	Render target objects (two known sufaces) to floating-point textures, storing normal and position information.
2	Render a full-screen quad to texture, sampling the four textures generated in step 1 to interpolate between the two points in space defined by a unique texture coordinate. This is done by sampling the two positions and normals, and using a bezier cubic equation, and a desired interpolation percentage.
3	Render to the window, sampling the texture generated in step two to find new vertex positions.
4	Transform and shade these new points using the point based rendering system described earlier.

The algorithm in C02 Viz is implemented in GLSL in order to remain platform independant and expose the highest possible API possible on a given graphics card. The algorithm requires 3 framebuffer OpenGL objects (alongside the standard window framebuffer) and the vertex texture fetch extension from the graphics card driver.

3.4. Research Framework

This thesis presents a quantitative study on the most appropriate methods for real-time visualization of the atmospheric Vulcan data. The research follows an experimental model that will manipulate several key independent variables:

- Method of geometric data compression

- Method of data representation
- Method of run-time data buffering
- Use of lighting and shading techniques
- Use of layer interpolation techniques

The effect of these variables will be measured across four performance areas:

- Data reduction, measured in percentage reduction of both file size on disk, and size in memory.
- Run-time loading time (primary interest), measured in milliseconds per data frame.
- Rendering complexity (primary interest), measured in the milliseconds per image frame generation.
- Pre-Processing Time, measured in milliseconds per data frame.

The research will focus on testing several null form hypotheses, identifying each correlation as specific to the Vulcan data, if H_{ax} is proven.

H_{o1} There is no effect of data compression on run-time rendering complexity and loading time.

H_{a1} There is an (positive) effect of data compression on run-time rendering complexity and loading time.

H_{o2} There is no effect of the method of data representation on the run-time rendering complexity, loading time, and data reduction.

H_{a2} There is an (positive) effect of data representation on run-time rendering complexity and loading time.

H_{o3} There is no effect of loading method on run-time rendering complexity and loading time.

H_{a3} There is a (positive) effect of loading method on the run-time rendering complexity, loading time, and data reduction.

3.5. Sample set

The research will use Regional Atmospheric Modeling System (RAMS) advected datasets. At the time of writing the most recent set available is the 2007 power plant-residential air data collection for the atmospheric carbon dioxide across the continental United States of America. The dataset was originally used in the legacy software (Andrysco et al., 2009) and is used with permission from the Vulcan project for the new research.

3.6. Testing Methodology

The experimental design involves implementing a visualization application capable of determining the effect of the independent variables (method choice/inclusion) on the dependant variables (performance). This implementation is a software suite called "CO2 Viz". The tests will be conducted in a 'laboratory' style test computer system, utilizing several methods of data collection.

The implementation of the rendering system will consist of a *Microsoft Windows* application using the following significant libraries:

- *C++.Net* language for CPU code
- *OpenGL* library for GPU code

The dependant variables will be measured using well accepted scientific computer clock measurement. Using the *C++* library computer clock

measurement to calculate processing and rendering time, the rendering process was measured at all stages of execution in isolation, so as to minimize the effect of measurement on the results. Additionally, run-time graphical performance is measured both by computer clock speeds and high performance graphics benchmarking software. The program that is used to benchmark rendering is *beepa FRAPS* (Fraps 2010). The benchmarking software is run as an additional process to the C02 Viz software running in a separate thread. A detailed spreadsheet is created on the rendering benchmark and is analyzed after program execution.

The laboratory test computer is operated under the following specifications and conditions:

- Core i7 architecture CPU 920
- *NVIDIA* GeForce GTX275 896MB GPU
- 6GB System RAM DDR3-1066
- *NVIDIA* Graphics Drivers (185.5) (latest as of writing)
- Defragmented 7200RPM Hard-drive
- Fresh start-up Windows 7 64bit operating system with bare-bones process set in operation.

The test will be performed in the following order:

- Compression
    - This suite of tests will determine the effects of the independent compression variables (simplification tolerance, representation format) on the dependant variable of size on disk. Secondary dependant variables include compression time.
- Loading

    - This suite of tests will determine the effects of the independent loading variables (threaded loading, burst loading) on the dependant variable of loading time.

- Rendering
    - This final suite of tests will determine the effects of the independent rendering variables (representation format, simplification tolerance, burst loading) on the dependant variables of rendering time.

3.7. Data Sources

From the previous testing methodology, several data sources are generated:

Primary:

- System Clock Measurements (quantitative tabulated numerical data)
- Rendering Performance Measurements (quantitative tabulated numeral data)

3.8. Data Analysis

Three identical pre-determined sequence testing routines are completed independently from each other for each hypothesis (section 3.1)

- Computer Start-up
- C02 Viz Start-up
- Initial loading of Vulcan Data-Set

Then the quantitative measurement takes place for each rendering component.

The quantitative data is statistically analyzed for establishing the correlation between the key independent and dependant variables.

3.9. Chapter Summary

This chapter covered the key variables for scientific investigation and the testing conditions for which the implementation application, C02 Viz, will be tested under. Additionally, the data collected from the three identical testing sessions was outlined in both type and method of analysis.

CHAPTER 4. RESULTS

This chapter will cover the results generated by applying the testing methodology detailed in chapter 3 of this thesis. The chapter begins by delving into each component outlined in the process overview. Initially the compression ratio results are detailed, followed by results from method of geometric data compression and the amortized loading times and then finally ending with run time rendering analysis of the loading, lighting, shading and interpolation techniques.

4.1. Compression Results Related to Size on Disk

The first results are obtained from passing the full collection geometric data, outputted from the legacy software into *Triconverter*, and measuring both the size on disk per data frame, and compression and simplification ratios. The results, utilizing the system clock to measure time differences, and the operating system to measure data sizes are detailed in tables 4.1 ,4.2 and 4.3 and fig. 4.5, 4.6 and 4.7.

Table 4.1 shows the results of the compression process, which is the combination of data representation and simplification. The simplification process implemented removes degenerate data as well as reducing data complexity.

Table 4.1
Simplification File Size Reductions

File	File Size Pre Simplification (MB / frame)	File Size Post Simplification (MB / frame)	Amortized Compression Ratio	No Of Frames
PPM 1 Set (380.5 PPM)	7.033	0.2400	96.6%	3679
PPM 2 Set (381.5 PPM)	5.919	0.2044	96.6%	2229
PPM 3 Set (382.5PPM)	4.072	0.1420	96.5%	1481

The results of compression show a very satisfactory compression ratio, averaging 96.6% compression from the original data when converted down to a point representation. The level of data compression per frame can be explored across the whole data series of a particular PPM level to show that it is indeed constant. Seen in fig. 1 and fig. 2 the compression ratio is uniform across the whole series, varying by only .10%.

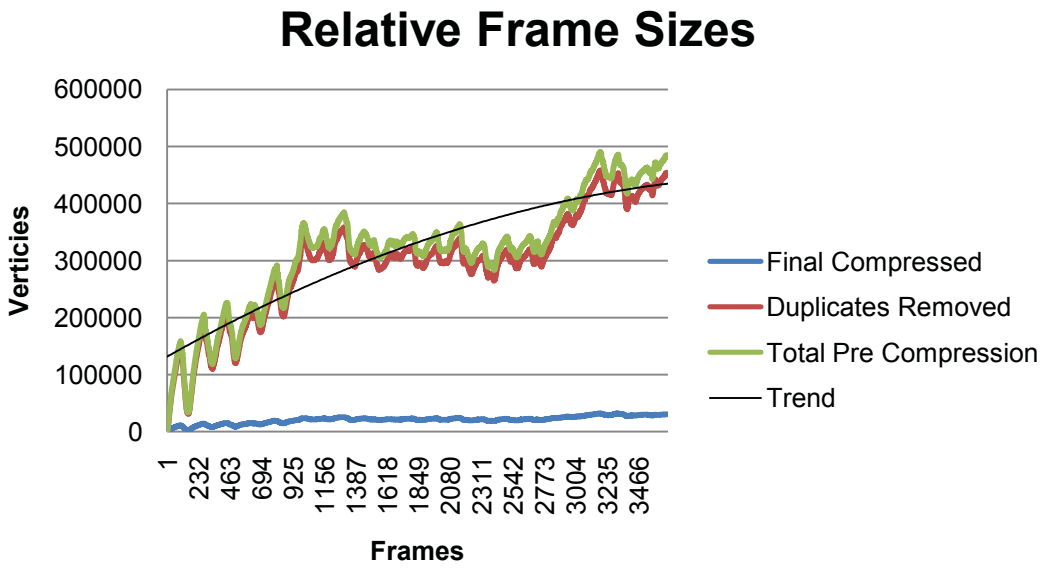

Figure 4.1 Graph of Relative Frame Sizes and Trend @ 380.5 PPM.

The effect of varying simplification tolerance is shown in table 4.2. The sample set used in this test was a representative 100 frames taken from the Vulcan dataset. The 100 samples were tested three times then averaged. The zero tolerance sample with no duplicates removed is provided as a control.

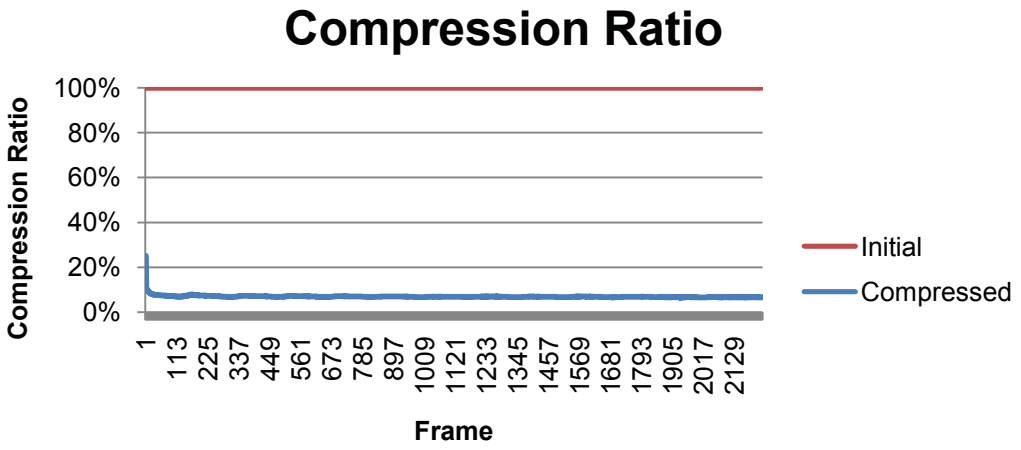

Figure 4.2 Compression Ratio Graph Across Series @ 381.5 PPM.

From table 4.2 it can be seen that the a significant factor contributing to 94% of the data size reduction is the removal of duplicate entries generated from the marching cubes algorithm (shown in the rightmost column). Further reduction of the data size is observed as the tolerance is increased from 0 to 0.035, but this only removes at most an additional 1% file size reduction. This indicates the structure of the source dataset contains mostly duplicate entries and comparatively little useful information, and why there is a significant bottleneck in previous attempts, a great amount of data redundancy.

In addition to removing duplicates and simplification, the representation reduction, shown in table 4.3, shows test cases of varying data size reduction based on geometric representation. These test cases are run at 0.001 simplification tolerance with degenerate triangle removal.

It is clear that the number of duplicate entries in the vertex field means that the number of primitives (and possible degenerate primitives) greatly increases the file size on disk. A further test can then be performed to determine if degenerate triangles will reduce the file size.

Table 4.2
Simplification Tolerance Reductions

Tolerance	Processing Time (Average) (s/frame)	File size (Average) (MB /frame)	Vertex Count (Average /frame)	Duplicates Removed (Average /frame)
0 (No Duplicates)	34.248 (100%)	2.13 (100%)	184348 (100%)	0
0	1.244 (100%)	0.1250 (100%)	10896 (100%)	139571
0.0000001	1.301 (104.6%)	0.1250 (5.87%)	10896 (5.91%)	139571
0.0001	1.310 (105.3%)	0.1250 (5.86%)	10893 (5.91%)	139572
0.001	1.300 (104.5%)	0.1220 (5.72%)	10616 (5.76%)	139646
0.002	1.286 (103.4%)	0.1190 (5.59%)	10339 (5.60%)	139732
0.003	1.282 (103.0%)	0.1160 (5.44%)	10068 (5.46%)	139815
0.01	1.138 (91.5%)	0.0977 (4.59%)	8457 (4.59%)	140397
0.015	1.164 (93.6%)	0.0864 (4.05%)	7484 (4.06%)	140980
0.025	1.067 (85.8%)	0.0669 (3.14%)	5790 (3.14%)	142325
0.035	1.002 (80.6%)	0.0507 (2.38%)	4393 (2.38%)	143501

Table 4.3
Representation File Size Reductions

File	Compressed Full Data (MB)	Triangles (MB)	Point-Normal (MB)	Point (MB)
PPM 1 Set (380PPM)	460	460 (100%)	236 (48.7%)	118 (24.4%)

A test was performed with a representative 100 frame random sample taken from the Vulcan data to determine the amount of degenerate triangles present in the Vulcan data, the results are summarized in table 4.4. With only a 8.27% reduction on top of the normal compression methods used earlier, degenerate information is proven to not a significant portion of the Vulcan data, however since it only adds on average a 0.25s pre-processing time increase each frame, it can be a matter of preference to include this technique or not.

Table 4.4
Degenerate Triangle Removal

Technique	Mean time (s)	Total Degenerates	Total Elements	Reduction
Removal	1.552s	55518 removed	671111	8.27%
No removal (control)	1.307s	0 removed	689617	0.00%

4.2. Compression Results Related to Run-time Loading

After compression, the data is loaded on-demand at run-time into RAM in sections appropriate for a 32bit windows process which is limited to using approx 1.2GB of RAM. The run time rendering application was used to test the effects of compression on the run time results. By using the system clock to measure the difference in time between beginning loading and finishing loading a sample the loading time can be measured closely. By recording the number of bytes read in

a single data transfer session the data transfer rates, measured in MB/s can be calculated using the formula shown on the next page.

$$Transfer\ Rate = \frac{Bytes\ Transferred}{Time\ Taken}$$

There are two types of processes measured; both in-thread and multithreaded loading. In in-thread loading, the data is loaded as the rendering thread detects it reaches the final frame of the current set in memory. Conversely in multithread loading, the data is triggered to load in a separate process on the CPU, which is often advantageous on multi-core CPU like the test machine. However this possible performance increase is lessened by the RAM bottleneck. In the multithreaded environment all processes will be reading and writing to RAM simultaneously, which inevitably causes read/write locks on the data. Slowing both threads below optimal results.

As tables 4.5 and 4.6 show, both in-thread and multithreaded processes experience similar memory bandwidth from disk to data structure transfer rates. The key statistic of the disk loading is the "Mean Load Time". This value should ideally be below that of what it would take the GPU to render one frame, so that no visible interruption is detected - this means below 0.016s for a 60fps experience, or 0.033s for a 30fps experience. From this we can determine that loading either in-thread or multi-threaded are acceptable loading methods, however the frame limits per load must be below 76. While mean transfer rates across the entire execution are a useful statistic, the peak times are a key consideration when controlling the entire experience.

Note how in Table 4.6 there are no values listed for the test performed at 16 frame sets - this is because the loads were completed so fast, the computer did not have sufficient accuracy to measure it, that is that the event occurred in less than 0.001s.

Table 4.5
Multithreaded Bulk Loading Results

Frames (at once)	Mean Load Time (s)	Mean Delete Time (s)	Verts (average)	Mean Data Transfer (MB/s)
16	0.0054	0.0015	794182	2944
26	0.0094	0.0023	1639032	3318
76	0.0195	0.0024	3976904	2280
101	0.0358	0.0076	6254136	3344
201	0.2830	0.0160	12179108	2963

The loading times across an execution follow a logarithmic pattern, as shown by the trend line in fig. 3. This can be analyzed against hard drive execution patterns knowing that hard drives have 'spin up' times and cache build up. The sustained transfer rate that could be expected for real world use is on the tail of the trend, showing an evening-out to 3700Mb/s. Knowing that a vertex has 3 x 4 bytes (the size of a floating point number) we can conclude that the hard drive is capable of supporting 308,000 vertices transferred per second.

Table 4.6
In thread Bulk Loading Results

Frames (at once)	Mean Load Time (s)	Mean Delete Time (s)	Verts (average)	Mean Data Transfer (MB/s)
16	-	-	861642	-
26	0.0048	0.0016	1577606	2961
76	0.0145	0.0045	4784983	2948
101	0.0361	0.0084	6335434	3314
201	0.0426	0.0143	10836738	2179

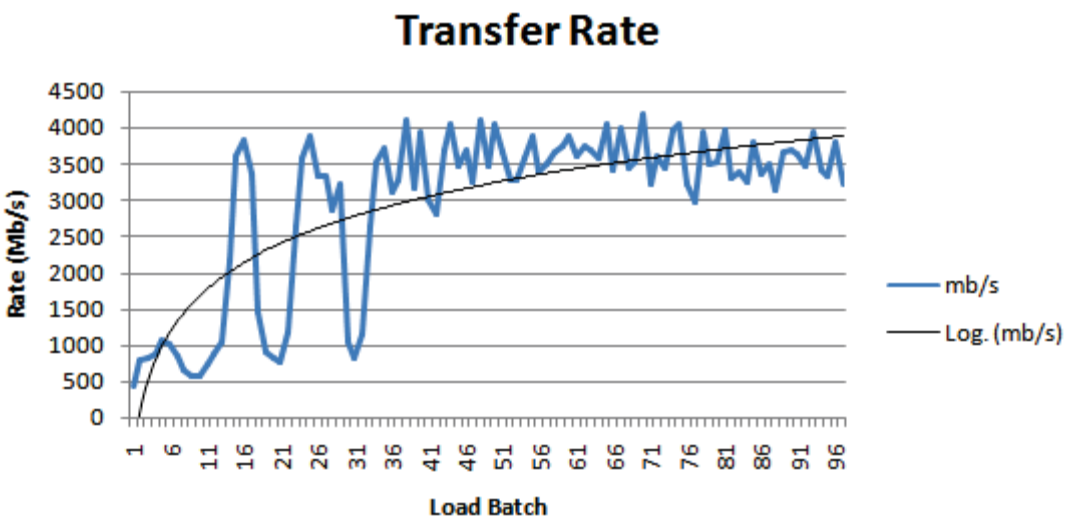

Figure 4.3 Graph of Transfer Rates across an execution.

4.3. Lighting and Shading Results

Two key factors influence the final image produced by the loading and rendering program - compression factor and method of data representation. The amount of data being pushed to the GPU each frame influences the possible speed of rendering. By having more vertices/points/normals being sent to the GPU this will negatively impact the rendering speed. Additionally, how compressed the data is impacts on the final image quality. A more sparsely (highly compressed) populated frame will require more interpolation and 'hole filling' algorithms to produce a high quality image. This is true for both parts of the rendering system in the case of C02 Viz, the terrain and the volume data.

Additionally, the rendering is influenced by data access locks. While the rendering system is somewhat independent of the CPU, the effects of loading are still seen in the frame-times, a frame may be delayed as it waits for new data to be loaded from the disk.

4.3.1. Terrain Rendering

While more of a minor element of the research contained in the thesis the terrain is still a required component for providing scale and scope to the Vulcan data. C02 Viz implements a ROAM 2.0 (Polack, T. 2002) method (detailed in Chapter 2, section 2.6) to integrate the terrain into the final image. C02 Viz is able to independently measure the rendering times of both the volumetric and terrain data for scientific comparison. The terrain data results shown in fig. 4.4, measured with the benchmarking tool in *Fraps* (Fraps, 2010) show the rendering of the terrain system during standard camera movements, a 360 degree rotation around the Y axis and several zoom operations.

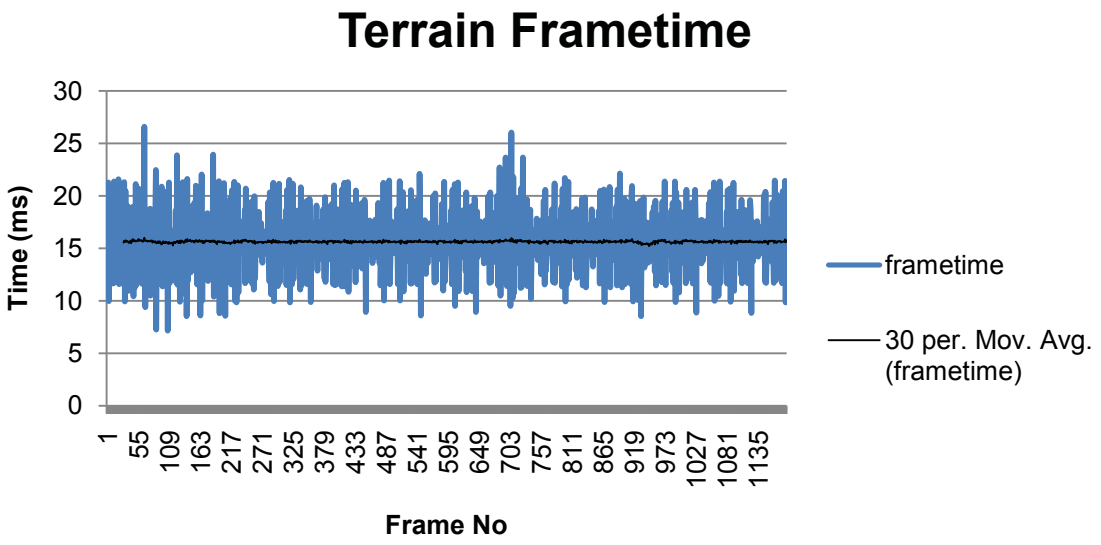

Figure 4.4 Graph of Rendering Times for Terrain Rendering.

The entire test process takes 1150 frames, or 19 seconds (with extra time for slow renders) to complete. Table 4.7 summarizes these results with 99% confidence intervals. It is clear that with a frame-time averaging 15.6ms, that obtaining a desired 60fps is easily achievable with this method.

Table 4.7
Summary of Terrain Rendering using ROAM 2.0

Technique	Max Frametime (ms/FPS)	Min Frametime (ms/FPS)	Mean Frametime (ms/FPS)	99% CI (ms/FPS)
ROAM 2.0	26.6/37.6	7.18/139.3	15.6 / 66.3	15.6 ± 0.215 / 66.3 ± 0.993

4.3.2. Volume Rendering

After the volumetric data is loaded into RAM, it is able to be rendered in real time, and the resultant rendering times are measured using *beepa Fraps* (Fraps, 2010). To test the rendering times, a view is chosen that displays the full extents of the frame and the data is allowed to play forward in time, automatically loading new data sets. Several variables are tested in this section, including method of geometrical representation; simplification tolerance ratio and in-thread/multithreaded loading. Finally the section concludes with a full frame-by-frame breakdown of results from optimal settings chosen using conclusions drawn from previous sections in this chapter.

4.3.2.1. Effect of Loading Method

The important results from figures 4.5 through 4.9 , summarized in table 4.8, are min and max frame-times and average FPS. The frametime/FPS isn't significantly affected by the loading technique as a whole. With a difference in means of less than 2 milliseconds (about 2 FPS) at the 60+ FPS margin there is no significant change, confirmed by a T test on the only differing values notable at 75 load and 15 load coming out at P = 0.95 (1 tail, type 2 test) .

Table 4.8
Summary of Loading Rendering Results

Technique	Max Frametime (ms/FPS)	Min Frametime (ms/FPS)	Mean Frametime (ms/FPS)	99% CI (ms/FPS)
75 Load	398 / 2.5	0.776/ 1288.7	16.4 / 70.5	16.4 ± 0.410 / 70.5 ± 1.587
50 Load	46.2/ 21.7	0.702/ 1424.5	15.6 / 72.6	15.6 ± 0.101 / 72.6 ± 1.845
25 Load	44.3/ 22.6	0.856/ 1168.0	15.6 / 68.95	15.6 ± 0.088 / 68.9 ± 1.116
15 Load	42.2/ 23.6	0.859/ 1164.0	15.6 / 68.5	15.6 ± 0.089 / 68.5 ± 0.901

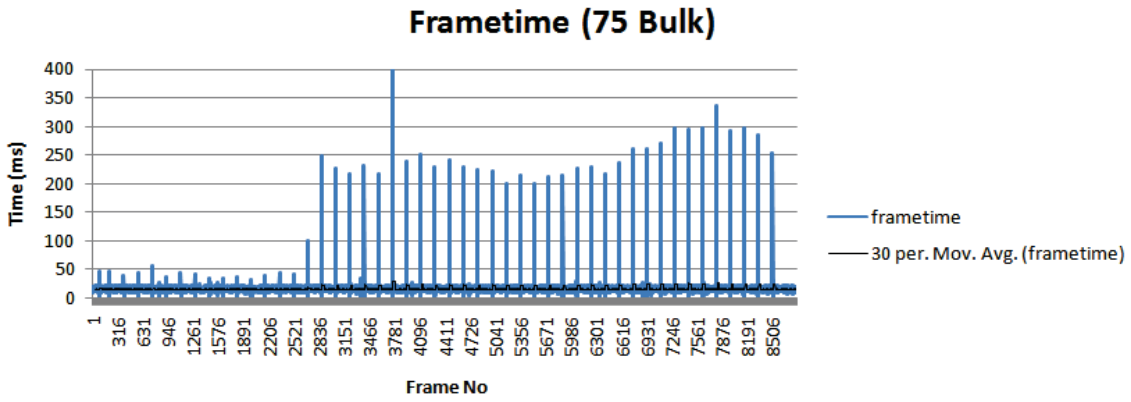

Figure 4.5 Graph of Render times from loading 75 frames at a time.

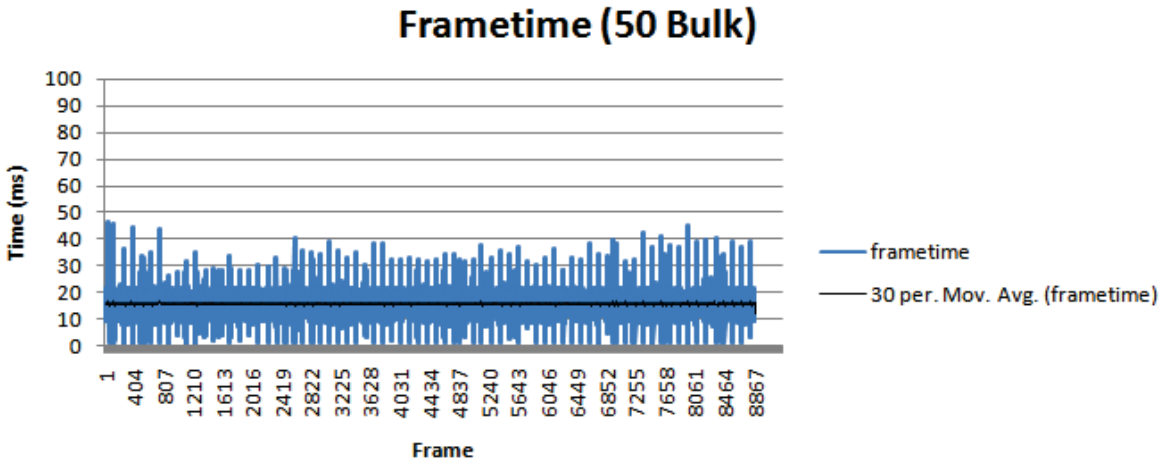

Figure 4.6 Graph of Render times from loading 50 frames at a time.

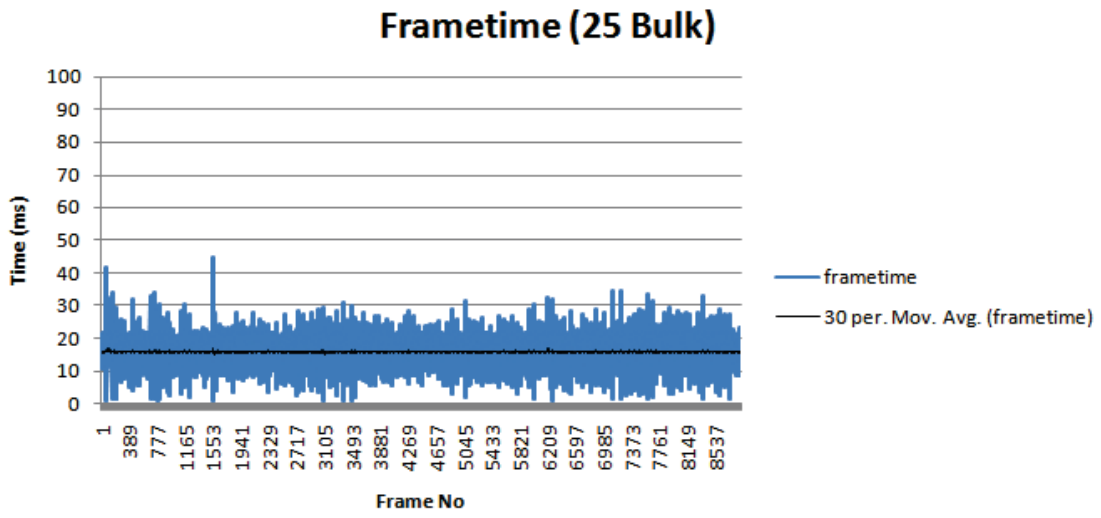

Figure 4.7 Graph of Render times from loading 25 frames at a time.

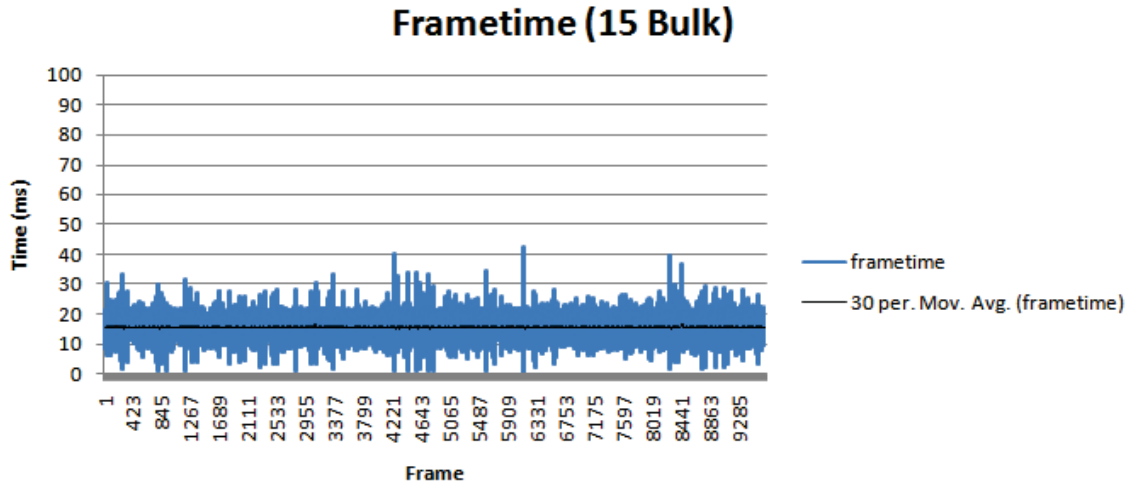

Figure 4.8 Graph of Render times form loading 25 frames at a time.

The "interruptions" to rendering are clearly visible on the graphs (figures 4.5 through 4.9) as peaks of noise. The noise is significant at loading levels 75 and 50 and by 15 frame bursts the loading is not visible outside the regular rendering times.

4.3.2.2. Effect of Data Representation

Next, examining the results shown in table 4.9 and fig. 4.9 , the effect of data representation on rendering time and loading can be observed. Table 4.9 summarizes fig. 4. 9 , showing that it is that while the population means are almost identical, the outliers (periods of loading) even under the smallest buffering period, in this case of this test, 15 frames at a time, are much greater and provide a much more disjoint ("stuttery") experience as the application is seen to freeze for up to half a second as the data is loaded from disk.

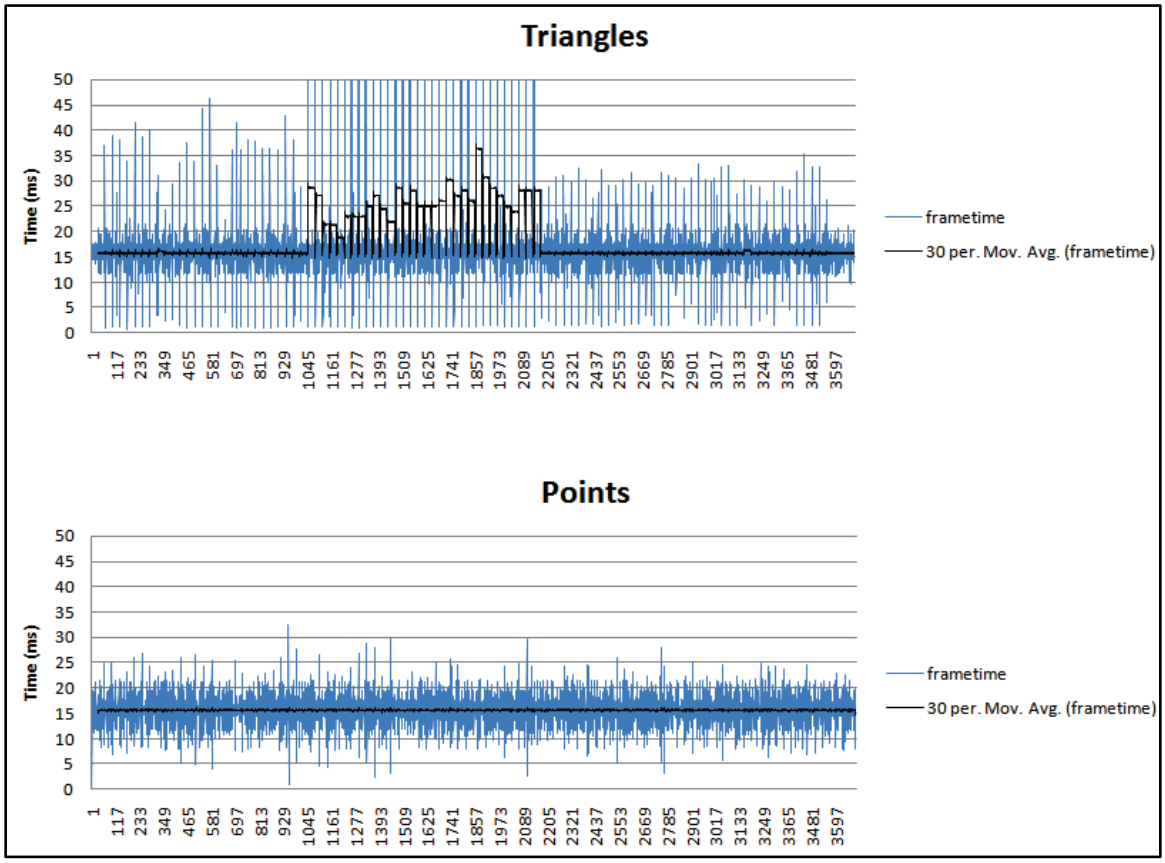

Figure 4.9 Comparative Graph of Triangle vs. Point Rendering on the Same sample.

Table 4.9
Summary of Geometric Representation Results

Technique	Max Frametime (ms/FPS)	Min Frametime (ms/FPS)	Mean Frametime (ms/FPS)	99% CI (ms/FPS)
Triangles	398 / 2.5	0.776/ 1288.7	16.4 / 70.5	16.4 ± 0.410 / 70.5 ± 1.587
Points	46.2/ 21.7	0.702/ 1424.5	15.6 / 72.6	15.6 ± 0.101 / 72.6 ± 1.845

With a statistical T test, there is a direct effect on the FPS on the rendering speed as a whole (P = 0.0000005, 2 Tails, Type 2). Triangles requiring more bandwidth to load (and thus taking more time to load in the equivalent amount of frames) and rendering statistically slower (but of little difference in the scope of a 60fps experience) are a poorer choice for a real-time rendering system.

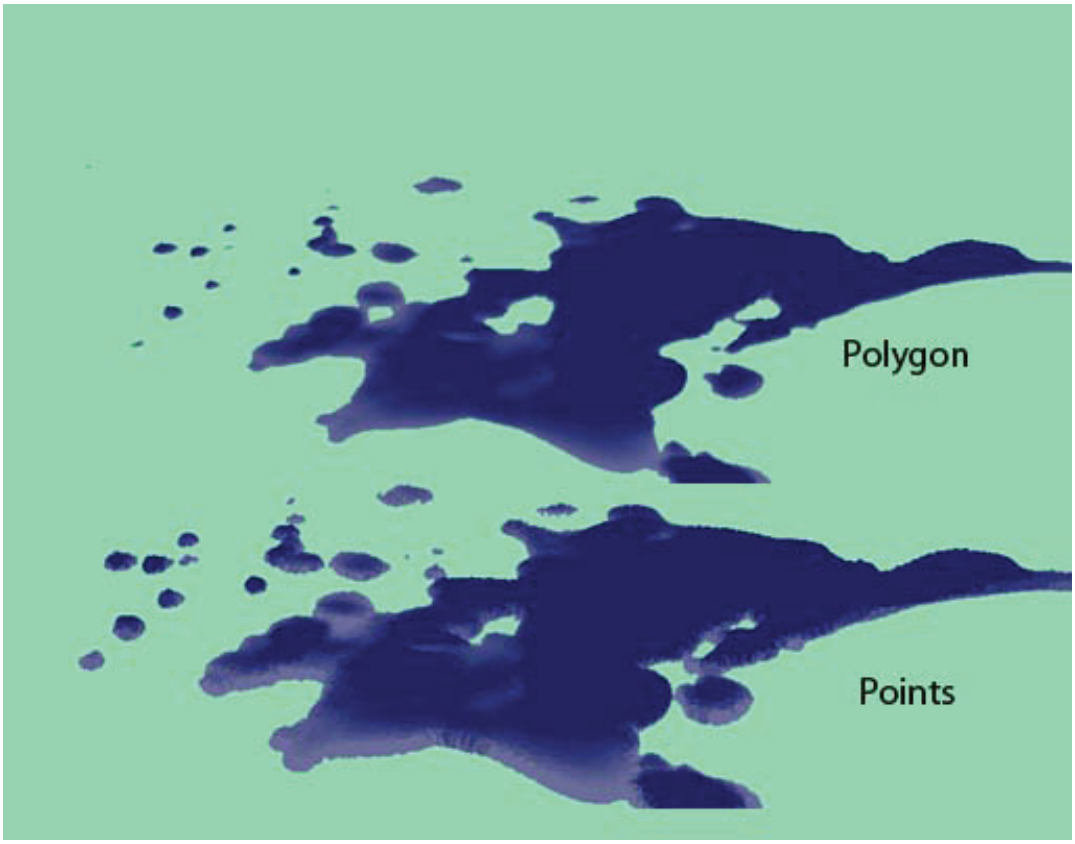

Figure 4.10 Graphically similar Point representation and Triangle representation.

The two methods of data representation, triangle and point, require two slightly difference rendering methods. The point based rendering system requires implementation of a point-size adjusting shader in GLSL to fill the holes between data points on the surface. However the triangle based rendering system can be simply run through a simple shader that does obligatory transforms and a ramp texture lookup. Analyzing the images produced by both triangle and point representation rendering techniques, seen side-by-side in fig. 4.10 reveal, that with triangle and point based representation, there is little visual difference, the only notable change being on the outer edges of the data.

4.3.2.3. Effect of Layer Interpolation Method

The performance of this run-time method can be measured by *beepa Fraps* (Fraps, 2010) as well and the results of testing are summarized in fig. 4.11 and table 4.10.

To test interpolation, the viewport was moved in a similar way to the test performed in section 4.4.1 for evaluating the terrain performance.

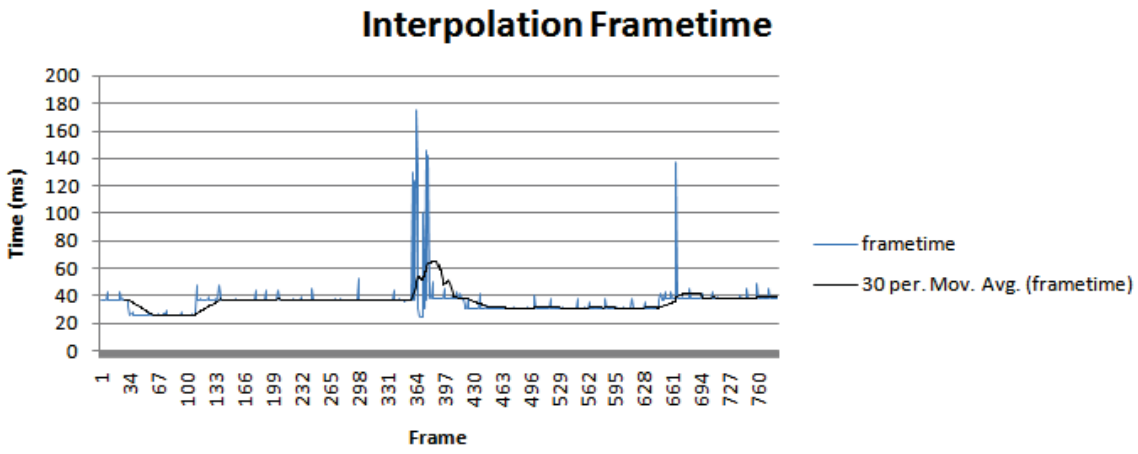

Figure 4.11 Graph of Render times from two layer interpolation using Triangle representation.

This method is chiefly dependant on the viewport size, and to simulate real-world use, the viewport was kept to 512x512 pixels the closest available match to the C02 Viz default size.

Table 4.10
Summary of Layer Interpolation Results

Technique	Max Frametime (ms/FPS)	Min Frametime (ms/FPS)	Mean Frametime (ms/FPS)	99% CI (ms/FPS)
Interpolation	175.6 / 5.69	24.9/ 40.2	36.1 / 28.9	36.1 ± 1.129 / 28.9 ± 0.430

As seen in table 4.11, the method has a significant impact on FPS, doubling the mean frame rendering time, and causing the system to run at approximately 30FPS. The difference between the results produced by layer interpolation and displaying the real data can be tested by rendering the same view of an interpolated mesh displaying position and normal information to texture and comparing these textures with the same information from the real data, and calculating the numerical difference. A sample interpolated view is shown in figures 4.12, 4.14 , 4.15 . In fig. 4.13 the common rendering errors are shown, geometry broken up by lines and floating "islands" of data in space.

4.4. Final Analysis

Taking an end-to-end approach across the entire software suite, a summary set of numbers, derived from results in previous sections can be analyzed in chain to determine the significant links in the image generation process. These summary results are shown in table 4.11

The optimal solution yields 96.6% compression on disk with loading times on average less than 1 rendered frame (16ms) long.

Table 4.11
End-to-end Results

Technique	Simplification	Representation	Loading	Interpolation
Optimal	YES ~0.01, duplicates removed	Points	15 burst, in thread	None

Continuing final analysis, the testing examines the full integration of the optimal solution. Sample executions of the optimal solutions are shown in figures 4.16 , 4.17 _x and 4.18 . Fig. 4.16 shows the terminal output of *Triconverter*, while figures 4.17 _x and 4.18 show two shots of execution taken from two positions from within a data series.

..

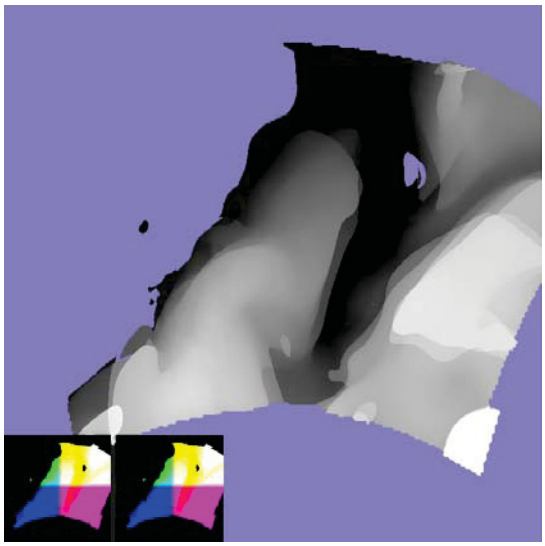

Figure 4.12 Interpolation Level 0.

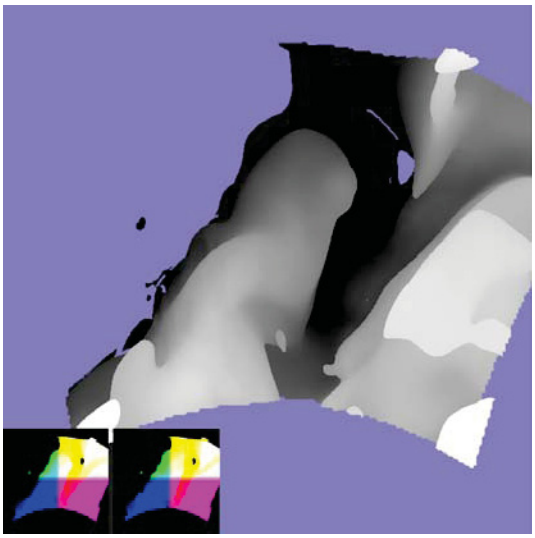

Figure 4.14 Interpolation Level 1.

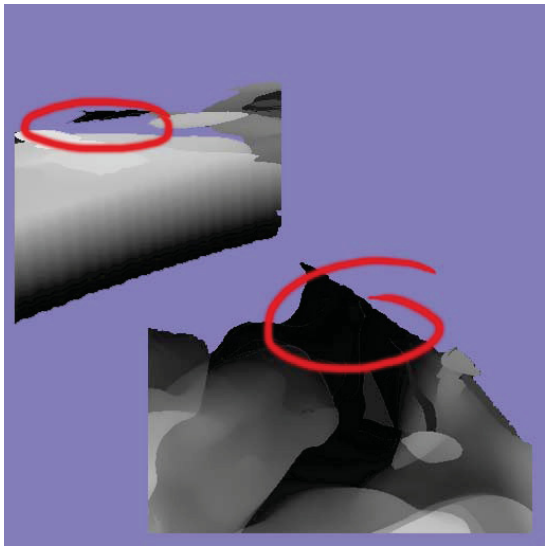

Figure 4.13 Common Artifacts in Interpolation.

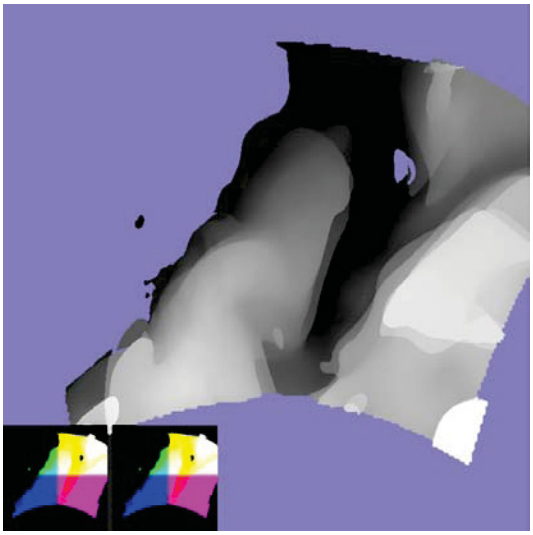

Figure 4.15 50% Interpolation between fig. 4.12 and fig. 4.14.

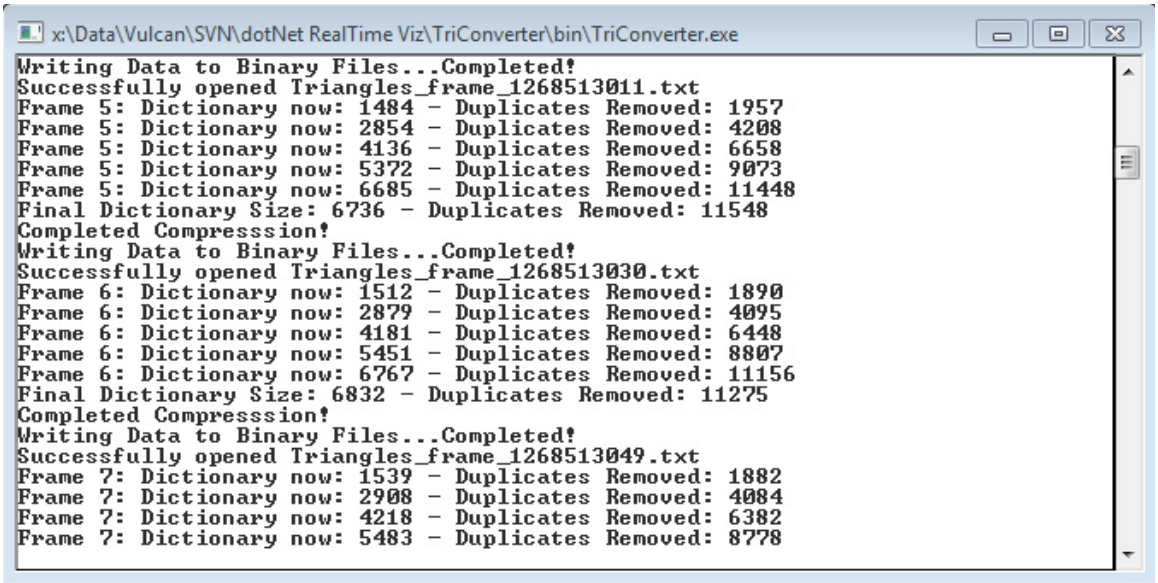

Figure 4.16 Sample Triconverter Execution.

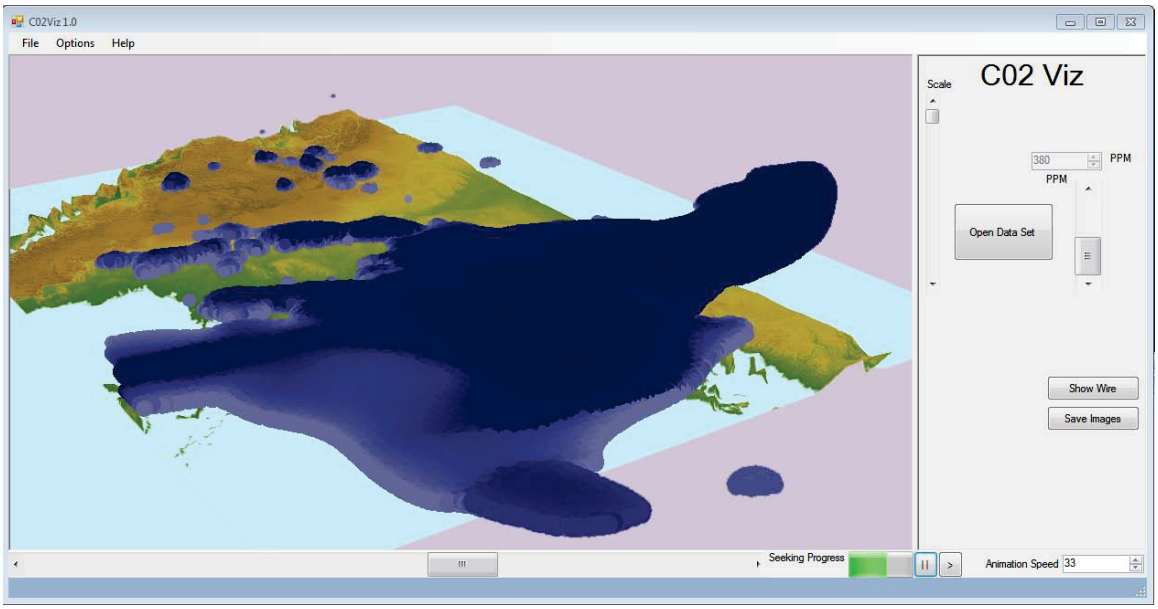

Figure 4.17 Sample Rendering Output A

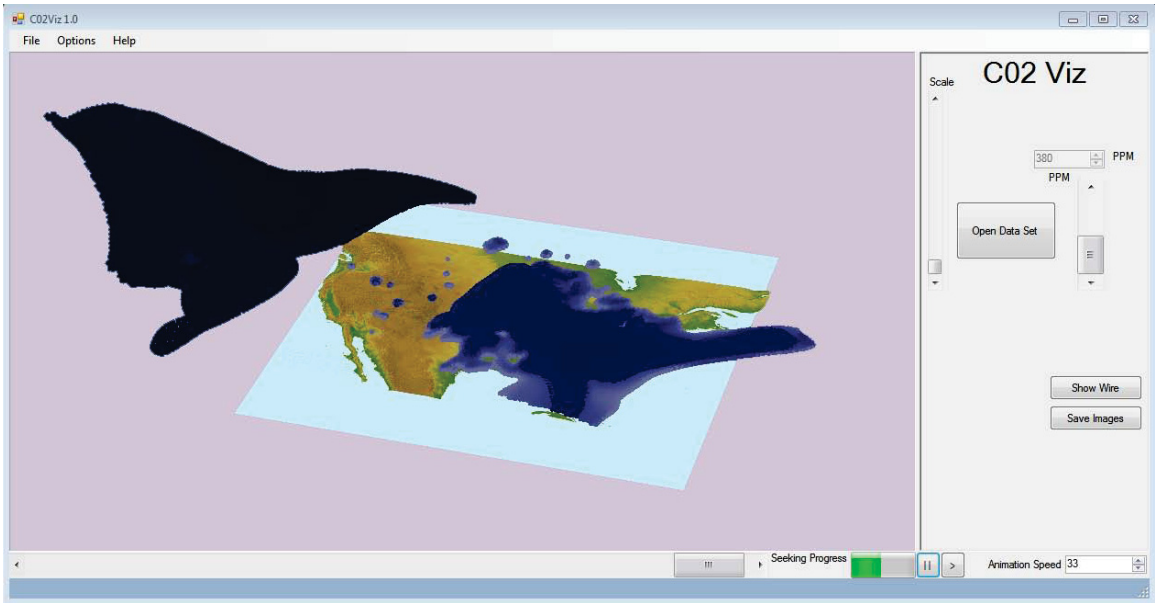

Figure 4.18 Sample Rendering Output B.

4.5. Chapter Summary

This chapter covered the results generated by testing the thesis research by implementation within a C++/OpenGL application and measurement across data on disk sizes and compression and rendering times. The algorithms were defined in pseudo code, detailing the results of each method/structure on the dependant variables indentified in the methodology.

Referencing the original hypotheses outlined in chapter 3, the following hypothesis have been proved true:

H_{a1} There is an (positive) effect of data compression on run-time rendering complexity and loading time.

There is a positive effect of data compression, as indicated in sections 4.1 and 4.2. By removing duplicate and degenerate entries and using binary representation thus reducing the data sizes by 96.6% reduces the spikes of

rendering time when loading and makes the loading of new frames invisible within regular rendering times

H_{a2} There is an (positive) effect of data representation on run-time rendering complexity and loading time.

By using point based representation, certain layer interpolation models are disabled, but loading times are significantly reduced, as shown in section 4.3.2.2.

H_{a3} There is a (positive) effect of loading method on the run-time rendering complexity, loading time, and data reduction.

This issue was explored in section 4.3.2.1, showing that loading either in thread or multi-threaded does not significantly change loading times, however by limiting the number of frames loaded at once to 15 or less, the loading times become invisible within regular rendering times.

CHAPTER 5. DISCUSSION

This chapter will discuss the results and side issues to methodological elements detailed in chapter 4 as well as postulate theories based on the data for volumetric rendering as applied to the Vulcan dataset. This chapter will begin with discussing the compression techniques and dealing with data on disk, then moving to loading methods and rendering data structures and finishing with lighting and shading and terrain techniques. In each section, additional details on the implementation process are discussed, in reference to results obtained. Finally the chapter is concluded by describing issues on further research that might take place to further the research continued within this thesis.

5.1. Data on Disk and Compression

The techniques chiefly responsible for modifying data on disk, that is the compression by simplification and data clean-up, had to be specially designed to work within strict memory limits. Initially, several compression methods not described in the results chapter were tried, including off-the-shelf software, both free and commercial that claimed to be able to reduce mesh sizes by many of the methods described in the literature review (surface approximation among others). The chief failing of many of these methods, that is, why they were not built into *Triconverter* or the rendering application, was that the individual frames exported from the Legacy application were too large. This would often result in application crashes, as these tools were designed for far smaller data sets. *Triconverter* succeeds because of its simplicity - simply removing wasted and degenerate data and moving it into a binary format provides the greatest reduction in file

sizes - and was the biggest factor in achieving real-time rendering performance after the compacted data was read into RAM.

5.2. Loading Methods

The final techniques chosen for rendering data, both multithreaded threaded and in-thread evolved after initially desiring to fit the entire dataset into memory at one time. Initially, iterative loading methods were implemented, hoping to load in partial representations of the data, and then refining the model over short periods of time. However as the number of frames present in a data series grew, from initial coarse tests of 700 frames to moving to the desired 5000+ frame series density, these methods became less and less effective, requiring significant pre-processing time at run-time application start-up. The final methods, of breaking up the data into small enough chunks that rapid binary reading of the data (streaming) could occur in such a short time that the overall data uptake was not noticed when analyzing the rendering frame-time results.

5.3. Rendering Data Structure

When considering the trade-off between triangles (polygonal) and points based rendering systems initially only a fixed function (OpenGL without GLSL) pipeline was considered. However, after implementation of a programmable shader assisted pipeline, one that included GLSL, the point based rendering system, with its drastically reduced memory requirements, showed to be the stronger method of the two.

5.4. Layer Interpolation

Blending two arbitrary surfaces is still an area of active research, and while this thesis presents a novel approach to solving this issue, it still requires

additional refinement. Subtle artifacts can occur when using the method with non-square viewports.

5.5. Lighting and Shading

This area of the research is open to additional human subjects testing. While this thesis attempts to quantifiable prove the effects of the use of a standard set of rendering techniques through rendering time analysis, there is a whole other hypothesis and testing regimen when the lighting and shading requirements are not taken as constant non-varying .

5.6. Terrain Rendering

A smaller portion of the research in this thesis, used mainly to provide spatial context for the Vulcan volumes - the ROAM 2.0 method with its 0.1s startup time and almost invisible effect on rendering times is a strong and stable algorithm used for the entire duration of the research. However, if this project were to be "scaled up" and used beyond North American carbon research - perhaps in the modeling of global carbon emissions- additional research would be required in the terrain rendering area. Geo-mip-mapping, in a style similar to that found in other atlas type packages could be an appropriate technique.

5.7. Further Work

The implementation used for testing was written in GLSL and was not fully optimized. There is room for additional research into the lighting and shading techniques available to boundary surface models rendering in real-time. For example implementing a global illumination model, complete with shadows is a starting technique change.

The implementation of rendering point-normal based representations was unable to be implemented during the timeframe of the research. While point based solutions were sufficient for real-time rendering, additional

The layer interpolation technique discussed in this thesis worked best for solid polygonal models. This technique could be expanded, or new techniques formulated for efficient blending of point-normal meshes. Additionally new testing of the error rates of these algorithms should be applied.

The Vulcan project has the capability to move to a global scale. There is a new research area in extending the work in this thesis to cover multiple geographical areas at once.

5.8. Chapter Summary

This chapter covered the main sections of work and research outlined previously in chapter four, but mentioning outside methods and techniques that did not make the final testing. Additionally the further work section outlined the possible continued research areas in new lighting and shading techniques and layer interpolation algorithms and how the Vulcan data could be extended for global geographical areas.

BIBLIOGRAPHY

BIBLIOGRAPHY

Andrysco, N., Gurney, K., Benes, B., & Corbin, K. (2009). Visual Exploration of the Vulcan CO2 Data. *Computer Graphics and Applications, IEEE*, *29*(1), 6-11. doi: 10.1109/MCG.2009.19.

Boucheny, C., Bonneau, G., Droulez, J., Thibault, G., & Ploix, S. (2009). A perceptive evaluation of volume rendering techniques. *ACM Trans. Appl. Percept.*, *5*(4), 1-24. doi: 10.1145/1462048.1462054.

Correa, W. T., Klosowski, J. T., Morris, C. J., & Jackmann, T. M. (2007). SPVN: a new application framework for interactive visualization of large datasets. In *ACM SIGGRAPH 2007 courses* (p. 6). San Diego, California: ACM. doi: 10.1145/1281500.1281566.

Department of Earth and Atmospheric Sciences. (2007). The Vulcan Project | Research. Retrieved July 2, 2009, from http://www.purdue.edu/eas/carbon/vulcan/research.php.

Fout, N., Ma, K., & Ahrens, J. (2005). Time-varying, multivariate volume data reduction. In *Proceedings of the 2005 ACM symposium on Applied computing* (pp. 1224-1230). Santa Fe, New Mexico: ACM. doi: 10.1145/1066677.1066953.

Fraps - 'beepa Fraps' (2010) Retrieved January 1, 2010 from http://www.fraps.com/

Gurney, K. R., Mendoza, D. L., Zhou, Y., Fischer, M. L., Miller, C. C., Geethakumar, S., and de La Rue du Can, S. (2009). High resolution fossil fuel combustion co2 emission fluxes for the united states. *Environmental Science & Technology*, 43(14):5535-5541.

Hadwiger, M., Ljung, P., Salama, C. R., & Ropinski, T. (2008). Advanced illumination techniques for GPU volume raycasting. In *ACM SIGGRAPH ASIA 2008 courses* (pp. 1-166). Singapore: ACM. doi: 10.1145/1508044.1508045.

Hwa, L., Duchaineau, M. A., & Joy, K. I. (2005, April). Realtime Optimal Adaption for Planetary Geometry and Texture. Retrieved July 2, 2009, from http://www.cognigraph.com/ROAM_homepage/ROAM2/hwa_tvcg04_revised.pdf.

Khodakovsky, A., Schröder, P., & Sweldens, W. (2000). Progressive geometry compression. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (pp. 271-278). ACM Press/Addison-Wesley Publishing Co. doi: 10.1145/344779.344922.

Lorensen, W. E., & Cline, H. E. (1987). Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Comput. Graph.*, *21*(4), 163-169. doi: 10.1145/37402.37422.

Luebke, D., Reddy, M., Cohen, J. D., Varshney, A., Watson, B., & Huebner, R. (2002). *Level of Detail for 3D Graphics* (1st ed.). Morgan Kaufmann.

Main Page - MC Wiki. (n.d.). . Retrieved July 2, 2009, from http://www.marchingcubes.org/index.php/Main_Page.

Neophytou, N., & Mueller, K. (2002). Space-time points: 4D splatting on efficient grids. In *Volume Visualization and Graphics, 2002. Proceedings. IEEE / ACM SIGGRAPH Symposium on* (pp. 97-106).

Polack, T. (2002). *Focus On 3D Terrain Programming* (1st ed.). Course Technology PTR.

Purdue University News. (n.d.). Carbon dioxide emissions map released on Google Earth. Retrieved July 2, 2009, from http://news.uns.purdue.edu/x/2009a/090219GurneyVulcan.html.

Rusinkiewicz, S. and Levoy, M. 2000. QSplat: a multiresolution point rendering system for large meshes. In *Proceedings of the 27th Annual Conference on Computer Graphics and interactive Techniques* International Conference on Computer Graphics and Interactive Techniques. ACM Press/Addison-Wesley Publishing Co., New York, NY, 343-352. DOI= http://doi.acm.org/10.1145/344779.344940

Turner, B. (2000, April 3). Gamasutra - Features - "Real-Time Dynamic Level of Detail Terrain Rendering with ROAM" [04.03.00]. *Gamasutra*. Game Programming Knowledge base, . Retrieved July 2, 2009, from http://www.gamasutra.com/features/20000403/turner_01.htm.

YouTube - 'Revolutionary' CO2 maps zoom in on greenhouse gas sources. (n.d.). . Retrieved July 2, 2009, from http://www.youtube.com/watch?v=eJpj8UUMTaI.

Zhou, J. (2005). The roles of perception for volume visualization and designing volume visualization methods based on perceptual factors. In *Proceedings of the 2nd symposium on Applied perception in graphics and visualization* (pp. 177-177). A Coroña, Spain: ACM. doi: 10.1145/1080402.1080462.

VITA

VITA.

I began my graduate studies at Purdue after completing my final year of my Bachelors degree on Exchange at Purdue University. Originally from Hastings, New Zealand, I came to Purdue to receive a world class education and to be closer to the cutting edge of technology research.

After graduation I plan to enter the workforce as a Software Engineer, using my research abilities to better solve the complex technological issues in modern software development - both in the Graphics area and Web Systems.