On-Demand Transmission Model Using Image-Based Rendering for Remote Visualization

A thesis submitted in partial fulfilment
of the requirement for the degree of Doctor of Philosophy

Asma Al-Saidi

July 2011

Cardiff University
School of Computer Science & Informatics

Summary

Interactive distributed visualization is an emerging technology with numerous applications. However, many of the present approaches to interactive distributed visualization have limited performance since they are based on the traditional polygonal processing graphics pipeline.

In contrast, image-based rendering uses multiple images of the scene instead of a 3D geometrical representation, and so has the key advantage that the final output is independent of the scene complexity, and depends on the desired final image resolution. These multiple images are referred to as the *light field dataset.*

In this thesis we propose an on-demand solution for efficiently transmitting visualization data to remote users/clients. This is achieved through sending selected parts of the dataset based on the current client viewpoint, and is done instead of downloading a complete replica of the light field dataset to each client, or remotely sending a single rendered view back from a central server to the user each time the user updates their viewing parameters. The on-demand approach shows stable performance as the number of clients increases because the load on the server and the network traffic are reduced. Furthermore, detailed performance studies show that the proposed on-demand scheme outperforms the current local and remote solutions in terms of interactivity measured in frames per second.

In addition, a performance study based on a theoretical cost model is presented. The model was able to provide realistic estimations of the results for different ranges of dataset sizes. Also, these results indicate that the model can be used as a predictive tool for estimating timings for the visualization process, enabling the improvement of the process and product quality, as well as the further development of models for larger systems and datasets. In further discussing the strengths and weaknesses of each of the models, we see that to be able to run the system for larger dataset resolution involves a trade-off between generality of hardware (the server and network) and dataset resolution. Larger dataset resolution cannot achieve interactive frame rates on current COTS infrastructure.

Finally, we conclude that the design of our 3D visualization system, based on image-based rendering coupled with an on-demand transmission model, has made a contribution to the field, and is a good basis for the future development of collaborative, distributed visualization systems.

Acknowledgment

I would like to thank all those who have made this thesis possible. First and foremost, I would like to thank my supervisor Professor David Walker for his understanding, encouragement and guidance. I want also thank him for giving me the space to experiment with my own ideas. I also want to thank my second supervisor Professor Omer Rana for his constructive comments.

Thanks also goes to the staff of the Cardiff School of Computer Science and Informatics, especially Mr. Robert Evans and Dr. Rob Davis for their technical help, even late at night and at weekends.

I would like also to thank my office colleagues for their support and fruitful discussions: Wan S Wan Awang, A. Elwaer, and H. Lenando.

I am indebted to many of my friends, Laila al Urimi, Haya Almagwashi, Iman Al-Kindi, Sonia Benghida and Fatimah Al-Rasbi for their encouragement and true caring from the start till the end of my PhD studies.

Finally, I want to thank my family for their constant support and care throughout my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Overview

Scientific visualization performs a key role in the exploration phase of large datasets, and facilitates their understanding and analysis. However, the increasing complexity of datasets often exceeds the rendering capabilities of the local processor. Moreover, recently in e-Science, an increasing demand has arisen among researchers to collaborate remotely through a geographically distributed environment. The focus of this thesis is to present and investigate an interactive remote visualization system which provides a highly interactive remote 3D visualization solution for large numbers of users using Image-Based Rendering (IBR).

This chapter introduces the work presented in this research. It discusses the main motivations behind the work and the intended research contributions. Finally, the structure of the thesis is outlined on a chapter by chapter basis.

1.1 Introduction

Nowadays we are witnessing a revolution in a wide range of technologies. Moore's law has successfully defined the trend of processor technology growth over the last half century, by predicting that processing capabilities double every two years [77]. Another version of the law called Butter's law says that the amount of data coming out of an optical fiber is doubling every nine months. Such growth has led to the emergence of new areas of research to investigate and explore.

While scientific visualization plays a key role in the exploration phase of large complex datasets and facilitates the understanding and analysis of such datasets [75], providing an effective visualization solution is challenging to achieve due to the fact that the increasing complexity of datasets often exceeds the rendering capabilities of local processors. Such challenges had previously limited visualization to only local access on powerful machines, or remote access to powerful machines via dedicated high throughput networks.

Furthermore, with recent advances in e-science applications, an increasing demand has arisen among researchers for a means to share data and perform analysis remotely. Consequently, visualization systems have started to evolve from standalone systems to distributed systems, both to exploit remote resources and to serve geographically remote users, thereby allowing them to collaborate and share visualizations.

The current widespread use of video and audio sharing sites such as YouTube, Yahoo Video [21], and flickr [15], over networks (particularly by the Internet) allows users to see video clips that have been uploaded by other users. Video-based visualization has two major drawbacks:

- The fixed camera path.
- The restriction to linear navigation along the video's time line.

Tan et al. [100] has classified the use of visualization into three categories: search to locate an object, gain knowledge, and inspection to maintain a particular view of the object. The above drawbacks shared among all 2D visualizations systems, limit the degrees of freedom of a user's 3D navigation experience in examining and analyzing datasets. The disadvantages of linear navigation becomes even worse for highly complex datasets and when users share their viewpoints with others.

In the past, the visualization of large datasets was limited to local users with powerful processing hardware, or remote users with dedicated networks. With the recent development of high-performance local area networks, a variety of distributed applications have emerged. However, current distributed visualization systems have either not provided a generic solution or have experienced a performance bottleneck in terms of the size of the dataset, and/or the number of concurrent users.

To construct an interactive remote visualization system two key challenges must be met and overcome. First, to ensure smooth navigation the distributed visualization system must be capable of delivering between 10-15 frames/second to each user connected to the system [32]. Such a high interactive frame rate is generally difficult to achieve for systems with low to mid range rendering hardware. The second issue is the limitation of the network resources, such as low bandwidth and high latency. In order for distributed visualization systems to work efficiently under conditions of high user interactivity, which places a heavy load on the communication infrastructure, careful performance characterization and tuning are required.

Current visualization systems require a traditional graphics pipeline, which uses 3D geometrical rendering to produce a 2D frame buffer that is then sent over the network. Once datasets become sufficiently complex the number of polygons [1]is too large to fit into the memory of a Graphics Processing Unit (GPU), and must be paged in from the main system memory. Once the dataset is larger than the local system memory, then these data must in turn be paged in from local disk.

Furthermore, mobile devices, such as smart phones, are becoming more widely used. These devices have less memory and weaker processing capabilities than desktop and laptop computers, and cannot accommodate modern GPUs because of chip size, cost, and power consumption considerations. There is, therefore, a need to investigate ways to provide interactive remote visualization solutions without requiring powerful processing hardware, and this is addressed by the on-demand approach proposed in this research.

An alternative to the traditional polygon rendering approach is the emerging area of Image-Based Rendering (IBR), which uses stored images of the 3D scene instead of its geometrical representation. IBR has received increasing attention recently due to the prospect of providing a more realistic representation of very complex scenes. In addition, by using IBR techniques the final output is made independent of scene complexity and depends only on the final image resolution [29], thereby reducing the overall computational cost of rendering the scene. Therefore, with IBR the rendering time is independent of scene complexity, enabling

[1]In 3D computer graphics, polygonal modeling is an approach for modeling objects by representing or approximating their surfaces using polygons.

constant frame rates if network bandwidth can be guaranteed. This thesis focuses on Light Field (LF) [2]rendering [70] which is one of the most well-known IBR representations. The LF approach essentially involves looking at 2D slices of 4D datasets. The light field can be represented as $L(u, v, s, t)$ where (u, v) are camera views and (s, t) are pixel positions in each view. The 4D LF dataset is generated for a targeted object in a pre-processing data acquisition phase using either a multi-array camera [6] or a moving camera gantry [9] for real objects, or by creating a synthetic scene using a modified ray tracing algorithm (see Fig. 1.1).

In addition to reducing the overall computational cost of rendering a scene, the discrete nature of the LF dataset, in which each image has a distinct representation, maps well to a hybrid solution which can overcome the performance drawbacks identified above. Instead of downloading a complete copy of the LF dataset to each client, or remotely sending a single rendered LF view back from a central server to the user each time a user updates their viewing parameters, our strategy combines both approaches. In response to a viewing query, initially the client cache is checked for the required images; if an image is not available it is retrieved from the server, and interpolated. The client cache is updated by storing the most recently used images. User navigation behaviour can be studied for random or orderly scenarios. For orderly exploration, in which user navigation follows a coherent pattern, it is possible to predictively pre-fetch the desired images, and this will further help to hide latency and improve client-side performance. Applying such a method will better utilize the rendering capability of the clients and minimize the server load, which will reduce the overall network traffic. Further-

[2]Method for generating new views from arbitrary camera positions by combining the input images which are images for the object from surrounding 6 directions.

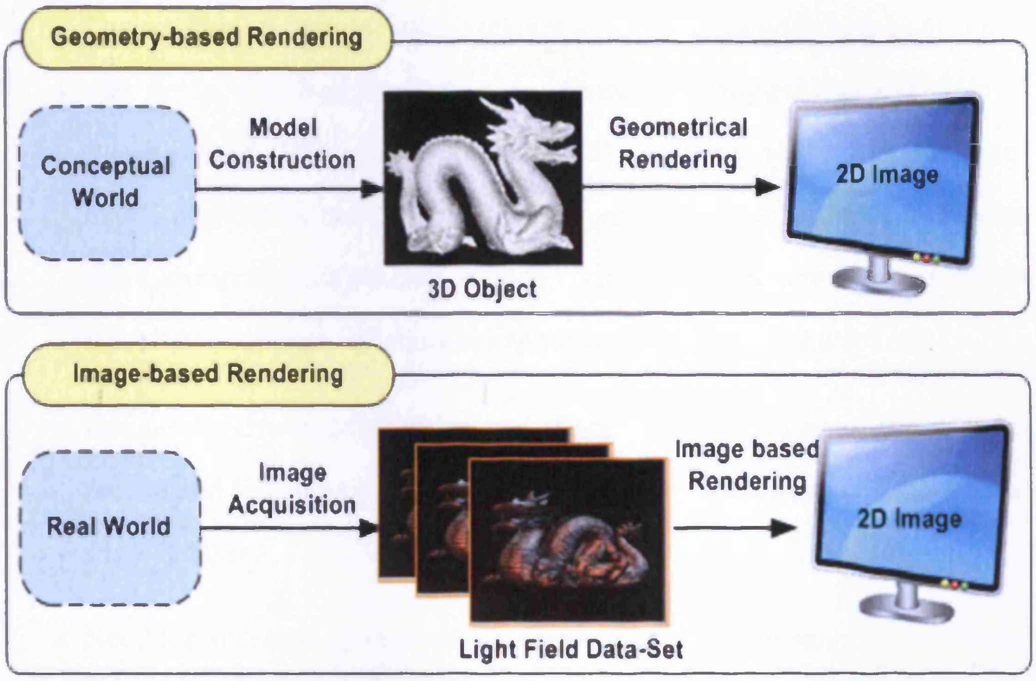


Figure 1.1: Geometry-based versus Image-based Rendering.

more, the excessive use of storage is avoided by using the on-demand download of the required partial datasets. We also present a performance study of the effect of deploying an on-demand transmission model. The design provides a novel combination of light field techniques and a transmission model which provides a general-purpose interactive solution for distributed collaborative visualization.

1.2 Research Problem and Motivation

Interactive distributed visualization is an emerging technology with numerous applications such as distance education and research collaboration. The main motivations for building an interactive remote visualization system can be summarized as:

- The continuous increase in dataset size generated in a wide a range of scientific fields, e.g., satellites or space stations (Terabytes/day) and medical scans dataset (100s of MB to a few GB). In view of the fact that automated algorithms have not provided a comprehensive solution to explore and understand such datasets, human visualization and interaction is still an essential component of many scientific and engineering disciplines.

- Geographically distributed experts. Experts from various scientific fields are located in geographically distributed places and need to collaborate and share datasets.

- Need for interactive photorealistic scenes. One of the primary goals of computer graphics is to create interactive photorealistic scenes. Achieving both interactivity and photorealism are contrasting goals. Creating photorealistic images involves the simulation of light propagation through an environment. To do this we must model the geometry of the objects in the environment. Modeling a real world with this process is exceedingly difficult because of the complexity of the geometry of the real world objects, and simulating the transport of the light cannot be done in a reasonable amount of time. On the other hand, interactive graphics has focused on hardware implementation of the rendering algorithm. In order to achieve interactive frame rates these systems use a comparatively low level of geometrical detail instead of global illumination. While the complexity of the scene that can be rendered by interactive systems is continually increasing, they are a still long way from producing photorealistic images.

- Scalability in terms of datasets and number of collaborative users. With the increasing demand from large numbers of users, in particular researchers looking for a means to share data and perform analysis in remote mode, the ability to scale systems to handle the increasing size of datasets and number of users in a cost-effective way is essential for present and future visualization systems.

- Recent developments in high-performance local area networks and wide area networks in terms of throughput and error rate, combined with the continuous increase in the number of computers connected to networks, has resulted in the emergence of a variety of distributed systems applications. However, existing distributed visualization systems have either not provided a generic solution or have experienced a performance bottleneck in terms of the size of the dataset or the number of concurrent users.

- Generic solution. For non-dedicated hardware and networks interactive performance is challenging. The design should leverage individual components of existing systems while making important innovations in other areas to provide a general-purpose visualization solution.

- Cost effectiveness. All users should be able to obtain a high quality and highly interactive visualization without the need for expensive hardware.

- Location transparency, where applications can be run independently of the location of both the researchers and resources. Remote visualization is an example of location transparency that provides access to geographically remote hardware resources.

Considering the above motivations, building such a visualization environment would experience an overall performance bottleneck. The two main sources for such a bottleneck are the actual rendering process, and limitations on the network resources. Current visualization systems require 3D geometrical rendering, and as datasets become increasingly complex the number of polygons is too large to fit into the memory of the Graphics Processing Unit (GPU), and must be paged in from the main system memory. Although, memory densities are increasing rapidly in line with Moore's law, the fact that datasets are keeping pace or overtaking these increases exacerbates the problem. Alternative approaches must therefore be investigated. The bottlenecks studied in this thesis match two of the most significant visualization challenges that were identified in 1999 by Hibbard [60] and revisited recently by Charters [39]. Although researchers in visualization have paid increasing attention to addressing these challenges, unfortunately some of them still remain unresolved. Our research focuses on two of these challenges. The first one is bandwidth flexibility. The wide variation in different network capabilities means that the visualization architecture should be able to adapt to network conditions. The second challenge is prediction algorithms, and addresses the need to deploy a prediction and pre-fetching mechanism that allows images to be rendered remotely and cached on a local client.

Our aims are centred on investigating an alternative method using Image-Based Rendering (IBR) which uses multiple images of the scene instead of a 3D geometrical representation. A key advantage of the use of IBR techniques is that the bandwidth required is independent of scene complexity and is therefore predictable, given knowledge of the desired final image resolution. IBR uses stored images of the object instead of the 3D geometry, thereby reducing the overall computational cost of rendering a scene, as the rendering cost depends only on the final image resolution. Therefore, final rendering is now independent of scene complexity, enabling constant frame rates if bandwidth can be guaranteed. Furthermore we are aiming to enhance the performance of rendering by using a hybrid rendering method which combines local and remote rendering. The corresponding images of a view are sent to the client for interpolation as more view queries are made. The old images are updated in the cache. This scenario will be studied for the random and orderly navigation cases [30].

1.3 Research Objectives

Based on the problem and motivations described in the preceding section, the objectives of the research described in this thesis are as follows:

- **To deploy and investigate an image-based rendering solution in a generic 3D remote visualization environment for a large number of concurrent users**. The purpose of this system is to seek solutions to problems found in current 3D visualization systems, namely the fluctuation in the frame rate due to its dependence on scene complexity, the tradeoff between generality in terms of hardware requirements and achieved performance, and the degree of user expertise needed in using a system. Some systems use dedicated hardware in order to achieve high performance, while others are designed for specialist users and may have a steep learning curve.

- **To study the interactivity and stability of the overall system performance**. We aim to investigate the effectiveness of the image-based rendering approach in terms of interactivity and the number of users (or visualization clients). Furthermore, we intend to address questions relating to the stability of the system in terms of the delivered frame rate for a complex dataset. The purpose of these analysis studies is to learn how to efficiently handle large datasets and increasing numbers of concurrent users.

- **To investigate different distributed light field rendering transmission models**. The purpose of this investigation is to seek a solution to problems associated with the current remote rendering approach, which sends the output image for each rendering request from the server to the requesting client, and the local rendering approach in which all the data are sent to the clients initially. We also aim to investigate an on-demand method to transmit data effectively in order to enhance the system performance by reducing the server load and utilizing client rendering capabilities.

- **To enable user-controlled viewpoints for both real and synthetic data**. Visualization is used in many areas, which requires the user to access different kinds of datasets – either real or synthetic. Real datasets are typically produced from multiple camera views, and synthetic datasets are usually created using a given scene geometry and light sources using a ray-tracing algorithm.

1.4 Research Hypothesis

3D collaborative visualization environments, created by distributed light field rendering techniques with tunable performance parameters, provide a generic, highly

interactive, remote 3D visualization solution for large numbers of users, enabling user controlled viewpoints for both real and synthetic data.

1.5 Scope of the Research

1.5.1 Applications Context

In general, the visualization system could be used for any static or time dependent field. To exemplify the category of applications we have selected the medical education field, where proving physicians and students with an accessible 3D exploration and analysis for medical data could makes a significant contribution to improve medical care in general and it can also provide an extra dimension to the learning process diagnosis, procedures training, and collaborative research. Our system can be applied also to non-medical domains and could cover various field such as geologists and engineers wishing to view a large, complex model (such as an oil drilling platform), or any user of visualization wishing to view complex datasets that would otherwise overwhelm a single graphics processor.

1.5.2 Targeted Users

In general, mainstream visualization systems target computationally intensive problems for high-end users. Such visualization is usually carried out on a high performance computing, data-storage and network infrastructures. As an alternative, in this research we investigate an affordable approach to support low-end users and devices. The system is generic and provides a visualization framework for:

- Low end users without any previous expertise or training requirements. One of the primary motivations of this work is to provide an opportunity for users to perform interactive visualization without the burdens typically associated with the remote visualization process. Most current visualization systems require skills in distributing/Grid computing, visualization skills and knowledge in particular scientific areas. This presents a steep learning curve for users to master all this knowledge. Our research aims to provide different ways to access visualization for different kinds of users and avoid this steep learning curve for users.

- Low end devices with insufficient rendering capabilities, e.g., Personal Digital Assistants (PDAs) or smart phones. Also high-end devices with general purpose network connections

Any higher performance devices could also join the remote visualization.

1.6 Research Contributions

The contributions of this thesis are as follows:

- The design and implementation of a remote visualization system using light field rendering. We shall focus on building a collaborative visualization environment for distributed users that involves complex datasets, but with static geometry and static illumination. The design provides a novel combination of light field techniques and a transmission model which provides a general-purpose interactive solution for distributed collaborative visualization.

- An on-demand transmission model for transmitting datasets based on the user/client viewpoint parameters. In this model excessive use of storage is avoided by downloading only partial datasets.
- A performance study for system behaviours for three different transmission models under different viewpoints, datasets sizes and viewers. We also present a performance study of the effect of deploying a streaming mechanism.
- A theoretical cost model for local rendering, remote rendering, and on-demand rendering. This model will provide a comparison with experimental results which enables validation of these results. The models will be used to determine which approach is most cost effective in each situation. Furthermore, the models will be used to provide a predictive tool to estimate the computation and communication costs in scenarios where parameters are different from those presented.

1.7 Organization of the thesis

This thesis is organized as follows:

Chapter 2 - Background and Literature Survey

Presents a review of the relevant literature related to the current work areas which we cover and interrelate: remote visualizations systems, Image-Based Rendering with particular focus to Light Field rendering, and transmission models for distributed graphics.

Chapter 3 - Light Field Rendering for Remote Visualization

Describes the design of the developed system architecture. It discusses the different phases and the potential applications for this system. Furthermore, it discusses the three possible transmission models with particular focus on the on-demand transmission model.

Chapter 4 - Experimental Results and Discussion

Illustrates the various experimental results from the system with different performance metrics. It also presents an analysis of experimental benchmarks and the scalability of the system.

Chapter 5- Theoretical Cost Comparison for IBR for Remote Visualization

Evaluates the system using a theoretical cost model for local rendering, remote rendering and on-demand rendering. This model provides a comparison with the experimental results presented in Chapter 4 which enables validation of these results. These models are used to determine which approach is most cost-effective for each situation. An overview of the lessons learned during the development of the system is also included.

Chapter 6 - Conclusions

Concludes this thesis with a review of the original research contributions based on the main findings of the research.

Chapter 7 - Future Work

Briefly examines some of the future directions that the work could be expanded into.

1.8 Chapter Summary

This chapter has giving an overview of this thesis. We also presented the main aims of our research and the major contributions which we have achieved through this research.

Chapter 2

Background and Literature Survey

Overview

This chapter presents an overview of the basic concepts and terminology used throughout the thesis. In addition, it surveys the current key state-of-the-art technologies associated with the three areas we are aiming to bridge between: Image Based Rendering, distributed visualization systems, and transmission models for distributed graphics.

This chapter is organized as follows. First, in Sections 2.1and 2.2 we introduce the basic concepts. Secondly, we review different stand-alone visualizations systems in Section 2.3, and different remote visualization systems in Section 2.4. Then we discuss Image-Based Rendering in general in Section 2.6 and then focus on the Light Field rendering approach used throughout this thesis in Section 2.6.2. Section 2.8 gives an overview of the different transmission models and their classification, and we describe the related work for each of the techniques. Finally, Section 2.9 summarizes the main points of this chapter.

2.1 Introduction

To understand the existing visualization solutions and contributions three basic questions need to be answered.

What is visualization? The term visualization is ambiguous and has been used differently in different areas (science, design and art, business, etc.). Several attempts have been made to define this field. Generally, visualization can be defined as the process of representing abstract objects as concrete images perceivable by the eyes and brain. Scientific visualization is applying the ability to visualize abstract things to help improve our understanding of arbitrary concepts and phenomena, and is often based on data sets gathered by various instruments or generated by software simulations[67].

Why use visualization? The goal of visualization is simplification and interpretation, ultimately to allow scientists to more easily understand and share their data. It is much easier to understand a visual image than a plain ASCII text file containing thousands of data points. Thus, visualization supports simulation and collaboration.

So What? The importance of visualization is that it leads to efficient scientific research. Whether this visualization occurs in your own head or occurs on a computer screen, or other device, visualization is vital to understanding many scientific problems.

2.2 Scientific and Information visualization

Information visualization is the interdisciplinary study of “the visual representation of large-scale collections of non-numerical information, such as files and lines of code in software systems, library and bibliographic databases, networks of relations on the internet, and so forth" [47].

In contrast, scientific visualization studies numerical data. Scientific visualization aims to allow users to gain insights into the data and a deeper understanding of it, especially the increasingly large datasets which can be generated from simulations or experiments. By transforming numeric data into graphics, visualization provides scientists with an opportunity to discover unseen features and relationships hidden in the data. For the scientific community, visualization has became a very important method for scientific discovery and research. In 1998 McCormick et al. even claimed that "in many fields it [visualization] is already revolutionizing the way scientists do science" [75].

2.3 Stand-alone Visualization Systems

Stand-alone visualization systems, sometimes refereed to as single-user environments, allow users to visualize and analyze a dataset on a single machine. Such systems normally require an expensive graphics workstation that may not available to many organizations or users. Examples of such systems are discussed in the following subsections.

2.3.1 VisIt

VisIt [20] is a free interactive parallel visualization and graphical analysis tool for viewing scientific data on Unix and PC platforms. Users can quickly generate visualizations from their data, animate them through time, manipulate them, and save the resulting images for presentations. VisIt contains a rich set of visualization features so that users can view their data in a variety of ways. It can be used to visualize scalar and vector fields defined on two- and three-dimensional structured and unstructured meshes. VisIt was designed to handle very large data set sizes in the terascale range, and yet can also handle small data sets in the kilobyte range.

2.3.2 ParaView

ParaView [23] is an open-source, multi-platform data analysis and visualization application. ParaView users can quickly build visualizations to analyze their data using qualitative and quantitative techniques. The data exploration can be done interactively in 3D or programmatically using ParaView's batch processing capabilities. ParaView was developed to analyze extremely large datasets using distributed memory computing resources. It can be run on supercomputers to analyze terascale datasets as well as on laptops for smaller data.

2.3.3 VTK

The Visualization Toolkit (VTK) is an open-source, freely available software system for 3D computer graphics, image processing and visualization. VTK consists of a C++ class library and several interpreted interface layers including Tcl/Tk, Java, and Python. VTK supports a wide variety of visualization algorithms including: scalar, vector, tensor, texture, and volumetric methods; and advanced modeling techniques such as: implicit modeling, polygon reduction, mesh smoothing, cutting, contouring, and Delaunay triangulation. VTK has an extensive information visualization framework, has a suite of 3D interaction widgets, supports parallel processing, and integrates with various databases on GUI toolkits such as Qt and Tk. VTK is cross-platform and runs on Linux, Windows, Mac and Unix platforms [12].

2.3.4 Amira

Amira is software platform for visualizing, manipulating, and understanding Life Science and bio-medical data coming from all types of sources and modalities. It exploits the latest graphics cards and processors and it has an intuitive user interface [3].

2.4 Remote Visualization Systems

Distributed visualization allows different parts of the visualization pipeline to be run on different machines. Therefore, we can optimize the use of different hardware resources in order to achieve a good visualization result. As classified by Haber and McNabb [57], usually a complete visualization process involves four types of steps with distinctive characteristics: Data, Filter, Map, and Render. We illustrate the distributed visualization by using an example visualization pipeline (see Fig. 2.1). In this pipeline, the Data step is a simulation which provides large-scale raw data every time step and also requires high computational capability on the machine where it is run. The Filter step can reduce the size of the data by applying some refining algorithm. Therefore, in the distributed visualization as the data needs to be transferred between different locations, the Filter step can significantly reduce the time cost for data transfer. Computational capability is also essential in the Map step, which is the process of converting data into a geometric representation. To gain high quality graphics at the Render step, powerful graphical processing hardware is needed. For end-users, a high resolution display screen will be important in order to have a clear view of the visualization result. However, without the distributed visualization, it is difficult and expensive to build up a machine which meets all the requirements for this example visualization pipeline.

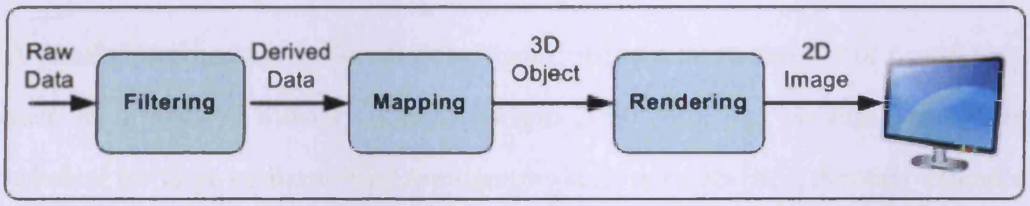


Figure 2.1: Visualization pipeline of Haber and McNabb.

Taking advantage from their modular features, most Modular Visualization Environments (for example, IRIS Explorer, AVS, etc.) enable users to distribute different modules onto different machines. More details and examples of these systems will be presented in Section 2.4.4.

Remote visualization refers to the interactive viewing of three-dimensional scientific data sets over the network. Because scientific data sets are in the gigabyte size range (or larger), it is difficult to send the entire data set over the network sufficiently quickly. Extraction, processing, network latency and rendering times add up and make the proposition of near real-time interactive visualization a challenge. Moreover, the client may have a limited amount of memory and CPU power for viewing and interacting with the data.

2.4.1 Specialized systems

Despite the fact that advances in computer graphics rendering techniques, such as ray tracing [49], are able to generate highly realistic images, they are still too slow to be used for 3D real-time applications using local desktop systems and limited network bandwidth that cannot support the transfer of large datasets where interactive frame rates are essential.

2.4.1.1 Specialized hardware

A parallel processing version of a ray tracer [1]using a large number of processors, such as a 60-CPU Silicon Graphics Origin 2000 [81], can provide a real-time solution for large computerized tomography scan datasets [82]. Another example is the Visapult system which utilizes specialized hardware [34], and performs a high speed parallel rendering process for a massive dataset (1-5Tb).

2.4.1.2 Specialized Networks

A different approach is to focus on minimizing communication effects by exploiting dedicated networking, such as ATM [68], to ensure low latency and high throughput. Unfortunately, such powerful systems are only accessible by a few

[1]Ray tracing algorithm describes a method for producing visual images constructed in 3D computer graphics environments, with high photorealism. It works by tracing a path from an imaginary eye through each pixel in a virtual screen, and calculating the color of the object visible through it (for more details refer to section 2.5).

users, and are mainly targeted at a limited class of datasets, such as volumetric datasets.

In the approach of Jin et al. [64] shared network storage is used, by placing multiple network buffers close to the clients. This approach allows the concurrent download of data, and provides fast access to large data sets. However, such infrastructure requires the installation of the network buffers for each client, which requires some technical skill and places a burden on general users.

2.4.2 Web-based Visualization

Cactus [13] provides web-based visualization by streaming the data to the user's desktop through an HTTP connection. This approach does not scale well with the larger datasets often created by high-performance computing applications.

The Cactus project a little out of place here as it is actually a problem-solving environment that consists of many modules. However, Cactus is designed for easy parallel work and collaborative development, and integration with Globus and the Grid has reached a mature stage. Additionally, Cactus provides for code development in a number of variants of C and Fortran, and supports cross-platform code development and execution. Cactus provides an abstract layer above the Grid middleware through its provision of 'thorns' (Cactus components) that implement different execution methodologies on the Grid [31], for example an MPI thorn. I/O is handled in basically the same way. Cactus can also link to visualization products such as Amira [3] to produce high-quality graphics. Using Cactus' socket-based data streaming capabilities, remote visualization of live computations can be performed. The collaborative environment is enabled through the ability of Cactus to send a data stream to multiple clients simultaneously [62]. A number of additional tools have been developed for Cactus on the Grid, including checkpointing of distributed simulations and remote application steering, as well as in developing portals to access Grid services.

2.4.3 Grid Computing

Following a different line, Grid computing promises a high performance distributed infrastructure that may be used for high-end visualization. For example, gViz [105] provides a middleware layer between a Grid-based environment and the IRIS Explorer [22] dataflow visualization software.

2.4.3.1 The Grid Globus Toolkit

The Globus Toolkit [4, 45] is a collection of software components designed to support the development of applications for high-performance distributed computing environments, or Grids [61]. The aim of the Globus Toolkit is not to provide application developers with a monolithic system, but rather a collection of standardized and standalone services. Each service provides a basic function such as authentication, resource allocation, information, communication, fault detection, and remote data access. The toolkit forms the heart of many approaches, but it is not a silver bullet. In [72] Lu et al. discuss the shortcomings and inefficiencies of Globus. Here, inefficiency arises on many levels, but stems mainly from the GRAM protocol. (1) File transfer is inefficient, as jobs cannot share file instances, which are subsequently transferred multiple times. (2) Authentication must be performed for each submission and file transfer, even when the same action is repeated multiple times. (3) Job descriptions must be modified for use with Globus. Condor-G, for example, requires a significant amount of extra metadata in comparison to a standard job submission. (4) Once the job is submitted to Globus it may then also be converted into a multi-request RSL (Resource Specification Language) job description. For example, if in a Condor submit script, queue = 10 (meaning submit 10 instances of this job) this will create ten Globus jobs rather than a job with an arity of 10. (5) As there is no resource broker, there are no means to perform load balancing. (6) Submit machines become performance bottlenecks because of the file replication process. (7) All clusters are seen as homogeneous. Although Globus implementations are improving new versions are often not backward-compatible and require significant changes to the core implementation of an adopted approach. From the perspective of running very domain-specific jobs with well-defined and rigid requirements of the operating environment, Globus is not a good candidate.

Grid systems have proven to give high performance to high-end users with access to high-end devices. Users employ dataflow techniques to define modules to run on remote machines, allowing better allocation of Grid resources and consequently better performance. At the user level, gViz requires Grid development skills. Shalf and Bethel [91] concluded that the current state of visualization software is not well-suited for exploiting the Grid, and that a new Grid-aware framework is needed for distributed visualization.

2.4.4 Modular Visualization Environments (MVEs)

Modular Visualization Environments (MVEs) are simply defined as modular blocks of routines which perform certain functions linked in an interactive visual programming style to build a visualization program [38]. Although widely used for post-processing simulation data, these systems are also interesting for computational steering because they allow the scientist to interact with the simulation code itself. Examples of this type of environment are IRIS Explorer [22], SciRun [24], and Paraview [23]. These systems are typically designed to be used by a single user at one location and are limited by the types of resources they execute on and the size of dataset they can manipulate.

The COVISA system [106] is a collaborative visualization environment built on top of IRIS Explorer. It has a master node that steers the visualization process and sends any changes in the data to the slave nodes. The slaves nodes act as viewers and may also have a local viewpoint. It is assumed that every slave node has a rendering capability (i.e., graphics hardware).

The ARTE environment [73] presents a hybrid approach whereby a full bitmap or geometry may be transmitted, but runs as a single server on a single platform and does not make use of remote resources. The system restricts the users to one view. Commercial visualization solutions are available, such as OpenGL VizServer [7] which extends OpenGL to work remotely, on the assumption that participating machines have no rendering capability. VizServer performs the rendering process and transmits the resulting frame buffer to the other machines in the system. This system restricts the users to one view.

2.4.5 Video and Data Conferencing

Video and data conferencing systems offer an effective solution to enable distributed collaborative teams to communicate and share ideas and images [84]. Although video-conferencing facilitates a form of face-to-face meeting experience, viewing and exploring data and models remains an essential requirement for collaborative teams [48]. Video-based visualization inherits two major drawbacks from 2D visualization: the fixed camera path and the restriction to linear navigation along the video's time line.

2.4.6 Desktop Sharing

2.4.6.1 VNC

The Virtual Network Computing (VNC) system [89] uses desktop sharing to provide remote visualization without moving the data to the clients. This approach supports high performance for complex datasets by sending the whole desktop screenshot at interactive rates. VNC consists of a server part and a client part. Multiple clients may connect to the same VNC server at the same time. The server specifies what area on the screen is going to be shared and what control permissions are offered to the connecting clients. VNC does not only share pixels on the screen amongst users, but also shares the control of the mouse or keyboard which means clients can also manipulate the items on the shared screen over a network, relaying the graphical screen updates back in the other direction. There are various types of VNC, among which the most widely-used VNC systems are RealVNC [10] and TightVNC [11].

This approach has two advantages: it is application-independent and it does not require changes in the interface. On the other hand, this strategy suffers from poor interactivity when limited bandwidth connections are used, and performance is not optimized since it is not tailored for any specific application.

2.4.6.2 Microsoft NetMeeting

Microsoft NetMeeting [16] and its successor, Office Live Meeting [17], share applications among distributed users through a screen sharing approach. However, low resolution bitmap images and low frame rates make it inappropriate for interactive manipulation of large datasets. Other applications, such as VisMockup [25] provide a real-time, 3D visualization solution that allows users to interact in a single visual environment. Only limited applications are supported and each of the users is required to have a local copy of the whole dataset which becomes a problem for large datasets and for computers with limited capabilities.

2.4.7 General-purpose Visualization Systems

To be able to create a general-purpose visualization system, we need to describe what characterizes such a system. General-purpose visualization engines are of many varieties. Domingue et al. [43] has summarized general-purpose visualization as a way to visualize and to make a good graphical design that may yield many different representations: text versus graphics, level of abstraction, displaying control versus data structures, static versus dynamic visualizations, one or multiple views, behaviour versus performance, and so forth.

This general aim can be achieved in different ways and requires an effort to abstract the target entity. Such a visualization system should further enable visualization sharing and enhance collaboration, which would allow non-constrained visualization solutions, instead of the existing visualizations solutions which are specific to groups of people and domains.

2.4.7.1 Resource Aware Visualization Environment

While several of the current remote visualization systems are standalone systems modified to work on remote resources, the Resource Aware Visualization Environment (RAVE) project is based on a new architecture [52, 54, 53, 55, 56]. RAVE is a Grid-enabled visualization system that supports heterogeneous machines, from high capability machines to PDAs, regardless of their underlying architecture. The RAVE system aims to choose appropriate rendering services, either remotely or locally, based on the rendering capabilities of the client.

It is increasingly evident that scientific, computational and other users require access to collaborative visualization resources that are currently unavailable to them in their own environment. Currently, the National Grid Service (NGS) does not provide such a service, meaning that any visualization or rendering of datasets of information must be done locally.

The Resource-Aware Visualization Environment (RAVE) is a distributed, Grid-enabled collaborative visualization environment that supports automated resource discovery across heterogeneous machines. Rather than commandeering an entire machine, RAVE runs as a background process using Web services, thus enabling resource usage to be optimized and shared between users. RAVE supports a wide range of machines, from hand-held PDAs to high-end servers with large-scale stereo, tracked displays. The local display device may render all, some, or none of the data set remotely, depending on its capability and present loading. This enables individuals to collaborate from their desks, in the field, or in front of specialized immersive displays.

RAVE provides a scalable and robust environment on which to create a production quality service that provides visualization and rendering capabilities to researchers working collaboratively over geographically dispersed regions.

2.4.8 Summary of Current Distributed Visualization Systems

Distributed visualization at present is often restricted to specific problem domains, such as volumetric rendering [2], or forwarding a single user viewpoint (usually the server) to all other users/clients as in OpenGL VizServer software [7]. Visualization systems that are truly distributed and support generalized rendering (such as RAVE [54]) have problems with scaling in terms of the number of users and the size of the dataset. An extensive review of different visualization systems, together with a visualization taxonomy, has been presented in [56] and and by Brodlie and coworkers in [36, 37].

2.4.9 Interactivity Rate Measured in Frames/Second

Frame rate is also a term used in real-time computing and gives the number of consecutive images (or frames) the system or device can produce per second. If the frame rate of a real-time system is 60 hertz, the system reevaluates all necessary inputs and updates the necessary outputs 60 times per second under all circumstances [46]. The designed frame rates of real-time systems vary depending on the equipment. For a real-time system that is steering an oil tanker, a frame

[1]It is a 3D dataset represented as a group of 2D slice images acquired by a CT, MRI, or MicroCT scanner. Usually these are acquired in a regular pattern (e.g., one slice every millimeter).

rate of 1 Hz may be sufficient, while a rate of even 100 Hz may not be adequate for steering a guided missile. The designer must choose a frame rate appropriate to the application's requirements. There is no reason to show more frames per second than the viewer can perceive. The exact limit of human motion perception is still a matter of scientific debate, but it is generally agreed that there is an upper threshold on the frame rate after which people cannot appreciate any difference. Apteker et al. [32] concluded that to construct an interactive remote visualization system it must be capable of delivering between 10-15 frames/second to each user connected to the system.

2.5 Common Geometrical Rendering Techniques

Traditionally, to render an image one models a scene geometrically to some level of detail and then performs a simulation which calculates how the light reacts with that scene. The quality, realism, and rendering time of the resulting image is directly related to the modelling and simulation process.

1. Isosurfaces are created using a triangulation algorithm that generates a theoretical hull based on the data points given. Isosurfaces give an impression of a 3D object and use many thousands of triangles to create surfaces that interact with the light sources.

2. Volume rendering. During the rendering a 2D projection of a 3D dataset is displayed. Most commonly 3D data is in RGBA (red, green, blue, alpha) form, where alpha is the depth value. Each data point is a voxel with a RGBA value. Different techniques are used to determine the pixel RGB

value from all of the voxels that are projected to a particular coordinate.

3. Ray tracing. The main idea behind ray tracing algorithm is that physically correct images are composed by light and that light will usually come from a light source and bounce around as light rays (following a broken line path) in a scene before hitting our eyes or a camera. By being able to reproduce in computer simulation the path followed from a light source to our eye we would then be able to determine what our eye sees. Essentially, an imaginary image plane is created for which a ray is cast through each pixel. Then as the ray travels from the camera to the clipping area (end of volume) at regular intervals RGBA values are accumulatively calculated.

2.5.1 Limitations to Visualization

As in all areas of science and research there are currently limitations on our ability to visualize data. Computers give rise to two main limitations:

1. The hardware may be unable to process data fast enough.
2. The software may be unable to provide useful and efficient algorithms.

More complex modeling and simulation leads to higher quality images, however, they also lead to longer rendering times. Even with today's most advanced computer running today's most powerful rendering algorithms, it is still fairly easy to distinguish between a synthetic photograph of a scene and an actual photograph of that same scene.

2.6 Image-Based Rendering

In recent years, a new approach to computer graphics has been developed: Image-Based Rendering (IBR). Instead of simulating a scene using some approximate physical model, novel images are created though the process of reconstruction. Starting with a database of source images, an image is constructed by querying the database for information about the scene. IBR has the potential to provide a more realistic representation of very complex scenes at much faster rates than classical geometrical rendering. There are a variety of IBR algorithms, however, generally they can be classified into three main categories depending on the amount of *a priori* knowledge of the scene: rendering which requires some geometry of the scene, rendering with implicit geometry, and rendering with no geometry [93].

Other variations are classified according to how they constrain the view space of the viewer. These techniques are based on the characterization of the plenoptic function [28], which is a function that is equivalent to the complete holographic representation of the visual world. In general this is a 7-dimensional function,

$$P7(V_x, V_y, V_z, \theta, \varphi, \lambda, t)$$

representing the intensity of the light observed from every 3D position (V_x, V_y, V_z), at every possible orientation (θ, φ), for every wavelength λ, at every time t.

However, applying the 7D plenoptic function for real world scenes involves very large amounts of data, which makes the data acquisition impractical. As a result, various techniques have been used to simplify it by limiting the degrees of freedom. McMillan and Bishop [76] introduced a 5D plenoptic function by

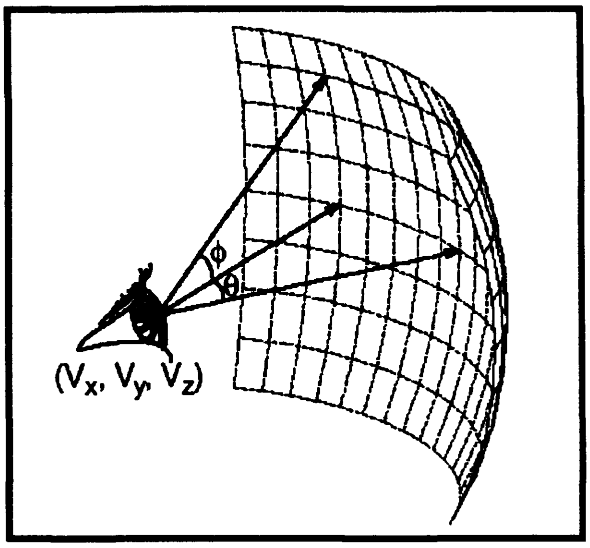


Figure 2.2: The plenoptic function.

removing the time and wavelength, resulting in

$$P5(V_x, V_y, V_z, \theta, \varphi)$$

Plenoptic modeling employs a 5D parameterization of the radiance in any 3D scene. However, radiance is constant along a given direction in free space (where there are no discontinuities due to intersections with objects). Hence radiance is constant along every direction outside the convex hull of a scene. This relaxes the dependence of the radiance on the position of the point of interest along the corresponding ray, and yields a representation of the plenoptic function by radiance along a four-dimensional set of light rays. The redundancy of the 5D representation is undesirable for two reasons: first, redundancy increases the size of the total dataset, and second, redundancy complicates the reconstruction of the radiance function from its samples.

Further restrictions could be applied to the plenoptic function. For example, constraining the viewer to a single point in space, resulting in the dimensionality being reduced to two. This is the principle used in environment mapping where the view of the environment from a fixed position is represented by a two-dimensional texture map [51].

2.6.1 QuickTime VR

If environment maps are used at a fixed position but at different orientations, then such a system approaches Apple's QuickTime VR system in which a 360-degree panoramic image, normally of a real scene, surrounds the viewer as a cylindrical image [40]. VR panoramas are panoramic images which surround the viewer within an environment (inside, looking out), yielding a sense of place. They can be "stitched" together from several normal photographs or two images taken with a circular fisheye lens, or captured with specialized panoramic cameras, or rendered from 3D-modeled scenes. There are two types of VR panoramas:

- Single row panoramas, with a single horizontal row of photographs.
- multi-row panoramas, with several rows of photographs taken at different tilt angles. VR panoramas are further divided into those that include the top and bottom, called cubic or spherical panoramas, and those that do not, which are usually called cylindrical panoramas.

Apple's QuickTime VR file format has two representations for panoramic nodes:

- Cylindrical (consisting of one 360deg image wrapped around the viewer.)

- Cubic (consisting of a cube of six 90 deg $\times$90 deg images surrounding the viewer) [18].

As the camera's orientation changes, the correct part of the image is retrieved, warped, and displayed. An alternative is to assume the view is constrained on a surface, for example the ground, as in the concentric mosaics approach [94].

2.6.2 Light Field Rendering

In the last decade image-based rendering has received increased attention due to the prospect of providing a more realistic representation of very complex scenes. The most well-known image-based rendering representation is light field rendering [70]. Research using similar approaches has also been carried out independently at Microsoft, and named Lumigraph [50]. Although, light field rendering is flexible in the nature of images it can present, it restricts the observer to a region of space free of occluders (i.e., outside the convex hull of the scene), and it is also not designed to cope with updates in the light field, i.e., the illumination is fixed. The amount of light travelling in a particular direction is measured in SI units in Watts per steradian per square metre, and usually is referred to as the *radiance*. With these constraints the radiance is constant along any ray, and the light field can be represented by a 4D dataset. All the rays emanating from a scene can be represented by their intersections with two arbitrary planes, referred to as the (u, v) and (s, t) planes. With this notation the 4D light field for a scene may be created by defining a 2D array of camera positions in the (u, v) plane [6], and generating images in which a pixel position is given by (s, t), as illustrated in Fig. 2.3. In this case the (s, t) plane is simply the focal plane of the camera. The images are sheared perspective views of the scene, and can be used to render a 2D output image of a 3D scene. By enclosing the scene in a cube, each face of which is a different (u, v) plane, any viewpoint in free space is supported. Alternatively a 4D light field for a real scene can also be generated using camera positions on the surface of a sphere using a moving camera gantry [9]. This results in a more complex mapping between an image pixel position and (u, v, s, t), and so this approach is not used in this thesis. In addition, the light field for a synthetic scene can be created using a modified ray tracing algorithm.

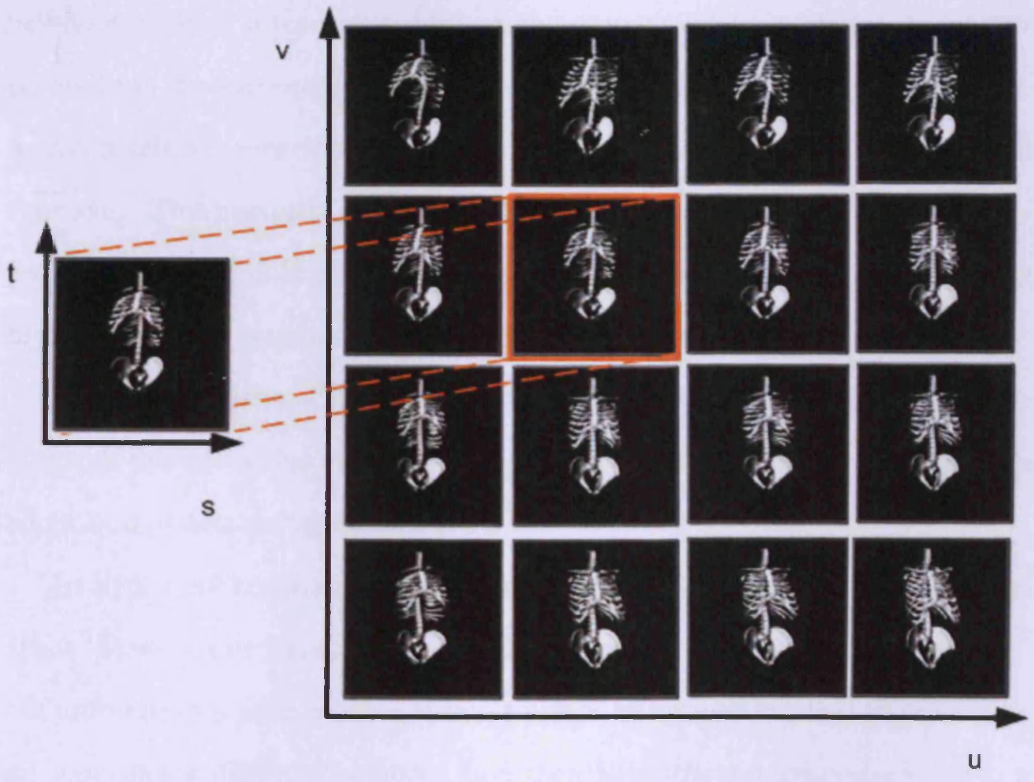


Figure 2.3: 4D Light Field representation in terms of an array of images taken at different positions in the (u, v) plane.

The rendering process involves the combining and resampling of the closest available images for the given viewpoint, and interpolation algorithms are applied to create the best estimate of the output image [88]. The simplest interpolation method is the *nearest-neighbor* algorithm where a pixel in the output image is computed as the value of the nearest mapped pixel in the source image. Since there is little calculation involved in this interpolation method it is the fastest. The next resampling method is *bilinear interpolation* where the destination pixel value is computed by combining the linear interpolation along two orthogonal axes. Although this interpolation method produces smoother results than the nearest-neighbor method, it tends to require more computation as it involves more data points from the surrounding neighborhood. In *quadrilinear interpolation* destination pixels are generated by combining linear interpolation with respect to all four axes. This approach gives high fidelity results compared with the other approaches, but its higher computation time makes it an appropriate choice only for highly interactive systems running on high-end devices.

Another extension of the light field approach uses a video camera [104, 103], although this gives rise to the problem of high storage volumes and requires a sophisticated data management scheme.

In light field rendering the light field generated consumes a lot of storage space. Many researchers have presented solutions to these limitations separately, but unfortunately none of these systems offers an optimal solution that works in all applications. Different solutions have their strengths and weaknesses, and as a result the best solution is application-specific. Accordingly, light field rendering is only now slowly emerging from the research laboratories to be used in real-life applications [83]. Thus, this thesis focuses on building a collaborative visualization environment for distributed users that involves complex datasets, but with static geometry and static illumination.

2.7 Image-Based Rendering in Distributed Environments

Image-based rendering is not currently widely used in the area of remote visualization. An exception is the work of Jin et al.[64] which uses a shared network storage infrastructure based on what they call Logistical Networking [33, 85]. The main idea of logistical networking is that placing multiple network buffers close to the clients shortens the latency on each route of data access. Although this approach provides an improvement in transmission bandwidth and latency, it requires network buffers to be pre-installed which requires technical skills and/or special software.

Image-based rendering is also used to accelerate remote visualization. Yoon and Neumann [107] adapt an image-based rendering acceleration method to the direct rendering of compressed images. The approach is attractive for remote rendering applications where a client system may be a relatively low-performance machine and limited network bandwidth makes transmission of large 3D data sets impractical. The efficiency of the server and client generally increases with scene complexity or data size since the rendering time is predominantly a function of image size. This technique uses stored images to accelerate the rendering and compression of new synthetic scene images. Visapult [34] also employs methods related to image-based rendering to accelerate remote visualization.

2.8 Transmission Models for Distributed Graphics

Rendering time is only part of the performance problem as the underlying network also imposes constraints. Even with the improvements in rendering time gained by using Light Field rendering [29], the overall performance is still constrained by the underlying network transmission model. In general, three distinct models for distributed graphics are in use today (see Fig. 2.4).

2.8.1 Local Rendering

In the local rendering scenario, the entire visualization pipeline is performed by each individual client machine which uses only local device resources. A copy of the geometric dataset is replicated and stored locally for access by each client. Since the rendering task is conducted by participating clients, the frame rate is determined by the graphics hardware of these clients. Such an approach is feasible when using the rendering capabilities that exist on high-end clients, or in cases where the dataset is limited in size. On the other hand, given a combination of low-end graphics hardware clients, low bandwidth networks, or large dataset size, the approach will lead to extensive download time and high storage costs, with potentially non-interactive rendering rates. Moreover, situations exists in which sharing the data set with remote clients is not advisable for security reasons. Most existing systems, such as DIVE[58], use this technique.

2.8.2 Remote Rendering

In the remote rendering scenario, the visualization clients interact with 3D objects and their corresponding requests are sent over the network to the server. Remote hardware is used to render the scene. The rendering takes place on the server and the resulting images are calculated and retransmitted back to the client for viewing. Remote rendering allows the sharing of expensive hardware via the network through a client computer.

In recent years there have been several different approaches to implement this partitioning strategy. Virtual Network Computing (VNC) [89] uses a client-server model. The custom client viewer on the local machine interprets any events and sends them to the VNC server on the remote machine. The server detects screen updates, processes them, and sends a whole screen update to the client. Other approaches to remote rendering are presented in OpenGL VizServer [7, 19] which uses a client-server model for remote delivery of hardware-accelerated rendering without needing to access the entire desktop.

The advantage of a remote rendering system is that there is no requirement for clients to have high-powered rendering hardware in order to take part in the visualization, hence even a thin client running on a PDA can join the visualization process. However, remote rendering has the disadvantage of high network traffic as each user interaction incurs the cost of transmission of the request and images. As a result such an approach is not scalable to support a large number of concurrent users. A comparison of local and remote rendering for a number of key criteria is given in Table 2.1.

2.8.3 An intermediate solution

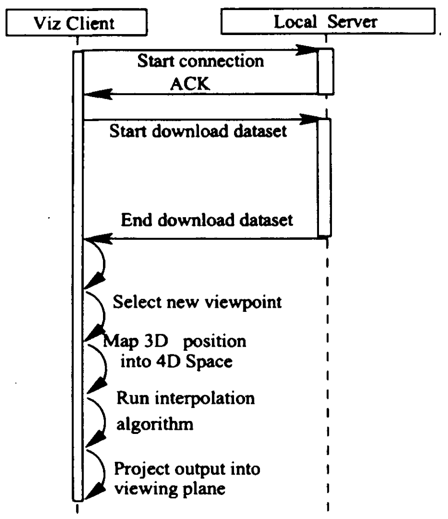
It is possible to combine both local and remote rendering approaches. The number of papers that discuss such an approach is still small (for example, Distributed GL

Criteria	**Local Rendering**	**Remote Rendering**
Network Traffic	Low	High
CPU capability	High	Low
Complexity/Admininstration	Less complex	High complexity
Storage costs	High	Low

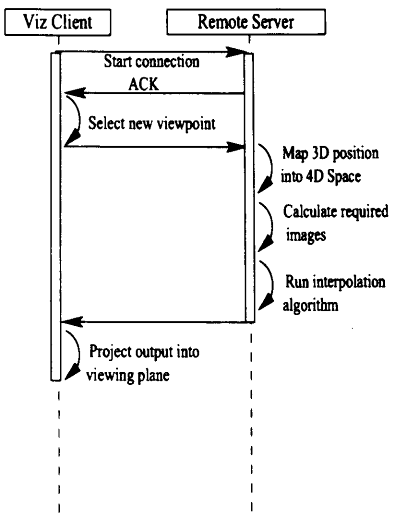
Table 2.1: Comparison of local and remote rendering.

[41, 92, 101]). Such solutions are particularly applicable to 3D geometrical datasets, where a simplified version of the geometrical dataset is sent to the clients and updated each time the user changes the viewing parameters. Whilst present systems using this combined approach show some improvements in performance, for large complex datasets such a hybrid approach necessarily inherits the drawbacks associated with both the local and remote rendering approaches outlined above.

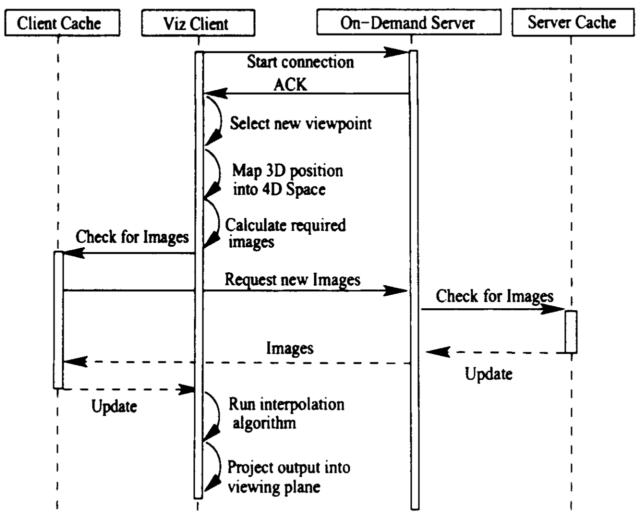
It is clear that each of the existing transmission models has several drawbacks, primarily associated with either the transmission of datasets and/or supporting a large number of concurrent users which are actively navigating the dataset and hence generating significant network traffic.



(a) local rendering scenario


Viz Client
Remote Server
Start connection
ACK
Select new viewpoint
Map 3D position into 4D Space
Calculate required images
Run interpolation algorithm
Project output into viewing plane

(b) remote rendering scenario



(c) on-demand rendering scenario

Figure 2.4: Sequence diagrams for different scenarios.

2.9 Chapter Summary

This chapter has presented an overview of the basic concepts and terminology used throughout the thesis. In addition, it has surveyed the current key state-of-the-art technologies associated with the three areas we are aiming to bridge between: image-based rendering, distributed visualization systems, and transmission models for distributed graphics.

Chapter 3

Light Field Rendering for Remote Visualization

Overview

This chapter describes the design and architecture of the remote visualization system that is the main focus of this thesis. It also demonstrates the use of light field rendering in remote visualization systems with a real application scenario.

Initially we introduce the three key components of the system. Then we give examples of the application domains. Following this the implementation of the three possible transmission models introduced in Chapter 2 is presented, along with a discussion of the details of creating a practical software solution for a remote interactive visualization system. Finally, in Section 3.3, we further focus on discussing the on-demand approach, where a partial dataset is downloaded to clients.

3.1 Light Field Rendering System

We propose a generic distributed visualization system that presents a solution which is independent of 3D model complexity and relies only on the final output image resolution. Our system provides the basis for a distributed collaborative environment, where multiple concurrent users visualize a remotely-stored 4D dataset. The developed system applies a multi-user client-server model and takes advantage of the availability of low-cost network equipment to provide an inexpensive visualization solution. There are three core phases that comprise our system architecture, as shown in Fig. 3.1.

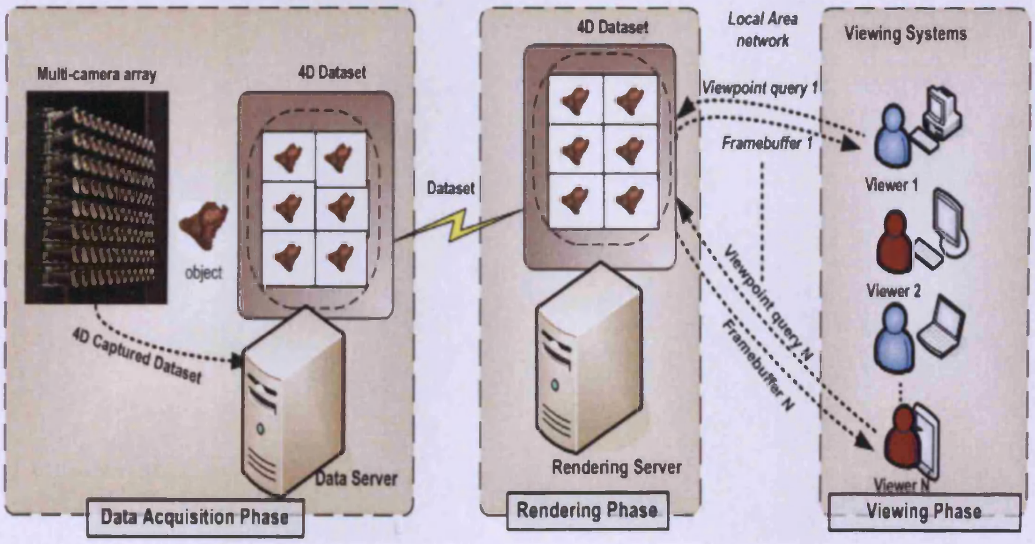


Figure 3.1: System Overview. In the Data Acquisition phase a 4D dataset is created. The Rendering phase performs interpolation across multiple images. In the Viewing phase viewpoint queries are sent and the received output frame buffers are loaded for display.

3.1.1 Data Acquisition Phase

In the data acquisition phase the 4D light field dataset is generated for a targeted object. The 4D light field dataset is generated in a pre-processing step using either a multi-array camera [6] or gantry [9] for real objects as shown in Fig. 3.2, or it is created for a synthetic object using a modified ray tracing algorithm, as shown in Fig. 3.3. The 4D data should be captured in free space, i.e., in a region free of occluders.

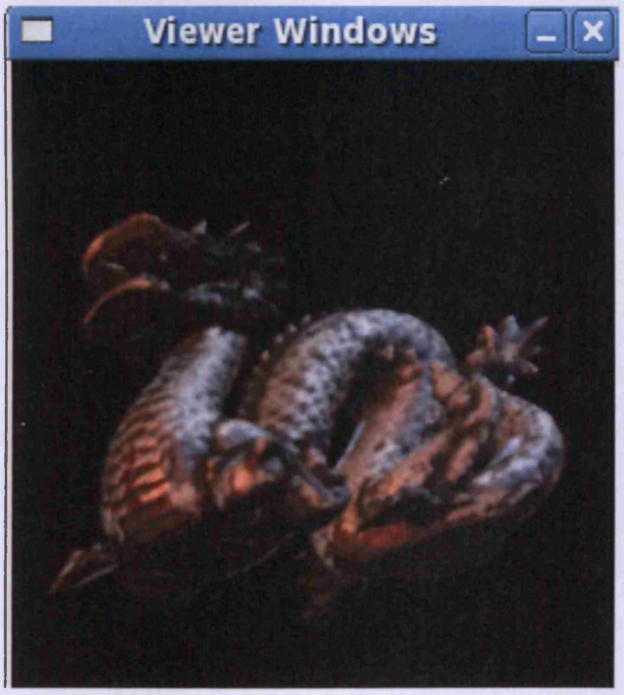


Figure 3.2: Real object viewer.

This includes the following two situations [70]:

1. Most geometric models are bounded. In this case free space is the region outside the convex hull of the object, and hence all views of an object from outside its convex hull may be generated from a 4D light field.

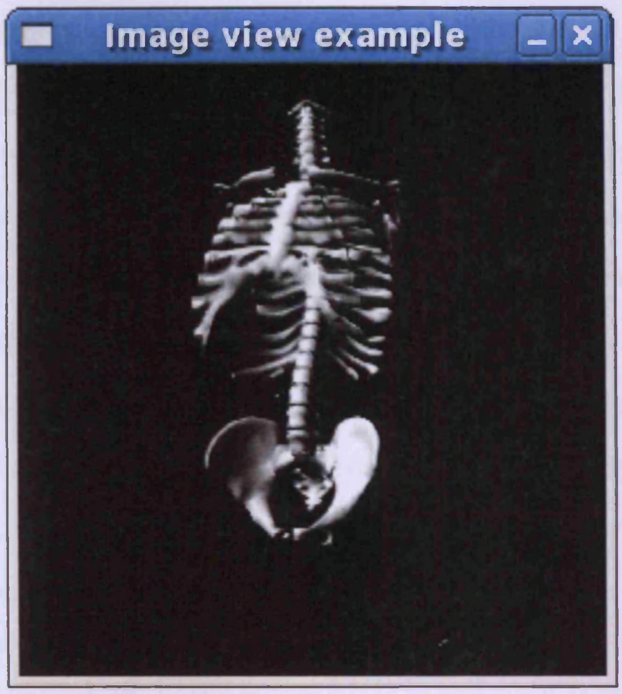


Figure 3.3: Synthesized object viewer.

2. If we are moving through an architectural model or an outdoor scene we are usually moving through a region of free space; therefore, any view from inside this region, of objects outside the region, may be generated.

3.1.2 Rendering Phase

The second phase involves the rendering process. During this phase, the nearest images to the requested view are interpolated to produce the desired view. In order to be able to view a 3D model on a 2D display the rendering process must create a 2D output image based on the description of the viewers position in 3D space. The light field is created by sampling from a 2D array of camera positions, represented by (u, v), and a pixel position (s, t) within the selected image. The rendering in this case simply combines and resamples the closest available images. Interpolation algorithms are applied to create the best estimate of the output image [88].The *nearest-neighbor interpolation* is the most basic and requires the least processing time of all the interpolation algorithms because it only considers one pixel - the closest one to the interpolated point. The next method is the *bilinear interpolation* combines the linear interpolation along two orthogonal axes around the interpolation point. This results in much smoother looking output than *nearest-neighbor interpolation*, but it requires a higher computation time as it performs more number of operations. *Quadrilinear interpolation* produces noticeably sharper output than the previous two methods, but it consumes longer computation time which limits its usage to only a high-end processing hardware.

3.1.3 Viewing phase

The final phase is when 3D viewing takes place. In the viewing scenario visualization participants (clients) run an instance of the viewing process in which their independent 3D viewing positions are created locally and processed remotely in the rendering server. Each visualization client or viewer is working independently. Figure 3.4 shows the concurrent interaction between the server process and the different viewer processes. As the user runs the viewer process it will connect to the server process. After the user selects a new viewing parameter set, a new request is sent to the server where the rendering request is processed. The client will then receive the updated framebuffer and a new view will be reloaded and displayed. The server executes the viewer's viewing requests based on their arrival time. Clients send their viewing requests separately to the rendering server. The rendering server processes the requested views concurrently and sends the resulting framebuffers to the clients, as shown in 3.4.

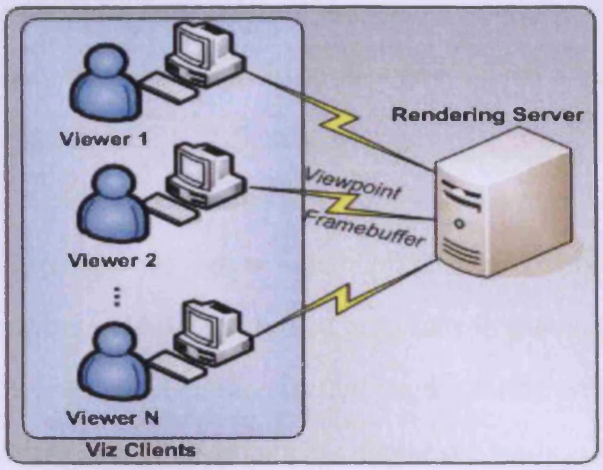


Figure 3.4: Independent viewers.

As already indicated, light field rendering is a pixel-based algorithm, and the results of tests on real data are similar to the results of experiments on synthesized objects. While the pre-generation of the data are different in each case, the general behaviour of the algorithms is similar.

3.2 Real Life Application Domains

The LF system developed could be used for any static or time-dependent distributed visualization application. Also the light field approach is categorized as a type of computational imaging which refers to any image formation method that involves a digital computer. As a result, listing all applications of light field rendering would require surveying all uses of computational imaging – in art, science, engineering, and medicine. In computer graphics, some selected applications are:

- **Medical domain**. One application in the medical domain is where trainee physicians and surgeons need to develop a deep and detailed understanding of the structure and variation of human anatomy. Traditionally this has been acquired through hands-on experience with a cadaver. In recent years a number of factors, including strengthening health and safety requirements, diversification of curricula, geographical distribution of learners (from undergraduate through to senior practitioners), and a need to work with multiple examples to experience variations in anatomy, have made this increasingly impractical and ineffective. In the future we anticipate extending the system's usage to encompass diagnosis, medical training, and surgery planning using real-time visualization in a distributed environment.

- **Virtual reality**. The Virtual Light Field project [78, 79, 96] is a system that allows real-time global illumination within a virtual reality system. Global illumination means that the virtual light that is distributed through the virtual world gives rise to similar effects to the real world. For example, different types of shadow, and reflections. There have been computer graphics techniques to achieve global illumination for years, but achieving this within an interactive virtual reality environment is a novel path for virtual reality. This could change how people respond to virtual environments, as well as making possible opportunities for new forms of interaction.

- **3D television and 3D displays**. By presenting a light field using technology that maps each sample to the appropriate ray in the physical space, one obtains an autostereoscopic visual effect akin to viewing the original scene. Non-digital technologies for doing this include integral photography, parallax panoramagrams, and holography. Digital technologies include placing an array of lenslets over a high-resolution display screen, or projecting the imagery onto an array of lenslets using an array of video projectors. If the latter is combined with an array of video cameras, one can capture and display a time-varying light field. This essentially constitutes a 3D television system [63, 74]. Image generation and pre-distortion of synthetic imagery for holographic stereograms is one of the earliest examples of computed light fields.

- **Face recognition**. Combining the light field and the Lambertain reflectance model gives an approach that integrates the handling of pose and illumination variation. This treatment of the light field results in an identity signature that is invariant to illumination and pose [108].

- **Geology and engineering**. The light field approach is useful when viewing a large, complex model (such as an oil-drilling platform), or any large dataset that would otherwise overwhelm a single graphics processor.

- **Glare reduction**. Multiple scattering of light inside the camera body and lens optics reduces image contrast. While glare has been analyzed in 2D image space [99], it is useful to identify it as a 4D ray-space phenomenon. By statistically analyzing the ray-space inside a camera, one can classify and remove glare artifacts. In ray-space, glare behaves as high frequency noise and can be reduced by outlier rejection. Such analysis can be performed by capturing the light field inside the camera, but it results in the loss of spatial resolution. Uniform and non-uniform ray sampling could be used to reduce glare without significantly compromising image resolution [87].

3.3 On-Demand Transmission Model Using Image-Based Rendering for Remote Visualization

In addition to reducing the overall computational cost of rendering a scene, the discrete nature of the light field dataset, in which each image has a distinct representation, maps well to a hybrid solution which can overcome the performance drawbacks identified in 2.8. We use pre-fetching and streaming mechanisms to efficiently and effectively transmit visualization data to remote users/clients. Instead of downloading a complete copy of the light field dataset to each client, or remotely sending a single rendered light field view back from a central server to the user each time a user updates their viewing parameters, our strategy combines both approaches. In response to a viewing query, initially the client cache is checked for the required images(similar to the caching approach of Sisneros et al. [95]). If an image is not available it is retrieved from the server, and interpolated. The client cache is updated by storing the most recently used images. User navigation behaviour can be studied for random or orderly scenarios. For orderly exploration, in which user navigation follows a coherent pattern, it is possible to predictively pre-fetch the desired images, and this will further help to hide latency and improve client-side performance. Applying such a method will better utilize the rendering capability of the clients and minimize the server load, which will reduce the overall network traffic. Furthermore, the excessive use of storage is avoided by using the on-demand download of the required partial datasets. The design provides a novel combination of light field techniques and a transmission model which provides a general-purpose interactive solution for distributed collaborative visualization.

3.3.1 System Architecture

The developed system provides a distributed collaborative environment in which multiple concurrent users visualize a remotely stored 4D dataset. A multi-user client-server model is used, taking advantage of the availability of commodity-off-the-shelf (COTS) computer hardware, software and network equipment. The main reason that our system is implemented only in software is that we targeted a generic system without the need for any special hardware. Furthermore, we aim to support for broad range of client-side platforms, ranging from a high performance desktop to a PDA for field scientists. There are two main components that comprise our system architecture, as shown in Fig. 3.5.

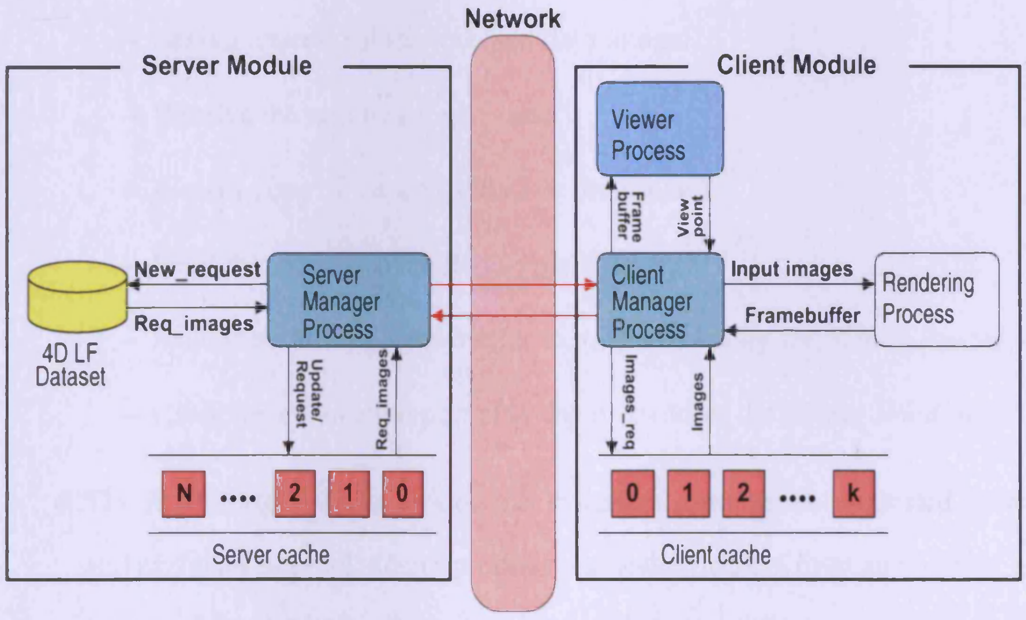


Figure 3.5: The architecture of an on-demand transmission model.

3.3.2 Client Module

The basic configuration of the client side of the system has the following components:

- The *Client Manager Process* is responsible for dealing with external network connections and internal process communication within the client. Its role can be summarized as follow:
  - Send user viewing parameters to the *Rendering Process.*
  - Check if the client cache has the required images.
  - Establish a connection port to the server.
  - Send a request for the required data image.
  - Receive the required image data.
  - Place a copy of the image data in the cache.
  - Send the images to the *Rendering Process.*
  - Send the resulting framebuffer to be displayed by the *Viewer Process.*
  - Close the connection port after the user closes the *Viewer Window.*
- The *Rendering Process* locates the images nearest to the requested view and performs interpolation to produce the desired view. Given an observer's viewpoint, each pixel in the output rendered image of the scene corresponds to a ray traveling through free space from the scene to the observer. Such a ray intersects the camera plane (u, v) and image plane (s, t). Interpolation on the light field dataset is used to estimate the light field (and hence the pixel value in the rendered image) for the ray. In our implementation

we have used bilinear interpolation in (u, v) and nearest-neighbour interpolation in (s, t) as this scheme provides a good trade-off between image quality and computational complexity, compared with other interpolation techniques, such as nearest-neighbour interpolation in both (u, v) and (s, t), and quadrilinear interpolation. With this type of bilinear interpolation each pixel in the output image is a weighted average of pixel values taken from images corresponding to the four nearest points to the intersection of the ray with the (u, v) plane.

- The *Viewer Process* is where 3D viewing takes place. As the user interacts with the Viewer Process, a corresponding set of viewing parameters is sent to the Client Manager Process for rendering. The resulting image is received from the Rendering Process, mapped onto the projection plane, and displayed using the QImage class of the Qt library [8].

- The *Client Cache* is filled with images based on recent viewing parameters within the 3D environment. This can be used to build a resource-aware layer to reduce the load on the server in order to increase scalability in terms of the number of participating clients. A further enhancement is to apply a prediction algorithm to pre-load images by predicting the navigation path of a user.

3.3.3 Cache Replacement

Theoretically we could assume that the client and server caches can store infinite items. Practically, given the hardware limitations, we assign an upper limit to the cache capacity. In this case when the cache is full, a discard process must be applied. We applied the Least Recently Used (LRU) algorithm. We assign the threshold values in the pre-experiment setup. This threshold may be set based on the memory resources available. During our experiments we have set this value to be ten dataset images. The system searches the cache for a required image, and if it is not found the client replaces the least recently used image.

3.3.4 Network

Network communication for remote and on-demand scenarios is built on connection-based TCP/IP sockets, which guarantee message delivery and order, while the cURL library is used for downloading the datasets for the local scenario [14].

3.3.5 Server Module

The server has three main components:

- The *Server Manager Process* handles new client connections using threads. When a new view request is sent to the server, the request is parsed, the viewing parameters are extracted, and the cache is checked. The *Server* executes the viewer's requests based on their arrival time (see Fig. 3.6). The client-server interaction model for our system represents collaboration in space and time, since participants could be located in different places at different times. Each individual viewer can join or leave the visualization at any time. Each visualization client or viewer independently controls their viewpoint requests. Each time a new request is received it checks the availability of the related images in the cache and if not present fetches them from the dataset and copies them to the cache for possible future requests.

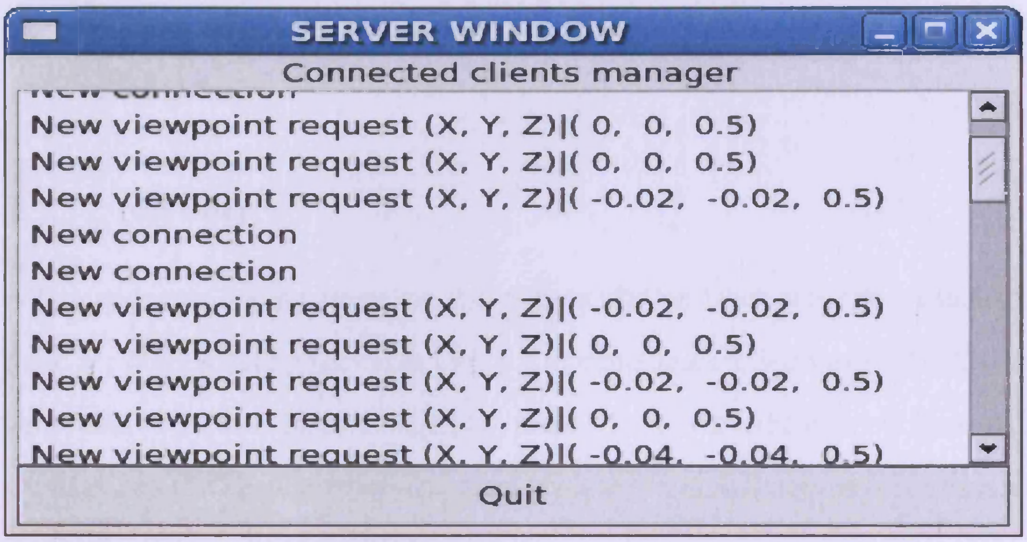
The role of the Server Manager Process can be summarized as follow:

- Accept a new connection with the joining visualization clients.
- Receive clients' data requests.
- Check Server cache for requested images, and if the are found send them to the client.
- Upload any required images not found in the cache from the light field dataset and send them to the client.
- Update the Server Cache with the recently requested data images.

- The *Light Field Dataset* stores the 4D light field images.
- The *Server Cache* stores in memory the images requested recently by the visualization clients. In collaborative visualization environments the users normally explore a section of the 3D scene in which they have a common interest, generating the same viewpoint for all clients.

3.3.6 Implementation

The *Server Manager Process* handles new client connections using threads. When a new view request is sent to the server, the request is parsed, the viewing parameters are extracted, and the cache is checked. The *Server Manager Process* executes the viewers' requests based on their arrival time. Each individual viewer can join or leave the visualization at any time, and each visualization client or viewer works independently. The client module's role will vary in accordance with the following three scenarios:



```
New viewpoint request (X, Y, Z)|( 0,  0,  0.5)
New viewpoint request (X, Y, Z)|( 0,  0,  0.5)
New viewpoint request (X, Y, Z)|( -0.02,  -0.02,  0.5)
New connection
New connection
New viewpoint request (X, Y, Z)|( -0.02,  -0.02,  0.5)
New viewpoint request (X, Y, Z)|( 0,  0,  0.5)
New viewpoint request (X, Y, Z)|( -0.02,  -0.02,  0.5)
New viewpoint request (X, Y, Z)|( 0,  0,  0.5)
New viewpoint request (X, Y, Z)|( -0.04,  -0.04,  0.5)
```

Figure 3.6: Server monitoring screen.

- In the local rendering scenario, the clients pre-fetch (download) the whole dataset and proceed with the rendering procedure.
- In the remote rendering scenario, as the user runs the viewer process it will connect to the server process, and each time the user selects new viewing parameters, a corresponding request is sent to the server where the rendering process is performed, and the resulting framebuffer is then sent back to the client. A new view will be loaded and displayed.
- In the on-demand rendering scenario, as the user interacts with the *Viewer Process*, a corresponding viewing parameter set is sent to the *Client Manager Process* for rendering. The *Client Manager Process* determines the images needed for rendering that user viewpoint, and if all the images are located in the cache then the *Rendering Process* performs interpolation to produce the desired view. Otherwise, the missing images will be requested from the server. The resulting image is received from the *Rendering Process*

and mapped onto the projection plane. The system architecture is illustrated in Fig. 3.5.

3.3.7 Network

All components communicate via the communication layer network. Although there are other existing packages used for communication, such as CORBA and Java Remote Method Invocation (RMI), these add more overhead to our systems. CORBA and RMI both provide a protocol layer above TCP. Since the performance is still the most important criteria for selecting the communication protocol the system was implemented using standard TCP/IP.

3.3.7.1 TCP

The Transmission Control Protocol (TCP) is one of the core protocols of the Internet Protocol Suite. TCP is one of the two original components of the suite, complementing the Internet Protocol (IP) and therefore the entire suite is commonly referred to as TCP/IP. TCP provides the service of exchanging data reliably directly between two network hosts, whereas IP handles addressing and routing messages across one or more networks. In particular, TCP provides reliable, ordered delivery of a stream of bytes from a program on one computer to another program on another computer. TCP is the protocol that major Internet applications rely on, such as the World Wide Web, e-mail, and file transfer. Other applications, that do not require a reliable data stream service, use a sister protocol, the User Datagram Protocol (UDP). This provides a datagram service, which emphasizes reduced latency over reliability. Network communication for remote and on-demand scenarios is built on connection-based TCP/IP sockets, which guarantee message delivery and order.

3.3.7.2 cURL

cURL stand for 'Client for URLs', although originally it was an abbreviation for "Client URL Request Library". The cURL project contains two products:

- A free and easy-to-use client-side URL transfer library, supporting FTP, FTPS, HTTP, TELNET, file transfer resume, http proxy tunneling, and more.
- The libcurl library is highly portable, and builds and works identically on numerous platforms. libcurl is free, thread-safe, IPv6 compatible, feature-rich, well-supported and fast.

Since cURL uses libcurl it supports a range of common Internet protocols, currently including HTTP, HTTPS, FTP, TELNET and FILE. libcurl is a reliable and portable library which provides an easy interface to a range of common Internet protocols. libcurl is free open-source software.

3.3.8 Running and Synchronization

Experimenting with remotely-accessed graphics resources can be problematic. The X11 protocol [90] is a network-transparent client-server system for the display of graphics on remote machines. This is contrary to the design of systems such as Microsoft Windows, which have not been designed with remote abilities in mind. The display that is being used runs the X server, and applications that utilize such a display are X clients. The X server maps the abilities of the operating system and graphics hardware that it is running on into the X protocol which is accessed by a client application through use of the Xlib library.

In order to actually launch a real time remote visualization session, we run a configuration shell script on the server machine.

The processing engines are hosted on a collection of 32 open-access Linux workstations representing the visualization clients. The client input represents the data set variability and different user viewpoints and allows the task sequence to execute in parallel. Before the processing is carried out an SSH connection is established to each workstation. This causes each client to establish a connection with the server machine *alsaidi.cs.cf.ac.uk* , and initiates a *Client process* on each of the workstations *cslx01,cslx02,..csl32*. Part of the script is as follows:

```
#!/bin/bash
rsh 1
xhost +
rsh -n -l scmaal0 cslx01 "DISPLAY=alsaidi.cs.cf.ac.uk:0.0;
export DISPLAY;/home/scmaal0/Desktop/client/client"&&
.
.
.
rsh -n -l scmaal0 cslx32 "DISPLAY=alsaidi.cs.cf.ac.uk:0.0;
export DISPLAY;/home/scmaal0/Desktop/client/client"
```

Criteria	Java/Swing	C++/Qt
Updated	Developed in late 90's and not updated recently	Active development
Availability	Available in Java installation	Library files need to be added
2D/3D functions	Not available in Java	Direct access to 2D and GL drawing functions
Memory-efficiency	Higher memory requirement	Lower memory requirement
Runtime-efficiency	Tends to run longer than the equivalent C/C++ code	Shorter running time than Java

Table 3.1: Java/Swing compared with C++/Qt.

3.4 The Programming Language

The programming language choice is a significant aspect as it has a considerable impact on overall performance. A common choice is Java, but this has too much overhead in terms of creating the Java Virtual Machine [42]. Furthermore, Prechelt in his empirical comparison [86] arrives at the conclusion that "a Java program must be expected to run at least 1.22 times as long as a C/C++ program". Also he states that on average, and with a confidence of 80%, that Java programs consume at least 32 MB (or 297%) more memory than the corresponding C/C++ programs.

While Java may be preferable to C++ since it is platform-independent language, Qt/C++ is a cross-platform toolkit that runs on many platforms. By using Qt [8] we can now add to the performance and efficacy features of C++, platform-independence and rich functionality.

3.4.1 Qt/C++

Qt is a cross-platform application and UI framework. Using Qt, it is possible to write web-enabled applications once and deploy them across desktop, mobile and embedded operating systems without rewriting the source code. Qt provides all the functionality needed to develop advanced GUI applications on desktop and embedded platforms. Qt uses the native graphics APIs of each platform it supports, taking full advantage of system resources and ensuring that applications have a native look and feel.

Features of Qt include:

- Intuitive C++ class library.
- Portability across desktop and embedded operating systems.
- Integrated development tools with cross-platform IDE.
- High runtime performance and small footprint on embedded systems.
- Auto-scaling, font-, language- and screen orientation-aware layout engine.
- Support for anti-aliasing, vector deformation and scalable vector graphics (SVG).
- Complete UI customizability with style API and widget stylesheets.
- Support for hardware accelerated graphics and multiple displays on embedded systems.
- Complete set of controls (widgets) from buttons and dialogs to tree views and tables.

3.5 Chapter Summary

In this chapter we have described the software architecture for using an on-demand transmission model using light field rendering within a distributed collaborative environment. The system makes use of commodity networking, hardware, and software for distributed visualization. We have discussed the existing local and remote transmission models and compared them with our on-demand model. Furthermore, we have discussed the implementation choices for the implementation of the three models introduced in the last two chapters. This began with an explanation of the different phases of the system and how they interact with each other. Then we explained how we automate the running and synchronization of the system. Finally we presented the different libraries that have been used.

Our results in the next chapter show that a light field rendering system enables constant framerates independent of the scene complexity. This is in contrast with geometry-based visualization, where the framerate depends on the complexity of the scene for the chosen viewpoint. The comparative efficiency of the light field rendering system increases as the scene complexity increases. The applicability of the light field rendering system covers a large number of application areas, where interactive collaboration could enhance and accelerate navigation and discovery.

Our system currently provides a generic on-demand remote rendering solution for users to access visualization services using computationally inexpensive light field rendering, allowing low-end devices, such as PDAs and mobile phones, to interactively explore 3D objects.

The on-demand approach shows stable performance as the number of clients increases because the load on the server and the network traffic are reduced. The main performance bottleneck arises from the limited transfer rate across the network, which can be improved with the availability of higher performance networks.

Chapter 4

Experimental Results and Discussion

Overview

In this chapter a performance study of the system described in Chapter 3 is presented in order to understand different factors that influence the performance, to help determine areas requiring further attention, and clarify where the focus for major future work should directed.

Performance experiments have been conducted to directly compare the on-demand model with the local and remote models outlined in Section 3.3.6. In the following we describe the testbed setup and present the key performance results.

4.1 Objectives

In general, there has been little research addressing the fundamental issue of analyzing the effects of different factors on different 3D rendering scenarios. The problem can be quite data set specific, and it is difficult to draw any general conclusions that fit the huge variety of datasets and rendering approaches. However, image-based rendering provides a constant rendering time that does not depend on scene complexity, but only the final output dimensions. Factors worthwhile investigating include the number of visualization clients, the dataset size, and interpolation schemes. The impact of these factors is important for both theoretical and practical purposes.

The experiments in this chapter explore a prototype interactive remote visualization system using image-based rendering which provides a highly interactive remote 3D visualization solution for large numbers of users. The objectives of the experiments include:

- To permit many users to access an interactive 3D remote visualization system based on commodity-off-the-shelf components.
- To investigate the effectiveness of the local, remote and on-demand transmission models in term of frames per second.
- To provide users options with different dataset sizes that are appropriate for their requirements and hardware. The aim is to measure the achieved interactive performance for each dataset size.

4.2 Testbed Environment

Our testbed environment uses commodity hardware consisting of 32 standard PCs running Linux Fedora 7. Each node has a 2.8 GHz processor, 2 GB of RAM, and an nVidia Corporation G70 graphics card (GeForce 7800 GS) with 256Mb of RAM. The server has 2 GB of RAM, and is a 64-bit Intel Pentium Dual CPU machine with a 3.4 GHz processor and an ATI Radeon X1300 graphics card. Communication is over a 100 Mb/s local Ethernet network.

The datasets used represent different views of part of a human skeleton. To determine the performance of the system we used four different light field dataset dimensions. These datasets all have a (u, v) array of 16×16 camera positions so there are 256 images for each dataset, and for each position we have image sizes of 128×128, 256×256, 512×512, and 640×640 pixels. These image sizes, summarized in Table 4.1, reflect the range of image sizes commonly supported by current camera, scanners and PDAs.

Dataset	**One Image**	**Whole Dataset**
128×128	131072	33554432
256×256	524288	134217728
512×512	2097152	536870912
640×640	3276800	838860800

Table 4.1: Dataset size in bits.

4.2.1 Experimental Procedure

The navigation path (a sequence of cursor positions) is recorded from human interaction with the visualized object to emulate a real-life navigation experience, and for homogeneity we make sure that these parameters are used in all experiments for quantitative results. Such a navigation path is used in ordinary computer viewpoint animation, and is in fact a form of constrained navigation consisting of a linear time sequence of camera models [59].

The system is cross-platform as Qt provides a high level of abstraction [8]. All timings were performed using the Qt time functions (*t.start()* and *t.elapsed()*) to measure time in milliseconds.

4.3 Effect Of Different Interpolation Schemes

As outlined in Section 2.6, given a particular viewpoint the corresponding 2D output image of the scene is derived from the light field samples at a set of points in (u, v, s, t). These sampled points are represented as a set of equally-spaced images in the (u, v) plane with pixel positions in an image denoted by (s, t). Suppose that a ray from the scene to the observer intersects the (u, v) plane at (u_*, v_*) and the (s, t) plane at (s_*, t_*). Interpolation is required to determine the light field at (u_*, v_*, s_*, t_*) from the set of images in the (u, v) plane.

Let the image at location (u_j, v_k) in the (u, v) grid be denoted by $L_{j,k}$. The grid spacing is Δu in the u direction and Δv in the v direction, and the pixels are Δs and Δt apart in the s and t directions. This thesis considers four possible interpolation schemes with our remote visualization system:

(a) No interpolation. Let (u_m, v_n) be the point nearest to (u_*, v_*) in the (u, v) grid, and let (s_p, t_q) be the nearest pixel to (s_*, t_*). Then the interpolated light field for this ray is:

$$\hat{L}(u_*, v_*, s_*, t_*) = L_{m,n,p,q} \tag{4.1}$$

where $L_{m,n,p,q} \equiv L_{m,n}(s_p, t_q)$. Thus, this is a nearest-neighbour interpolation scheme in both $(u, v,)$ and (s, t).

(b) Bilinear in (u, v), and nearest-neighbour in (s, t). Let $u_m \leq u_* \leq u_{m+1}$ and $v_n \leq v_* \leq v_{n+1}$, and let (s_p, t_q) be the nearest pixel to (s_*, t_*). Then the interpolated light field for this ray is:

$$\begin{aligned} \hat{L}(u_*, v_*, s_*, t_*) &= \mu_m^1 \nu_n^1 L_{m,n,p,q} + \mu_m^0 \nu_n^1 L_{m+1,n,p,q} \\ &\quad + \mu_m^1 \nu_n^0 L_{m,n+1,p,q} + \mu_m^0 \nu_n^0 L_{m+1,n+1,p,q} \end{aligned} \tag{4.2}$$

where

$$\mu_m^i = \frac{(-1)^{i+1}}{\Delta u}(u_{m+i} - u_*), \qquad \nu_n^i = \frac{(-1)^{i+1}}{\Delta v}(v_{n+i} - v_*)$$

(c) Nearest-neighbour in (u, v), and bilinear in (s, t). Let (u_m, v_n) be the point nearest to (u_*, v_*) in the (u, v) grid, and let $s_p \leq s_* \leq s_{p+1}$ and $t_q \leq t_* \leq t_{q+1}$. Then the interpolated light field for this ray is:

$$\begin{aligned} \hat{L}(u_*, v_*, s_*, t_*) &= \sigma_p^1 \tau_q^1 L_{m,n,p,q} + \sigma_p^0 \tau_q^1 L_{m,n,p+1,q} \\ &\quad + \sigma_p^1 \tau_q^0 L_{m,n,p,q+1} + \sigma_p^0 \tau_q^0 L_{m,n,p+1,q+1} \end{aligned} \tag{4.3}$$

where

$$\sigma_p^i = \frac{(-1)^{i+1}}{\Delta s}(s_{p+i} - s_*), \qquad \tau_q^i = \frac{(-1)^{i+1}}{\Delta t}(t_{q+i} - t_*)$$

(d) Quadrilinear in (u, v, s, t). Let $u_m \leq u_* \leq u_{m+1}$ and $v_n \leq v_* \leq v_{n+1})$, and let $s_p \leq s_* \leq s_{p+1}$ and $t_q \leq t_* \leq t_{q+1}$. Then the interpolated light field for this ray is:

$$\begin{aligned}\hat{L}(u_*, v_*, s_*, t_*) &= \mu_m^1 \nu_n^1 \sigma_p^1 \tau_q^1 L_{m,n,p,q} + \mu_m^1 \nu_n^1 \sigma_p^1 \tau_q^0 L_{m,n,p,q+1} \qquad (4.4)\\ &+ \mu_m^1 \nu_n^1 \sigma_p^0 \tau_q^1 L_{m,n,p+1,q} + \mu_m^1 \nu_n^1 \sigma_p^0 \tau_q^0 L_{m,n,p+1,q+1}\\ &+ \cdots + \mu_m^0 \nu_n^0 \sigma_p^0 \tau_q^0 L_{m+1,n+1,p+1,q+1}\end{aligned}$$

The righthand side of this expression can also be written as:

$$\sum_{i=0}^{1} \sum_{j=0}^{1} \sum_{k=0}^{1} \sum_{\ell=0}^{1} \mu_m^i \nu_n^j \sigma_p^k \tau_q^\ell L_{m+i,n+j,p+k,q+\ell} \qquad (4.5)$$

Figure 4.1 shows the average rendering time for ten different viewpoints (V_x, V_y, V_z) repeated in five experimental trails representing the user movement in 3D space. We can observe that the rendering time is the same for all viewpoints. Thus, it does not matter what viewpoint the user chooses as the rendering times for the different viewpoints have very close average values with small standard deviation (± 0.8 ms to ± 3.5 ms). This indicates that the scene complexity does not affect the total rendering time. The navigation path defining the viewpoints used started at (0,0,4) and then visited the following succession of points: (-0.649806, -0.375056,4), (-0.649806, -0.388139,4), (-0.675972, -0.357611,4), (-0.702139, -0.361972,4), (-

0.767556, -0.379417,4), (-0.815528, -0.388139,4), (-0.828611, -0.388139,4), (-0.872222, -0.383778,4), (-0.894028, -0.375056,4), and (-0.920195, -0.357611,4). Note that $(x, y) = (0, 0)$ corresponds to the centre of an image.

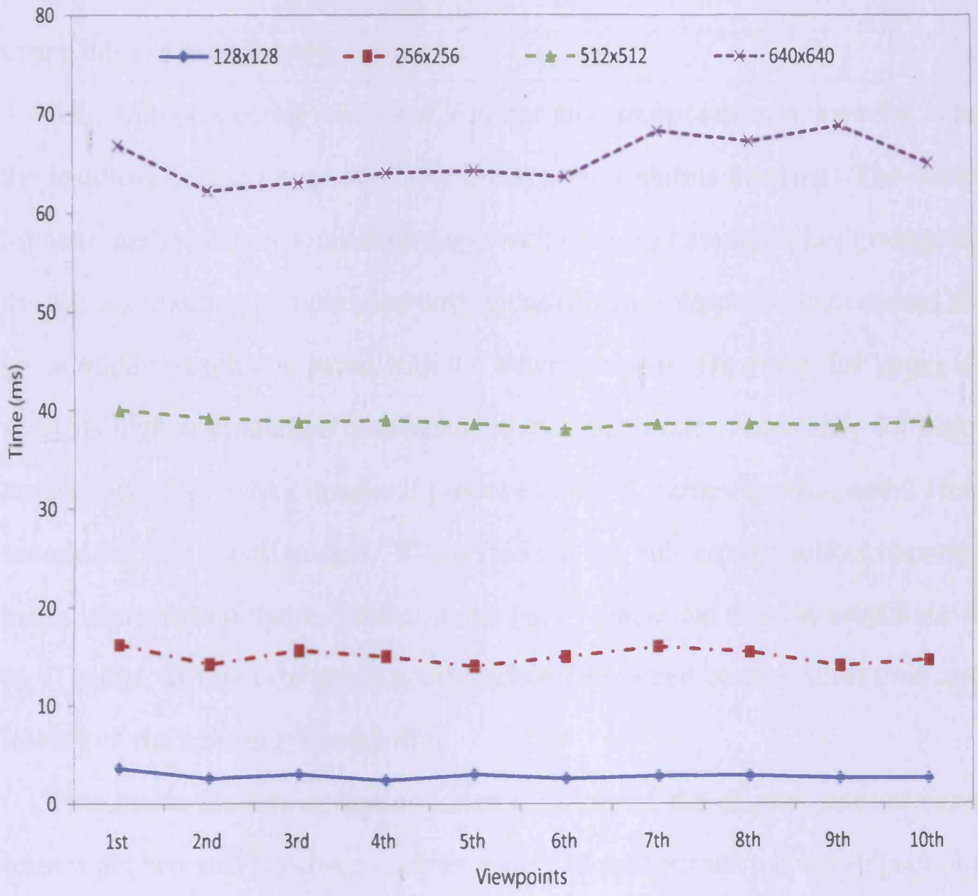


Figure 4.1: The rendering time for ten different viewpoints.

Fig. 4.2 plots the rendering times needed when different interpolation schemes are used for various sizes of FrameBuffer. It is clearly shows that rendering time increases for larger images and also clearly shows the difference in time required for the four interpolation schemes. In fact, if we look in more detail at the rendering time for one pixel (by dividing the total rendering time by the number of pixels) we can conclude that the interpolation time is equal to a small constant, α, for the interpolation operation times the number of pixels in the output image. The overall rendering cost will depend only on the number of pixels in the output image. This helps in predicting the rendering time for any image size within the upper limit of the memory.

Although producing the output without any interpolation processing is faster, the resulting output image is of low fidelity and exhibits blurring. The other two bilinear methods show similar timings, with case (c) having higher timings due to the bilinear method computation cost. Quadrilinear interpolation produces the highest fidelity result compared with the other schemes. However, for larger image sizes its high computation cost makes it an appropriate choice only for high-end devices: for 512×512 images it produces only 3 frames/second, and 2 frames/-second for 640×640 images. We carried out the subsequent sets of experiments using interpolation that is bilinear in the (u, v) plane and nearest-neighbour in the (s, t) plane, as this provides a good trade-off between computation time and the fidelity of the resulting framebuffer.

This thesis focuses on performance in terms of the display rate measured in frames per second. However, another aspect of performance is the impact of each interpolation method on the quality of the displayed image. Thus, in general, it might be expected that using no interpolation would result in a poorer quality image than when quadrilinear interpolation is used. If the variation in image quality for different interpolation schemes is significant then clearly there is a trade-off between frame rate and image quality which must be considered. Figure 4.3 shows a rendered image for each of the four interpolation schemes considered above, and the quality for no interpolation is discernably poorer than when

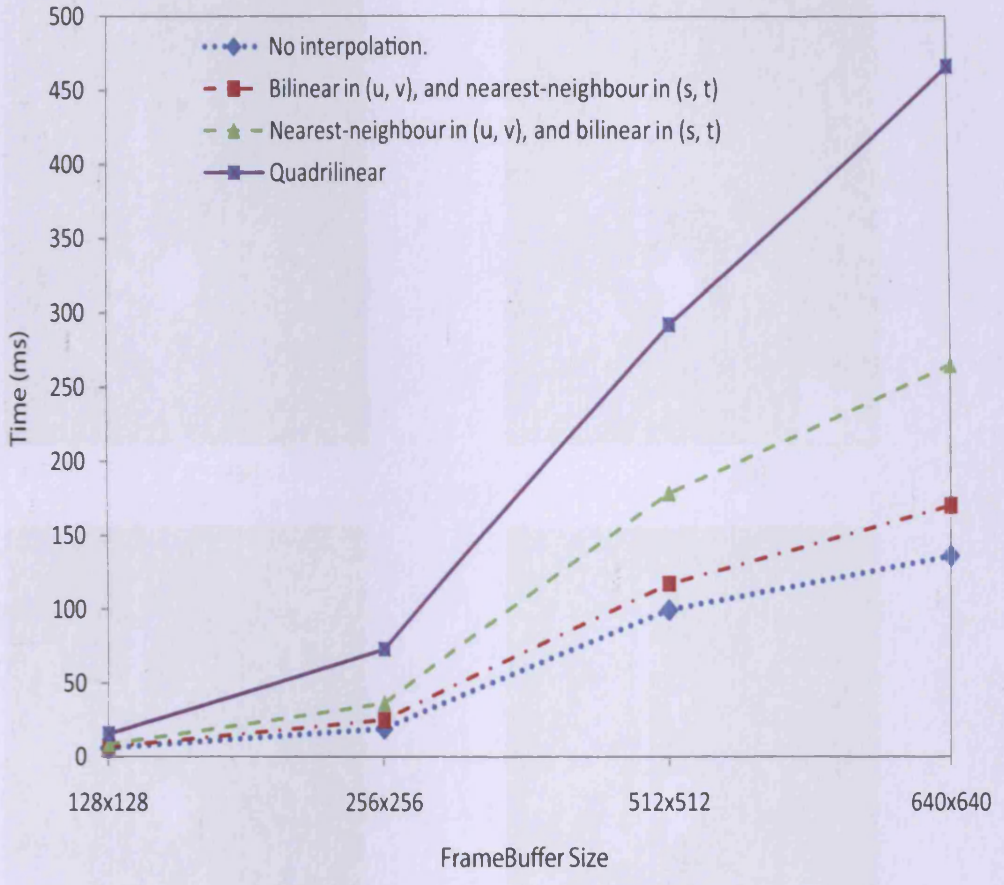


Figure 4.2: The rendering time for different interpolation schemes.

quadrilinear interpolation is used.

4.4 Effect Of Different Transmission Models

This experiment studies the relative amounts of time spent in rendering and communication. We further subdivide the communication time into actual transfer time for the data and the queuing time waiting for processing on the server side. In this experiment 512×512 datasets are used as it is a typical image size used in

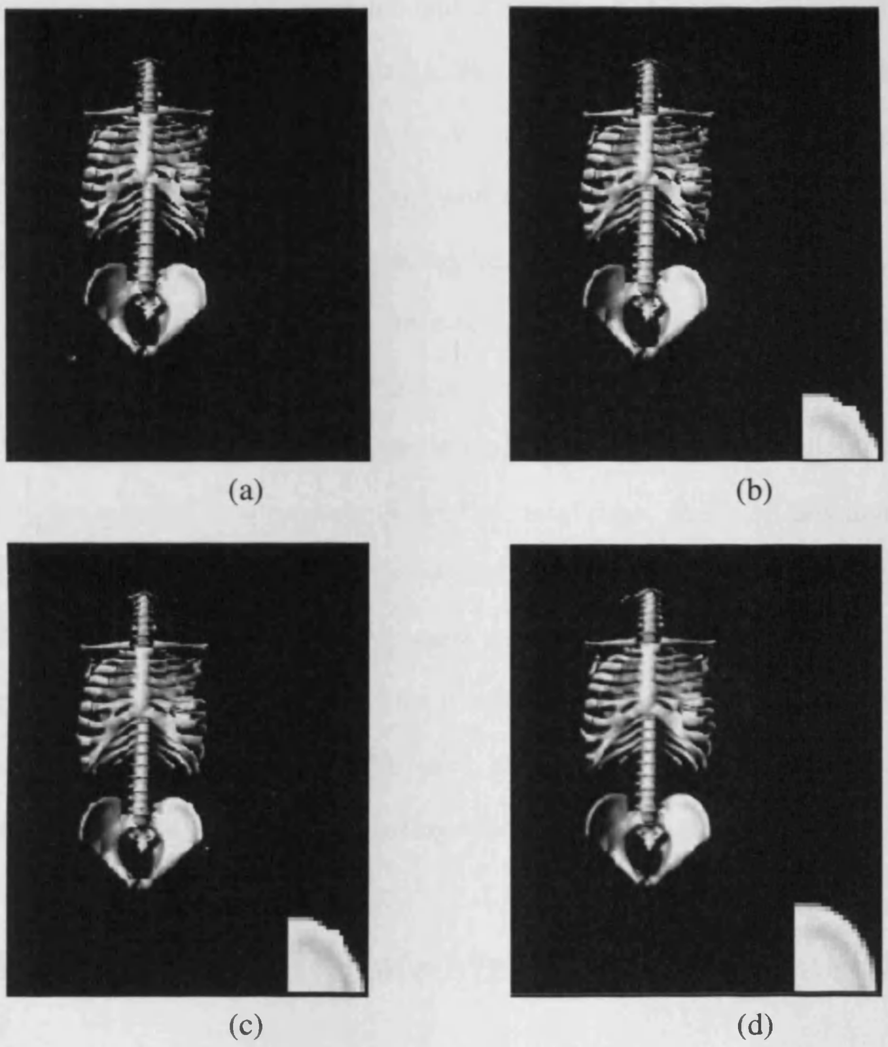


Figure 4.3: The effect of interpolation scheme on image quality: (a) without interpolation; (b) nearest-neighbour in (s, t) and bilinear in (u, v); (c) bilinear in (u, v) and nearest-neighbour in (s, t); (d) quadrilinear. The inset in the lower left corner of each image shows an enlarged portion of the iliac crest from the righthand side of the skeleton. This shows in more detail the differences in image quality for the four interpolation schemes.

many applications. As seen in Fig. 4.4 and Table 4.2, communication time (transfer plus queuing) becomes increasingly important as more clients join the visualization. This is a common drawback for all systems based on a central server as the server becomes overloaded and clients need to queue before their requests are processed. In remote rendering, the queuing time eventually dominates the overall performance, being 55% for 32 users. While the original components of the visualization process (rendering and transfer) only occupies 45% of the overall time, the rest is wasted in queuing time. In the case of local rendering, where the whole dataset needs to be transferred, the queuing time shows an even higher influence as it takes more than 90% of the total time.

The on-demand case illustrates the effectiveness of this mechanism by reducing the queuing time to about 20% of the total time. In the on-demand approach the communication time and the rendering time do not change much with the number of viewing clients (once there are more than two), which indicates the reduction in the server load and the overall reduction in the network traffic compared with the remote and local cases, respectively. The lower server load is a consequence of the use of processing resources on the client side.

4.5 Interactivity Rate Measured In Frames Per Second

As previously mentioned, the frame rate is a major benchmark tool to assess the whole visualization process. Exporting any stand-alone visualization system into a distributed computing environment means less performance and more variation is expected due to the nature of the network. Thus, providing a high and constant frame rate is difficult to achieve within a distributed computing environment.

Figure 4.5 and Table 4.3 illustrate the performance measured in frames per se-

No. Of Viewers	Rendering Mode	Rendering Time(ms)	Comm. Time(ms)
1	Remote	19.9	9.6
	On-Demand	74	37
	Local	58	966
2	Remote	19.6	9.35
	On-Demand	73.5	55.75
	Local	61	905.5
4	Remote	18.975	9.55
	On-Demand	73.5	55.75
	Local	62	966
8	Remote	18.5875	39.475
	On-Demand	73.125	60.125
	Local	60	1074.125
16	Remote	18.8875	110.725
	On-Demand	73	51.25
	Local	60	2388.375
32	Remote	18.534375	229.875
	On-Demand	73.03125	47.28125
	Local	64	21635.13333

Table 4.2: The amount of time spent by each client in rendering and communication.

cond (FPS). At small resolutions (128×128 and 256×256) the remote rendering case shows a higher interactivity rate due to the small time interval taken by the rendering process (averages of 2.9 ms and 14.2 ms). For a large dataset (512×512 and 640×640) remote rendering performs slightly better with one or two viewers, but on-demand rendering gives better performance for more than two viewers. As the dataset resolution increases the rendering carried out by the server machine takes a longer time. The rendering time on the server is also increased as more clients join and this will eventually overload the server so that rendering requests from clients must be queued, which causes further delay. The on-demand rende-

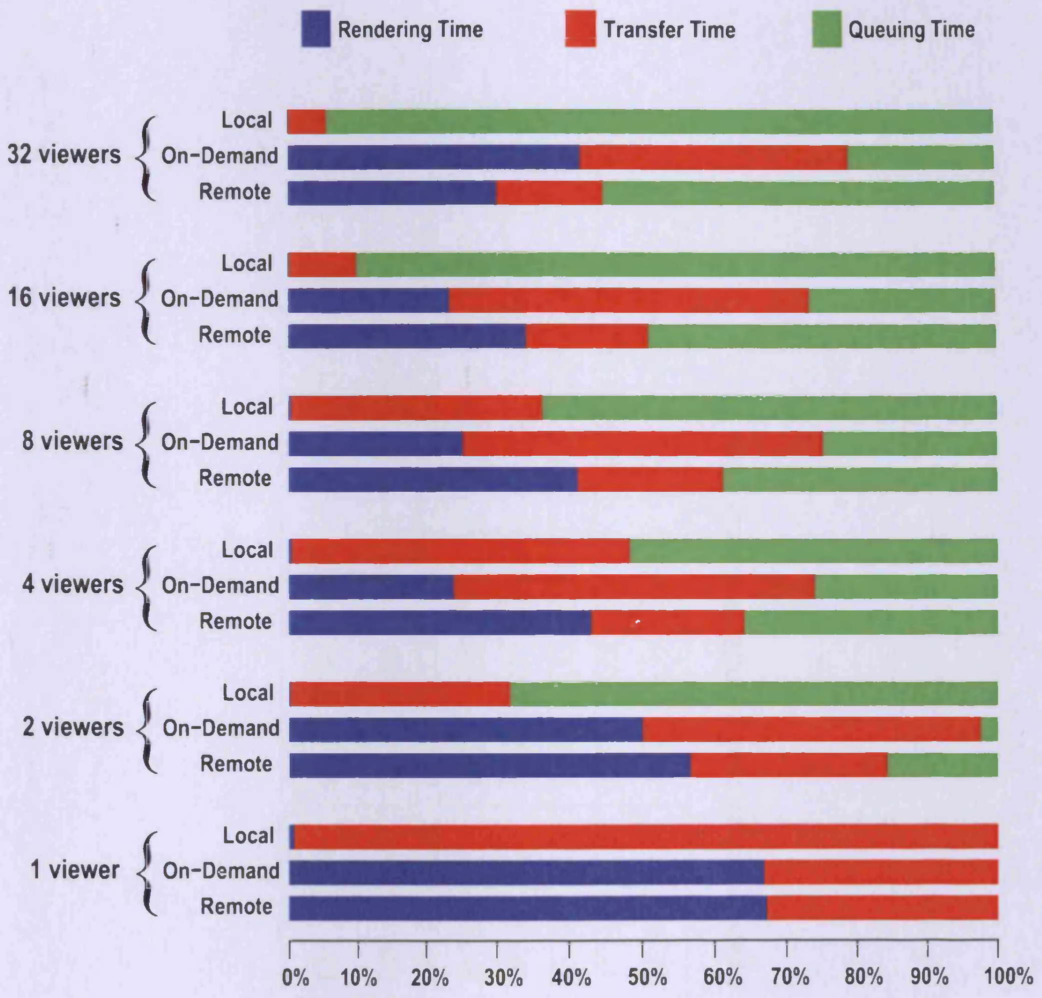


Figure 4.4: The percentage of time spent by each client in rendering and communication.

ring results show a more constant performance under different client loads. The maximum standard deviation shown is ±0.7ms for on-demand rendering, ±1.2ms for remote rendering, and ±1.7ms for local rendering.

Number of viewers	Local				On-Demand				Remote			
	128 x 128	256 x 256	512 x 512	640 x 640	128 x 128	256 x 256	512 x 512	640 x 640	128 x 128	256 x 256	512 x 512	640 x 640
1	1.22	1.22	0.105	0.0820	84.033	24.65	10.32	6.648	178.57	41.32	17.30	10.289
2	1.21	1.18	0.034	0.025	92.59	22.72	10.512	7.021	181.812	40.99	14.50	10.32
4	1.12	1.13	0.051	0.0201	102.04	25	10.087	6.77	182.645	42.15	11.05	6.42
8	1.099	1.0526	0.038	0.012	86.206	24.19	10.39	6.71	182.09	41.48	10.51	4.59
16	0.507	0.19	0.01017	0.0078	73.53	21.27	9.95	6.68	181.41	38.64	8.82	5.36
32	0.06	0.022	0.0058	0.003	89.29	22.43	9.69	6.37	159.92	44.87	7.69	2.62

Table 4.3: Frames per second for different scenarios.

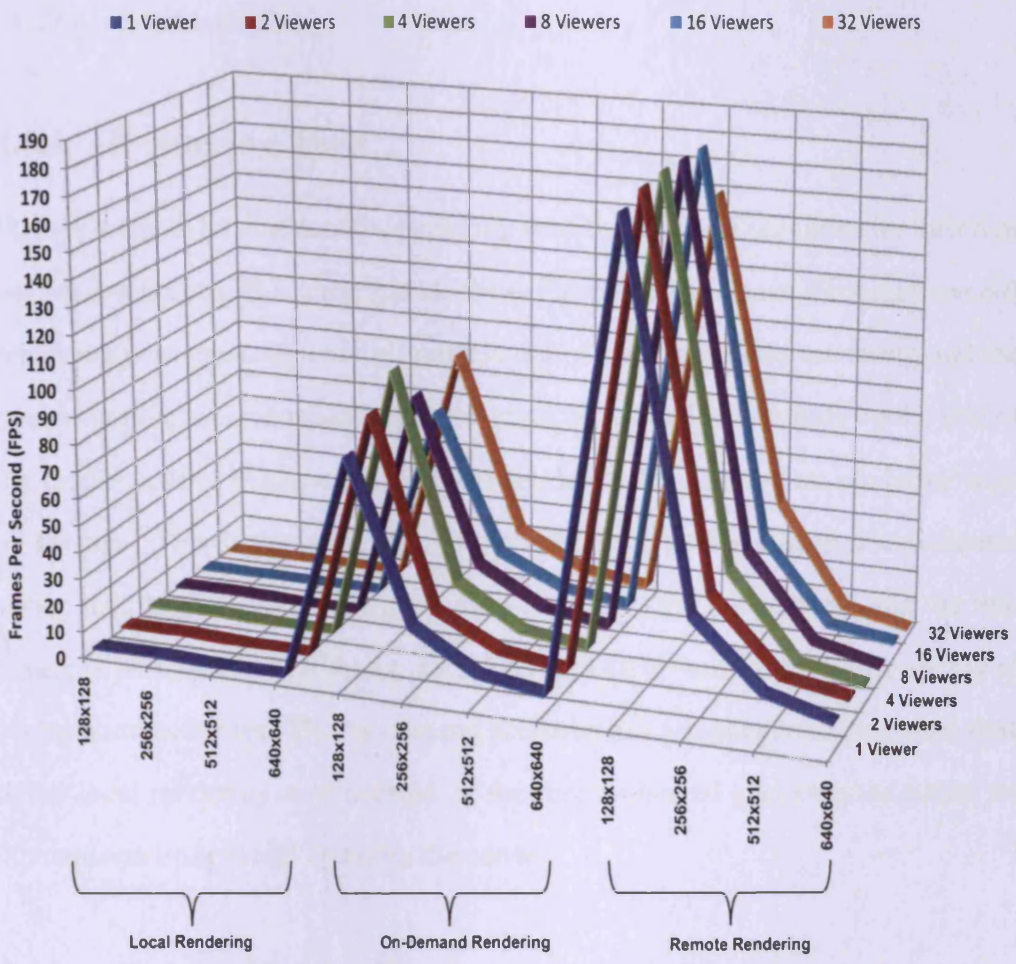


Figure 4.5: Frames per second for different scenarios.

4.6 Effect Of The Number Of Concurrent Visualization Clients

Initially each client starts a connection with the server. On the server side the server forks a new thread for each of the individual client connections so that all the connection requests can be handled concurrently. We have examined the effect of the number of visualization clients for local rendering, remote rendering, and on-demand rendering.

4.6.1 Rendering Time

Figures 4.6 and 4.7 compare the rendering time for the three scenarios for different dataset resolutions. It is clear that all of the three scenarios have a constant overall rendering performance, since all models are using Light Field rendering and the rendering process is independent of the viewpoint and depends only on the size of the final resulting framebuffer. Remote rendering has the lowest rendering time on average. This is due to the fact that rendering is performed on one dedicated server with higher processing capabilities, while in the on-demand case the rendering is performed on different, non-dedicated client machines with a variety of background activities. The on-demand scenario has a higher rendering time than in the local rendering case because of the mechanism of checking the cache for the required images and updating the cache.

4.6.2 Communication Time

An examination of the communication times for the three transmission models (see Figs. 4.6 and 4.7) shows that the local rendering approach has the highest communication time (15.7 seconds for the 128×128 dataset and 288 seconds for the 640×640 dataset) for 32 concurrent clients, because clients must pre-fetch the whole dataset before rendering. For the smaller 128×128 and 256×256 datasets the remote rendering case has slightly better performance, but as the dataset size increases then for a larger number of clients (more than two) the on-demand case has more stable performance.

4.6.3 Total Time

The overall impact of the communication time (see Fig. 4.8) varies according to the dataset size and the number of visualization clients. For the remote case the communication time is similar to the rendering time for the 128×128 dataset, while for the 256×256 dataset the average rendering time is 14ms and the communication time is 10ms. However, for the 512×512 dataset the rendering time is 40ms and the communication time rises from 19ms for one client to 91ms for 32 clients. At this point the communication time starts dominating the total time. The on-demand approach overcomes the server queue time by processing the rendering at each client, and by building a cache of images.

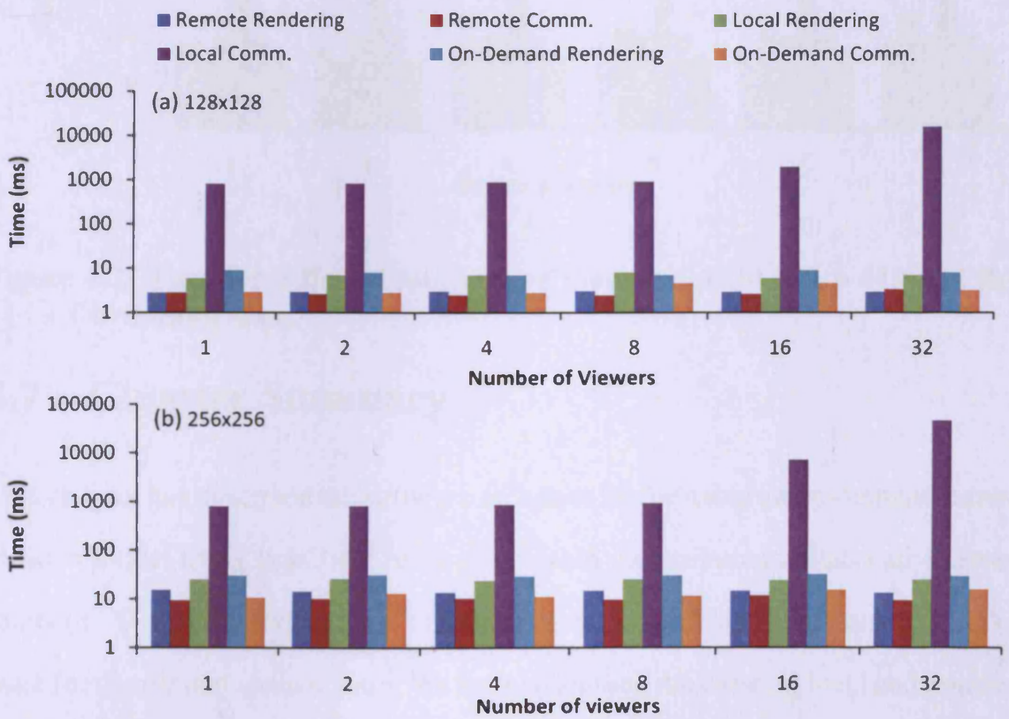


Figure 4.6: Time for different numbers of viewers for (a) 128×128 and (b) 256×256 dataset sizes.

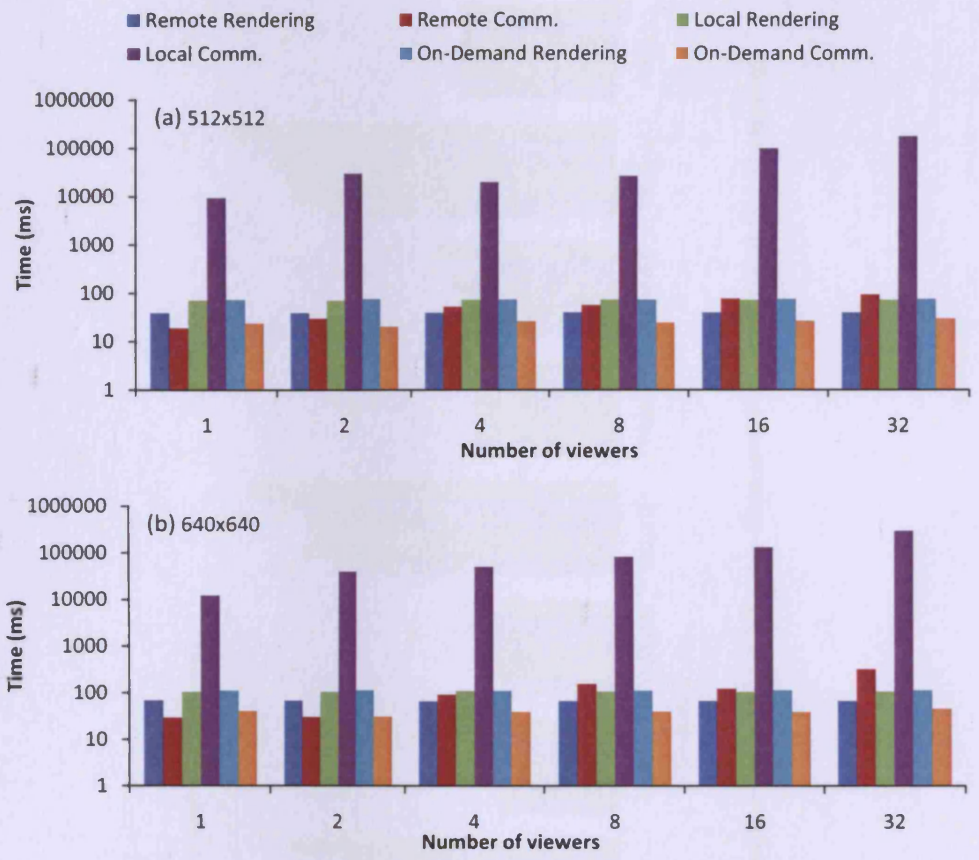


Figure 4.7: Time for different numbers of viewers for (a) 512×512 **and (b)** 640×640 **dataset sizes.**

4.7 Chapter Summary

This chapter has described the software architecture for using an on-demand transmission model using light field rendering within a distributed collaborative environment. The system makes use of commodity networking, hardware, and software for distributed visualization. We have discussed the existing local and remote transmission models, and compared them with our on-demand model.

Our results show that a light field rendering system enables constant frame

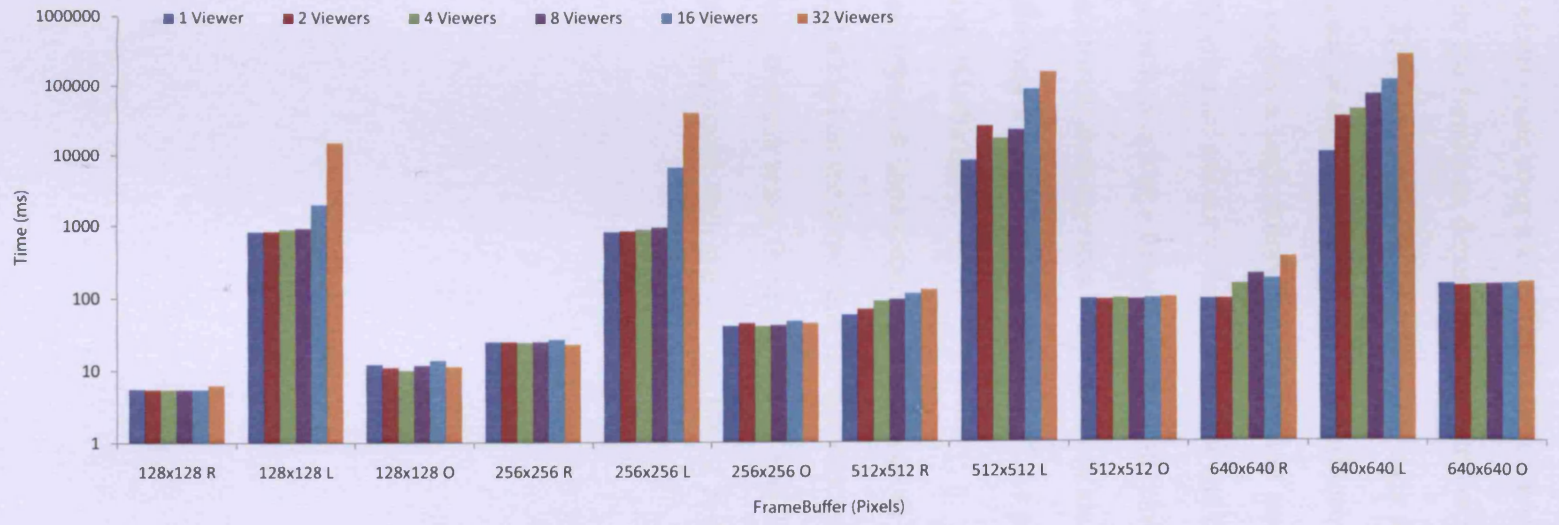


Figure 4.8: Total time for different dataset sizes for the remote (R), on-demand (O), and local (L) scenarios.

rates independent of the scene complexity. This is in contrast with geometry-based visualization, where the frame rate depends on the complexity of the scene for the chosen viewpoint. The comparative efficiency of the light field rendering system increases as the scene complexity increases. The applicability of the light field rendering system covers a large number of application areas, where interactive collaboration could enhance and accelerate navigation and discovery.

Our system currently provides a generic on-demand remote rendering solution for users to access visualization services using computationally inexpensive light field rendering, allowing low-end devices, such as PDAs and mobile phones, to interactively explore 3D objects.

The on-demand approach shows stable performance as the number of clients increases because the load on the server and the network traffic are reduced. The main performance bottleneck arises from the limited transfer rate across the network, which can be improved with the availability of higher performance networks.

Chapter 5

Theoretical Cost Comparison for Different Transmission Models

Overview

This chapter will discuss a simple theoretical cost model for the three image-based rendering scenarios: local, remote and on-demand. We aim to find a cost model that provides a reasonable fit to the performance data. These models will provide a comparison with experimental results presented in the previous chapter which will enable validation and interpretation of these results. Furthermore, such models can be used as predicative tools to project performance for computation and communication time where parameters are different from those measured experimentally.

The organization of this chapter is as follows. Section 5.1 discusses the performance characterization of each of the three transmission models. Section 5.2 presents a brief overview of the basic concepts and parameterization of our queueing analysis. The results in Section 5.4 compares the estimated model results with the experimental ones. In Section 5.5 an estimate of the predicted performance for other situations of interest is presented. Section 5.6 explains different reasons that may result in discrepancies between the models and the measured performance results. Finally the important conclusions and some ideas for possible future work are highlighted.

5.1 Performance Model

In this chapter we develop theoretical performance models for each of the three transmission models that been discussed in the previous chapters. For the analysis we consider each method individually. We examine the performance characterization of each model, and compare the theoretical estimated performance with the experimental results. Furthermore, these models can be used to estimate the costs of the computation and communication. Theoretical models are abstract, and hence many assumptions are made in conducting a modeling study. These assumptions are motivated by simplicity, adequacy of measurements, and ease of evaluation. Examples of parameters that are variable and may be tuned in the performance model are the following:

- Type of transmission protocol. We used TCP as it is retransmits the lost packets and so results in a more reliable service.
- Data-set attributes (image type, size, depth, etc.).
- Number of visualization clients.
- Network bandwidth and latency.

5.2 Queuing Analysis

Before we describe the models in detail, it is apparent that, based on what we have seen from the results presented in the previous chapter, the server can serve only one client request at a time. Thus, if the server gets more rendering requests within a short period time the service delay in the system will also increase. Understanding the relationship between congestion and delay is essential for designing an effective solution to this problem. *Queuing Theory* provides the tools needed for such an analysis. In this section we describe the basic queueing model and we discuss some important fundamental relations for this model. Further details on these methods can be found in standard textbooks on queuing theory [44, 66, 97, 102].

There is a standard notation for classifying queueing systems into different types proposed by Kendall [1]. This is a shorthand notation to characterize a range of queueing models. It has a three-part code a/b/c. The first letter (a) specifies the interarrival time distribution, and the second one (b) the service time distribution. For example, for a general distribution the letter G is used, M for the exponential distribution (M stands for Memoryless), and D for deterministic times. The third and last letter (c) specifies the number of servers. Some examples are M/M/1, M/M/c, M/G/1, G/M/1 and M/D/1.

We will now categorize our system using this classification notation, based on the results of the previous chapter, by examining the fundamental parameters: the arrival rate ρ, the service time T_s, and the number of servers.

- If the arrival rates stay constant over the time period then the arrivals follow a Deterministic distribution (D). If no pattern of distribution applies, the arrivals follow a General distribution (G). If arrivals occur continuously and independently of one another, then they follow a Poisson distribution, thus (a)=M (M stands for memoryless). Further descriptions of the Poisson classification are based on following points:

  - Requests are processed in a sequence. If requests are submitted simultaneously, one is done first. Order is not relevant.

  - Arrivals can have known peaks, but knowing the time should not allow the prediction of the number of arrivals.
  - The actions at one node (or of one user) should not affect the requests from other nodes. A user entering data may wait and consider the effect of an action before making another request but this does not affect other users' actions.

  In this research we focus on the Poisson distribution as our system fits into the above description of this category.

- Based on the experiments in the previous chapter, the rendering time, which corresponds to the service time in the queueing system parameters, is constant so (b)= D for the Deterministic case.
- A single server is used in the system, therefore, (c)=1.

The description of the queuing system is the M/D/1 queue, where job inter-arrivals are Markovian or Memoryless (M), service times follow a Deterministic distribution (D), and there is a single server in the system.

5.2.1 Queuing Parameters

The main parameters used by the *Queueing System* are summarized in Table 5.1.

To model the time spent queuing we make the following assumptions:

- We assume a single queue model, in distinction to other model approaches that consider situations where there are multiple queues.

Parameters Description	
λ	Parameterizes the arrival rates in terms of the mean number of arrivals per second
T_s	Parameterizes the mean service time for each arrival, i.e., the amount of time being served, not counting time waiting in the queue.
ρ	Utilization; fraction of time facility (server or servers) is busy.
T_w	Mean waiting time, including items that have to wait and items with waiting time of zero.

Table 5.1: Parameters descriptions.

- The arrival rate obeys a Poisson distribution, so arrivals occur randomly and are independent of one another.
- Since the service time T_s (which in our case equals the rendering time) is constant, the standard deviation is almost zero, and in all the cases the difference is due the factors explained in the last section of this chapter.
- Items/arrivals have the same priority. No item is discarded from the queue.
- The queueing protocol is First In First Out (FIFO), which is the easiest queueing system to analyze.

To analyze the queuing time for a certain system, we need to know input parameters such the arrival rate and the service time. We are particularly interested in finding T_w, the mean waiting time, as it is has a major influence on the performance time.

5.3 General Performance Model

In this study, we focus on estimating the response time, which is defined as the total cost, T_{tot}, for one viewpoint (V_x, V_y, V_z) request to be processed. The total cost T_{tot} comprises three components: computation time (T_r), communication time (T_t), and queuing time (T_q). The computation time (T_r) is the average rendering time for one viewpoint query, whereas the communication time (T_t) is the time to transfer one request or result, not counting any queuing time (T_q) which might occur on the server side. Thus,

$$T_{tot} = f\left(T_r, T_t, T_q\right) \tag{5.1}$$

The rendering time T_r is

$$T_r = C_r(F) \tag{5.2}$$

where C_r is the rendering cost for each pixel, and $F = ST$ is the number of pixels in the $S \times T$ final image size (i.e., the size of the frame buffer).

The transmission time required to send data, or the output, is T_t:

$$T_t = \left[2L + \left[\frac{C_i D}{B}\right]\right] \tag{5.3}$$

where $2L$ is the round-trip latency, D is the data size in pixels, C_i is the number of bytes to store each pixel colour value, and the B is the network bandwidth.

When more clients join the visualization process, clients will experience a service delay. The queuing time for the (M/D/1) category can be time represented by [97]

$$T_q = \left[\frac{\rho T_s}{2\,(1-\rho)}\right] \tag{5.4}$$

where T_s is the mean service time for each arrival (not counting queueing time), and ρ is the utilization.

The utilization ρ is computed using Little's law [71],

$$\rho = \lambda T_s \tag{5.5}$$

where λ is the arrival rate.

In our system implementation the server assigns a thread each time it receives a request from a client. All the requests go into one queue and are scheduled to any available processor. Since our server is a tightly-coupled dual processor, if threads are not used then all client processing requests would be assigned to one processor while the other one will be idle. In general, the allocation of client requests to processors will be different for each execution. In our case, although we have one machine as server, in our model calculation we assume the arrival rate per processor is $\lambda/(\text{no. of processors})$ as the client requests are evenly distributed among the processes (on the assumption that the server machine is a dedicated machine with no other tasks).

5.3.1 Local Rendering Model

In this scenario the whole dataset is retrieved from the server (see Fig. 5.1). We make use of the *Curl* protocol. The dataset size is computed by all have a $U \times V$ array of 16×16 camera positions (ST) so there are 256 images for each dataset, and the transfer time is calculated using the equation:

$$D_l = C_i\,(UV)\,(ST) \tag{5.6}$$

where $UV \times ST$ is the dataset size in pixels.

The rendering time T_r is as given in Eq. 5.2, so the overall total execution time is:

$$T_{local} = \left[2L + \left[\frac{C_i[UV]\,[ST]}{B}\right]\right] + C_r\,(ST) + \left[\frac{\rho T_s}{2\,(1-\rho)}\right] \tag{5.7}$$

We are using a bilinear interpolation scheme with computational complexity O(N^2), where $N = ST$ is number of pixels in the final image.

5.3.2 Remote Rendering Model

In the remote rendering scenario the viewpoint request (V_x, V_y, V_z) is sent to the server. The server pushes the resulting images into the TCP socket as quickly as possible, and each client reads from its TCP socket as quickly as it can. After the rendering the resulting image is sent back to the client (see Fig. 5.2). The number of bytes to be transmitted is determined by the final output framebuffer size. Since is the total number of pixels copied to the final output framebuffer is $F = ST$, the rendering time is,

$$T_r = C_i\,[ST] \tag{5.8}$$

The overall time is calculated by the following equation

$$T_{Remote} = \left[2L + \left[\frac{C_i\,(ST)}{B}\right]\right] + C_r\,(ST) + \left[\frac{\rho T_s}{2\,(1-\rho)}\right] \tag{5.9}$$

The first term gives the network latency and the second gives the transmission

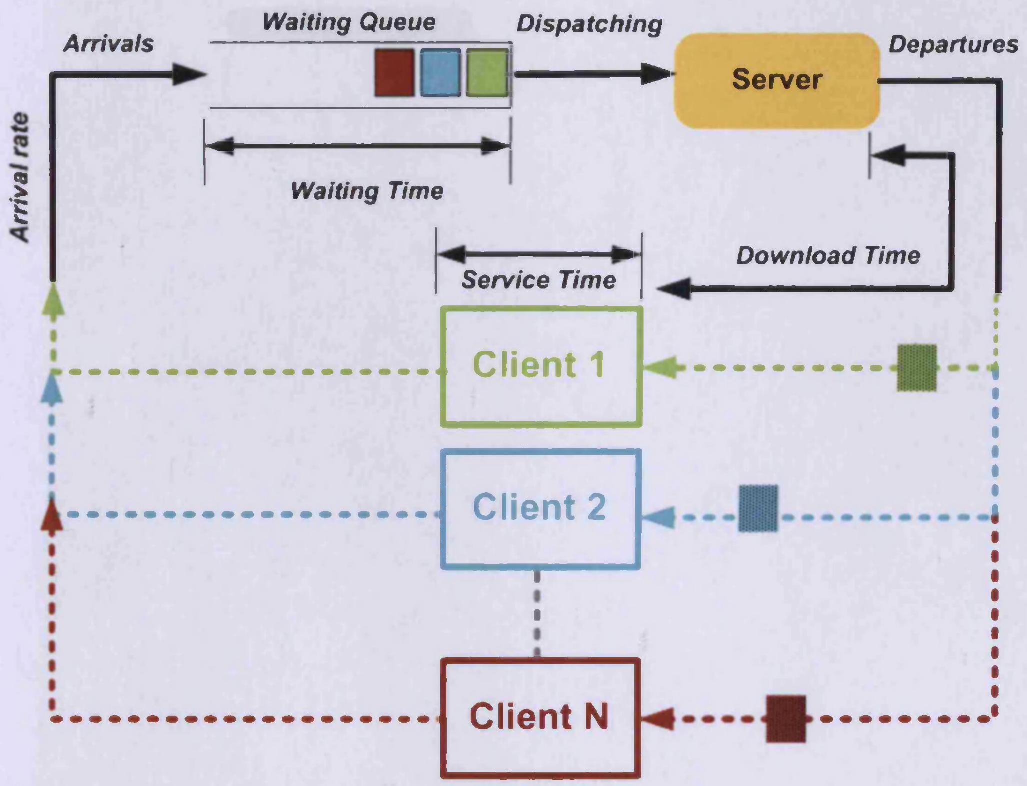


Figure 5.1: Local rendering system architecture and parameters for single server queuing.

time for all pixels. L is the network latency, ST is the number of pixels, and B is the network bandwidth. We assume 8 bits per pixel for colour or greyscale information.

5.3.3 On-Demand Rendering Model

In the on-demand rendering scenario, as the user interacts with the *Viewer Process*, changing his/her viewpoint parameter, then a viewpoint query is sent to the *Client Manager Process* for rendering. The *Client Manager Process* determines the images needed for rendering that user viewpoint, and if all the images are lo-

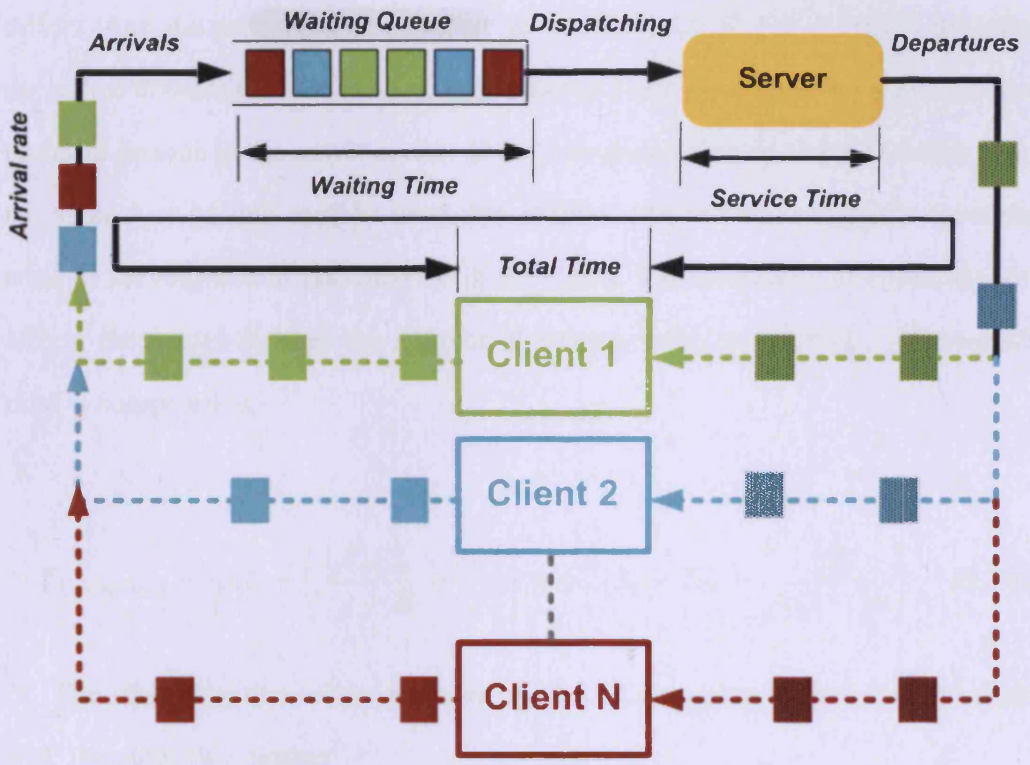


Figure 5.2: Remote rendering system architecture and parameters for single server queuing.

cated in the cache then the *Rendering Process* performs interpolation to produce the desired view. Otherwise, the missing images will be requested from the server. The resulting image is received from the *Rendering Process* and mapped onto the projection plane (see Fig. 5.3).

The amount of data that needs to be transferred in the on-demand scenario is calculated by:

$$T_t = \left[2L + \left[\frac{I_r(ST)}{B} \right] \right] \tag{5.10}$$

where I_r is the number of images needing to be transferred from the server each time. The value of I_r varies and depends on how much the current viewpoint differs from the previous ones, and we have $0 \leq I_r \geq 4$. For example, initially the server downloads four images, and then for a new viewpoint it checks for the required images in the server cache. If the viewpoint has not changed much then the same four images may be used, but if the viewpoint has changed a lot then none of the required images may be in the cache. The frequency of cache misses affects the model through the number of images to be transferred. The overall time is computed as:

$$T_{Ondemand} = \left[2L + \left[\frac{I_r C_i (S_i \times T_i)}{B}\right]\right] + C_r\left(S_i \times T_i\right) + \left[\frac{\rho T_s}{2\left(1-\rho\right)}\right] \quad (5.11)$$

The rendering time is the same in Eq. 5.8. Comparing the on-demand case with the other two models:

1. In the remote rendering scenario the navigation scheme does not effect the overall performance as for the remote rendering the user issues a rendering query to the server for each new viewpoint.

2. In the the local rendering scenario the whole dataset is downloaded so the client does not have any more contact with the server.

5.4 Results

We now determine the total time for each model mentioned in the last section for the particular experimental setup used. We first determine the cost for C_i and C_r. We then analyze the performance for various image sizes ($S \times T$), and

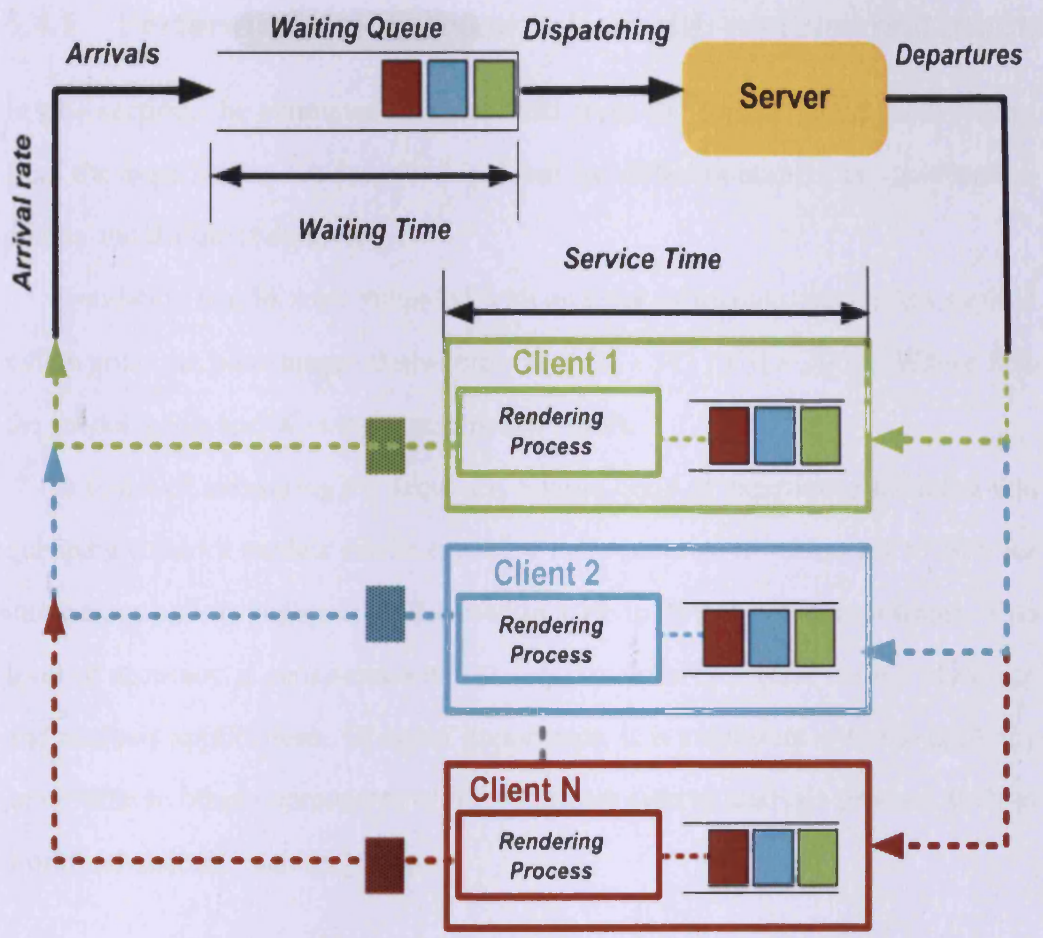


Figure 5.3: On-demand rendering system architecture and parameters for single server queuing.

for N vizualization clients. The network bandwidth is estimated based on the `ibmonitor` software[27].

B	97.5 Mb/s [1]
L	0.675 ms
C_r	1.135790×10^{-4} ms per pixel
C_i	8 bits per pixel [2]

Table 5.2: Parameter values.

5.4.1 Performance model comparison with experimental results

In this section, the estimated costs derived from the formal models and results from the experiments are described in detail for different number of visualization clients and different datasets.

Simulation results were validated with an error estimation calculation method which gives the percentage relative error as $|((Y - X)/Y)| \times 100\%$. Where Y is the model result and X is the experimental result.

In terms of measuring the accuracy, a large body of experience indicates that queueing network models can be expected to be accurate to within 5% to 10% for utilizations and throughputs, and to within 10% to 30% for response times. This level of accuracy is consistent with the requirements of a wide variety of design and analysis applications. Of equal importance, it is consistent with the accuracy achievable in other components of the computer system analysis process, such as workload characterization [44].

5.4.2 Remote Rendering for One Client

We initially investigate the results for one user, with waiting time T_w set equal to zero. Then we use these results to get the values for the queueing time when there are more visualization clients. By using the parameters in Table 5.2 in Eq. 5.9 for each S_i and T_i, where $S_1 = T_1 = 128$, $S_2 = T_2 = 256$, $S_3 = T_3 = 512$, and

[1]Ethernet's maximum frame size is 1526 bytes (maximum 1500 byte payload + 8 byte preamble + 14 byte header + 4 Byte trailer). An additional minimum interframe gap corresponding to 12 byte is inserted after each frame. This corresponds to a maximum channel utilization of 1526/(1526+12)Ţ100 % = 99.22%, or a maximum throughput of 99.22 Mbit/s inclusive of Ethernet datalink layer protocol overhead in a 100 Mbit/s Ethernet connection. The maximum throughput is 1500/(1526+12) = 97.5 Mbit/s exclusive of Ethernet protocol overhead [35].

[2]The total number of colours is 2^8 =256 for our images.

$S_4 = T_4 = 640$, then the total cost is :

$$T_{Remote}(S_i, T_i) = 2(0.675) + \left[\frac{8\,(S_i \times T_i)}{97.5\,(1024)^2}\right] + \left[(1.135790 \times 10^{-4})\,(S_i \times T_i)\right] \tag{5.12}$$

Table 5.3 and Fig. 5.4 show the average experimental time versus the theoretically estimated time for remote rendering from Eq. 5.9. The percentage difference between the theoretical model and experimental results is between 3.09% and 12.7%. Although 12.7% may seem to represent a big gap in reality it is 2.7ms as for a small dataset the system is unable to measure less accuracy to within less than 1 ms. Overall the theoretical model is a good estimate of the system performance. This performance study has shown that the model was able to provide realistic estimations of the results even for small (128 × 128) and large (640 × 640) datasets.

Dataset Size	Model Comm. (ms)	Experiment Comm. (ms)	Total Model (ms)	Total Experiment (ms)	Relative Error
128 × 128	2.63	2.8	5.43	5.6	3.09 %
256 × 256	6.48	9.2	21.48	24.2	12.7 %
512 × 512	21.86	19	60.66	57.8	4.7 %
640 × 640	33.40	28.9	101.70	97.2	4.4 %

Table 5.3: Theoretical model estimates and measured timing results for one remote vizualization client.

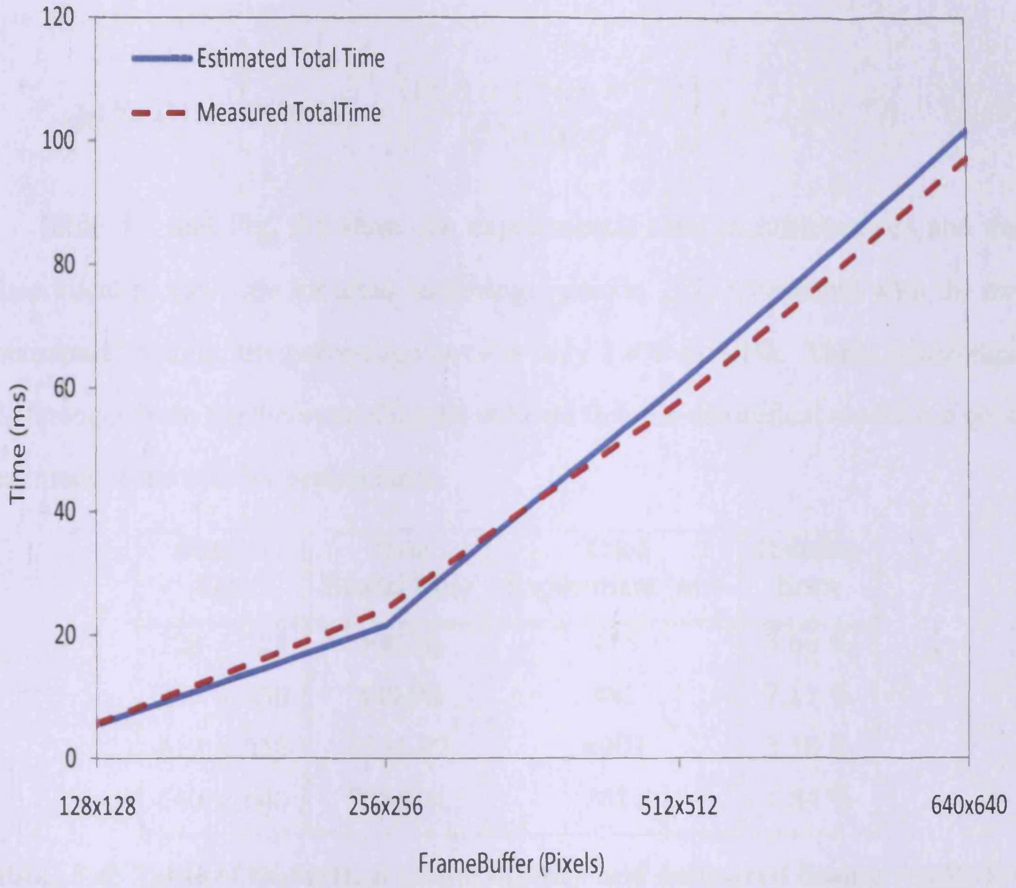


Figure 5.4: Remote rendering model and measured timing results for one visualization client.

5.4.3 Local Rendering for One Client

Similarly, initially we test our model for one visualization client, thus the waiting time/queuing time is zero. To compare the model with the experimental results, the parameter values from Table 5.2 are substituted into the local rendering modeling equation, Eq. 5.7. The total cost is :

$$T_{local}(S_i, T_i) = \left[2(0.675) + \left[\frac{8[16 \times 16]\,[S_i \times T_i]}{97.5\,(1024)^2}\right]\right] + C_r\,(S_i \times T_i) \qquad (5.13)$$

Table 5.4 and Fig. 5.5 show the experimental time in milliseconds and the theoretical model time for local rendering from Eq. 5.7. Compared with the experimental results, the percentage error is only 1.8% to 7.1%. These percentage differences from the theoretical model indicate that the theoretical model is a good estimate of the system performance.

Dataset Size	Total Model (ms)	Total Experiment (ms)	Relative Error
128 × 128	400.36	415	3.66 %
256 × 256	449.98	482	7.11 %
512 × 512	4731.62	4901	3.58 %
640 × 640	7286.91	7421	1.84 %

Table 5.4: Table of theoretical model timings and measured timing results for one local visualization client.

5.4.4 On-Demand Rendering for one client

For the on-demand rendering scenario, each time the user selects a viewpoint the client downloads the four nearest data images. As for the other two scenarios, initially we test our model for one visualization client, so the waiting time/queuing time is zero. To compare the model with the experimental results the parameters values from Table 5.2 are substituted into the on-demand rendering modeling equation, Eq. 5.11. The total cost is :

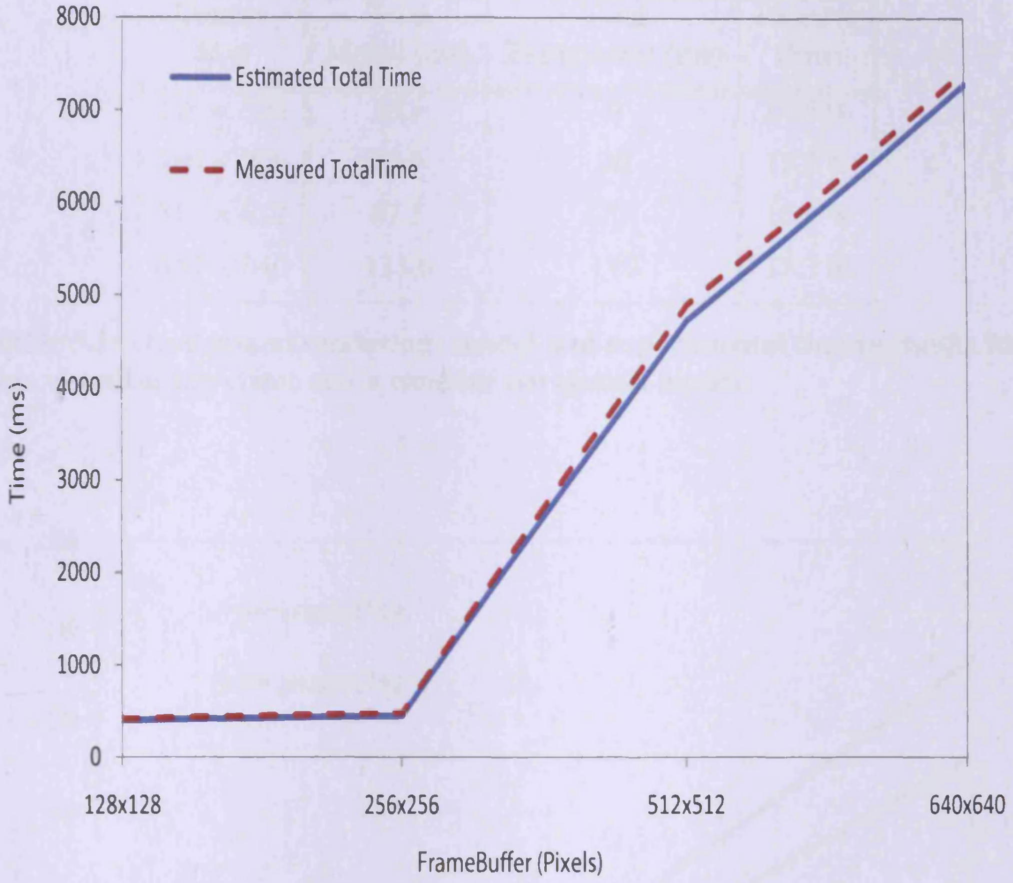


Figure 5.5: Local rendering model and measured timing results for one visualization client.

$$T_{Ondemand}(S_i, T_i) = \left[2(0.675) + \left[\frac{I_r(S \times T)}{97.5\,(1024)^2}\right]\right] \tag{5.14}$$

Table 5.5 and Fig. 5.6 compare the model and experimental timings in for the on-demand rendering scenario under the assumption of random navigation, whilst Table 5.6 and Fig. 5.7 show the same timings but under the assumption of coherent navigation – in this case rotation around the scene.

Although we did not achieve as good an agreement as for the remote and local

Dataset Size	Total Model (ms)	Total Experiment (ms)	Relative Error
128×128	10.5	9	14.5 %
256×256	25.9	30	15.8 %
512×512	87.5	73	16.5 %
640×640	133.6	110	17.7 %

Table 5.5: On-demand rendering: model and experimental timing results for one visualization client and a random navigation model.

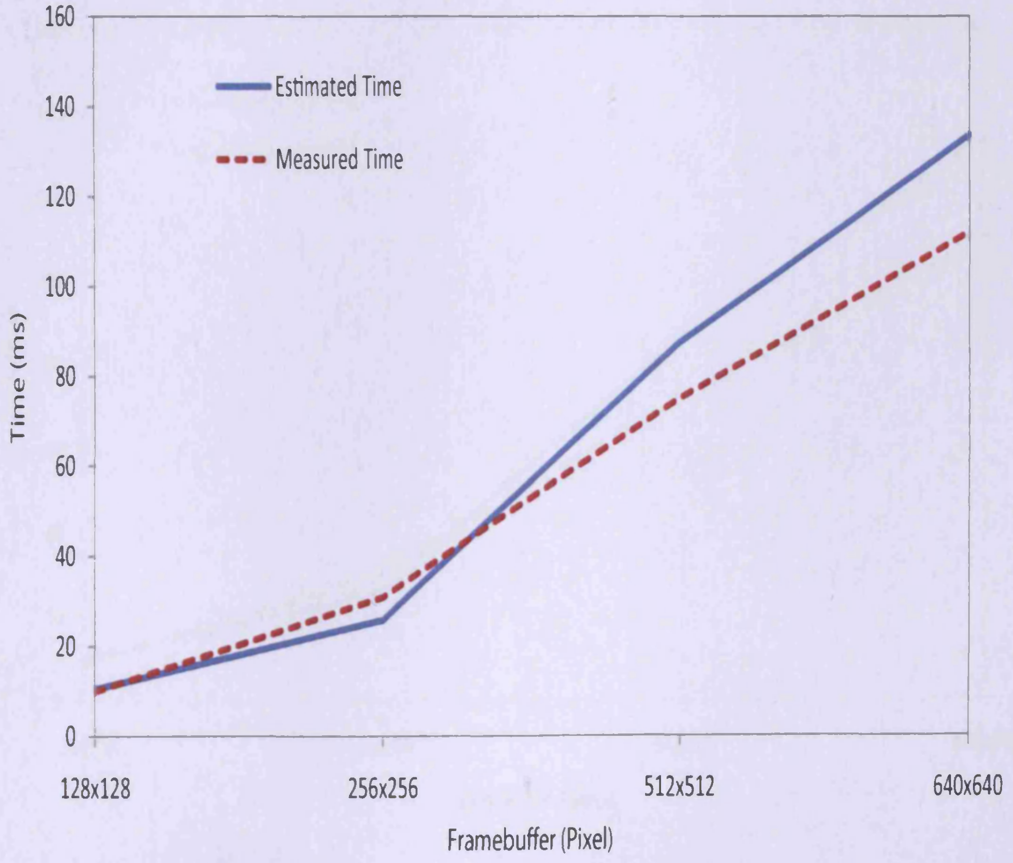


Figure 5.6: On-demand rendering: model and experimental timing results for one visualization client and a random navigation model.

Dataset Size	Total Model (ms)	Total Experiment (ms)	Relative Error
128×128	10.5	10	5.0 %
256×256	25.9	31	19.6 %
512×512	87.5	75	14.2 %
640×640	133.6	112	16.2 %

Table 5.6: On-demand rendering: model and experimental timing results for one visualization client and a coherent navigation model.

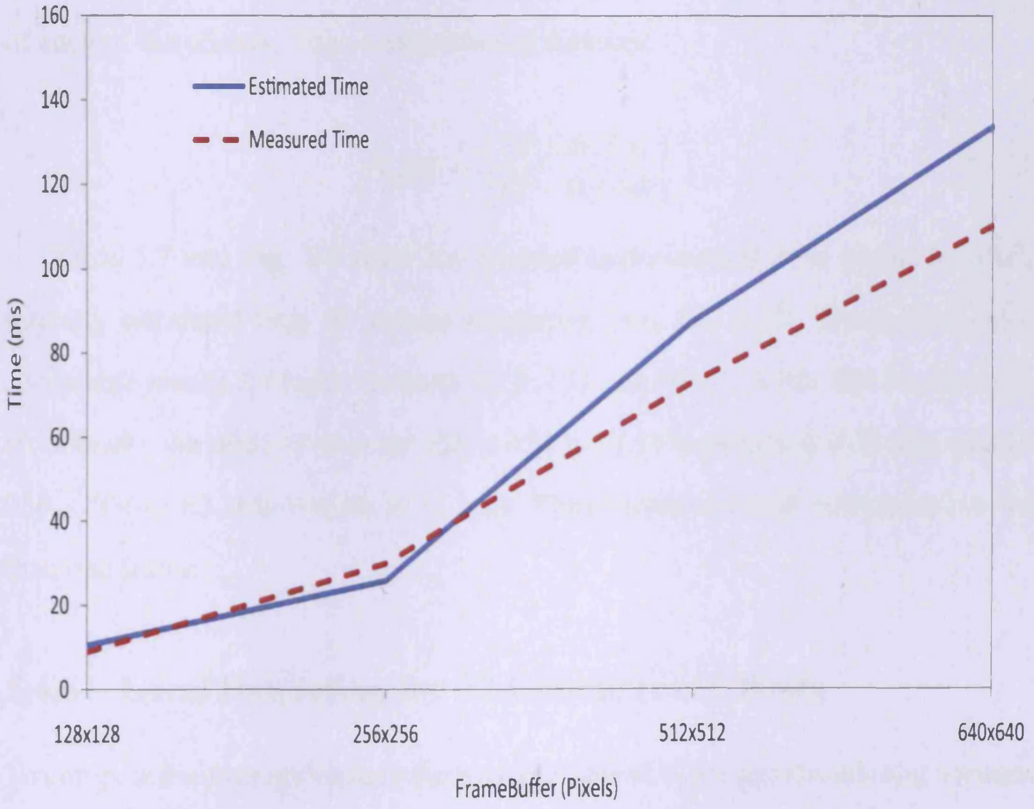


Figure 5.7: On-demand rendering: model and experimental timing results for one visualization client and a coherent navigation model.

rendering scenarios, we still obtained acceptable model estimates that were within 30% of the experimental values. The on-demand model shows less accuracy in comparison with the local and remote models because of the overhead of cache checking. In general, the comparison shows satisfactory agreement between the predictions of the numerical models and the experimental data.

5.4.5 Remote Rendering for 32 Concurrent Clients

To analyze how the performance varies as more concurrent clients join the visualization we need to calculate the average queuing time or, the average waiting time, of each of the clients. This is estimated as follows:

$$T_q(i) = \left[\frac{(0.136)Ts_i}{(2-0.136)}\right] \tag{5.15}$$

Table 5.7 and Fig. 5.8 show the average experimental time versus the theoretically estimated time for remote rendering from Eq. 5.15. The model shows acceptable results for larger datasets (512×512 and 640×640). But for the smaller datasets, the relative error for 128×128 is 62.17% (which is 0.33 ms), and for 256×256 is 83.16% (which is 11 ms). These relative errors correspond to less than one frame.

5.4.6 Local Rendering for 32 Concurrent Clients

To compute the average waiting time of each client in the local rendering scenario we use equation Eq. 5.15.

Figure 5.9 shows the experimental time in milliseconds and the theoretically calculated time for queueing given by Eq. 5.15. The results are also shown in

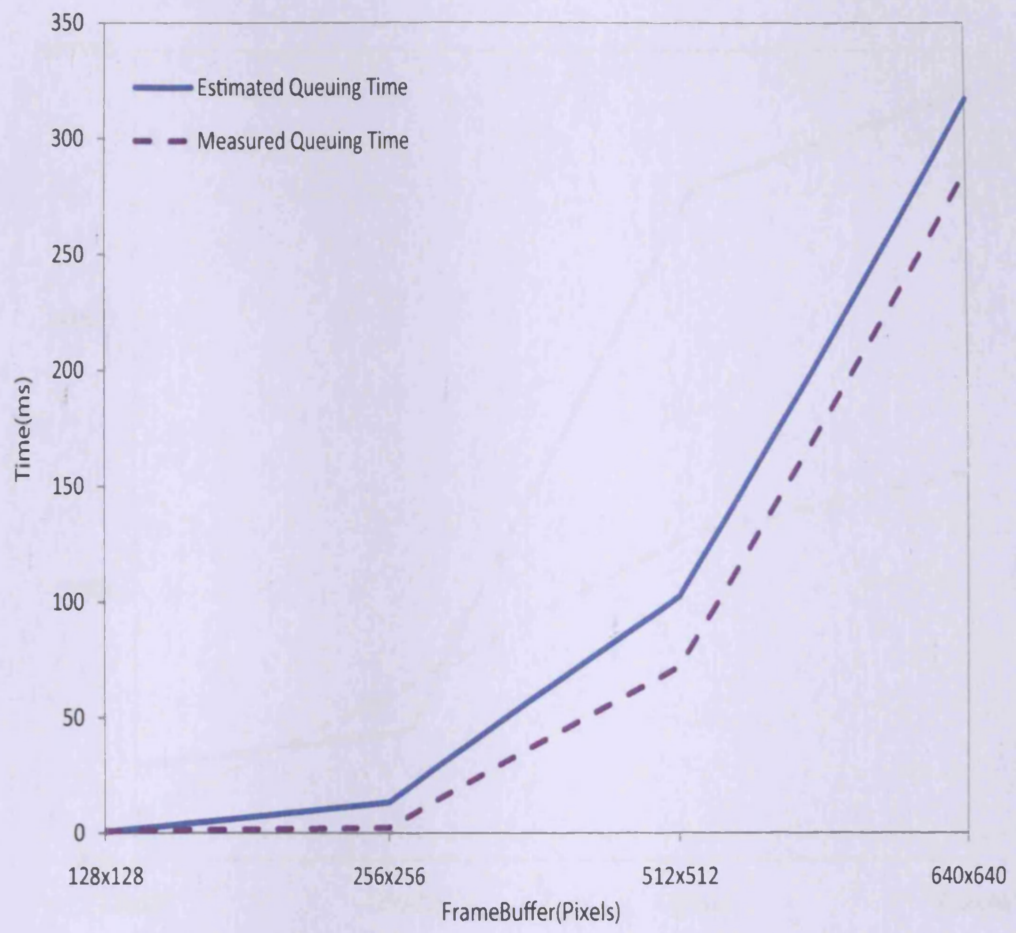


Figure 5.8: Queuing time model and measured timing results for remote rendering.

Dataset Size	Total Model (ms)	Total Experiment (ms)	Relative Error
128 × 128	0.53312	0.864583333	62.17%
256 × 256	13.328	2.243579545	83.16%
512 × 512	102.36992	72.103125	29.56%
640 × 640	317.21252	286.24375	9.76%

Table 5.7: Queuing time for remote rendering model and measured results for 32 clients.

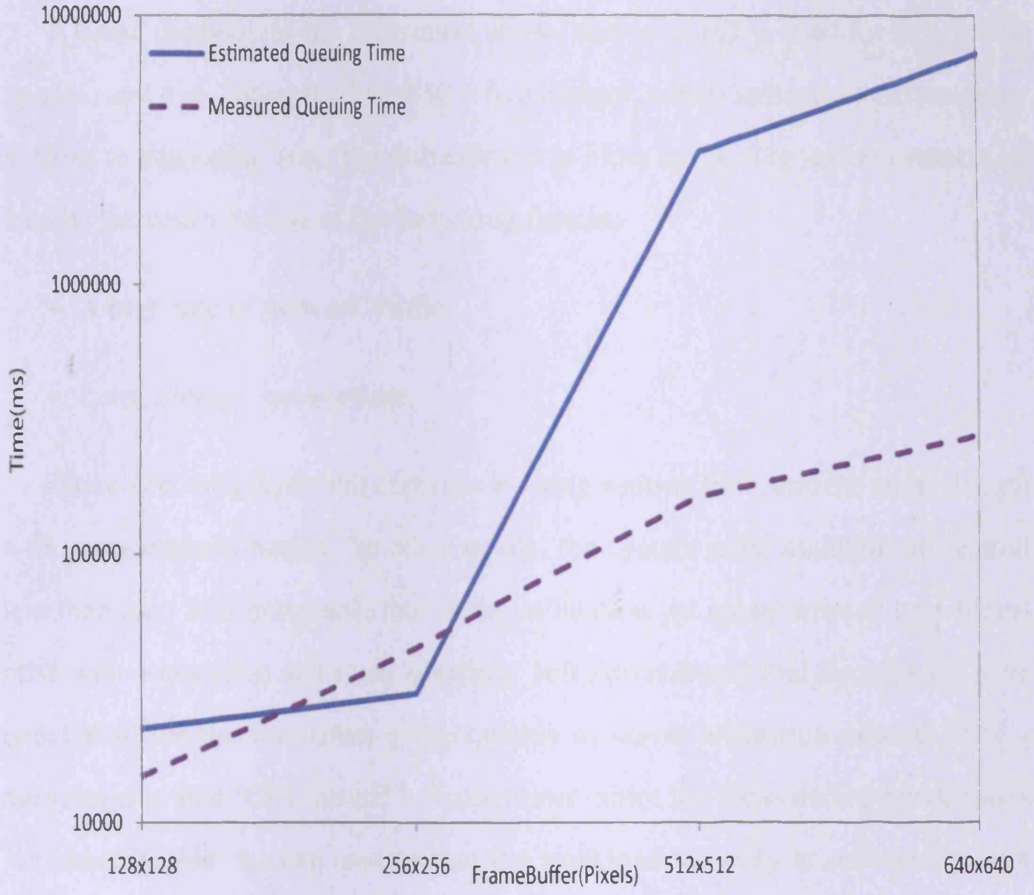


Figure 5.9: Queuing time for local rendering model and measured results for 32 clients.

Dataset Size	Total Model (ms)	Total Experiment (ms)	Relative Error
128×128	22389.25	14878.03	33.55%
256×256	30202.12	44638.72	47.79%
512×512	3122574.13	163717.97	94.75%
640×640	7159261.33	276032.77	96.14%

Table 5.8: Queuing time for local rendering model and measured results for 32 clients.

tabular form in Table 5.8.

A close analysis of the utilization shows that $\rho = 0.7 \pm 0.46$ for 512×512 images, and $\rho = 1.00 \pm 0.46$ for 640×640 images, which indicates that the server is close to saturation (i.e., the utilization ρ is close to 1). The system saturation may be the result on one of the following factors:

- A high rate of network traffic.
- Long average service time.

Therefore, the clients will experience a long waiting time, and the queue length with grow without bound. In other words, the system must maintain utilization less than one. It is noticeable that as the utilization get closer toward 1 problems arise with congestion and stack overflow. Jeff Atwood and Joel Spolsky [5] have concluded that the wait times go up quickly as server utilization exceeds 80% – the system is then "CPU-bound". Furthermore, since the measured response times are unacceptable, we can assume that the workload intensity is sufficiently high that the CPU is approaching saturation. This case the illustrates the importance of performance models in understanding performance data.

When the entire system becomes saturated, no increase in the arrival rate of client requests can be handled successfully. Thus, the throughput bound is the smallest arrival rate, λ, at which any server saturates. Clearly, the server that saturates at the lowest arrival rate is a bottleneck since it is the server with the highest service demand [102].

5.4.7 Arrival Rate λ

Table 5.9 and Fig. 5.10 clarify the relation between arrival rate λ and system saturation. In this experimentation we run the synchronization script with an average arrival rate of 0.136. As the arrival rate increases the response time of the server increases.

Arrival Rate	128 × 128	256 × 256	512 × 512	640 × 640
Experimental	0.0028	0.014	0.0388	0.0683
1	0.00000784	0.000196	0.00150544	0.00466489
10	0.0000784	0.00196	0.0150544	0.0466489
20	0.0001568	0.00392	0.0301088	0.0932978
30	0.0002352	0.00588	0.0451632	0.1399467
40	0.0003136	0.00784	0.0602176	0.1865956
50	0.000392	0.0098	0.075272	0.2332445
60	0.0004704	0.01176	0.0903264	0.2798934
70	0.0005488	0.01372	0.1053808	0.3265423
80	0.0006272	0.01568	0.1204352	0.3731912
90	0.0006272	0.01568	0.1204352	0.3731912
100	0.000784	0.0196	0.150544	0.466489

Table 5.9: Server response times in seconds for different arrival rates for remote rendering.

5.5 Performance Model Prediction

Based on the previous analysis, the numerical differences between the model estimation and the experimental timing data for the three scenarios were generally small. Therefore, these models could be used to estimate the performance for

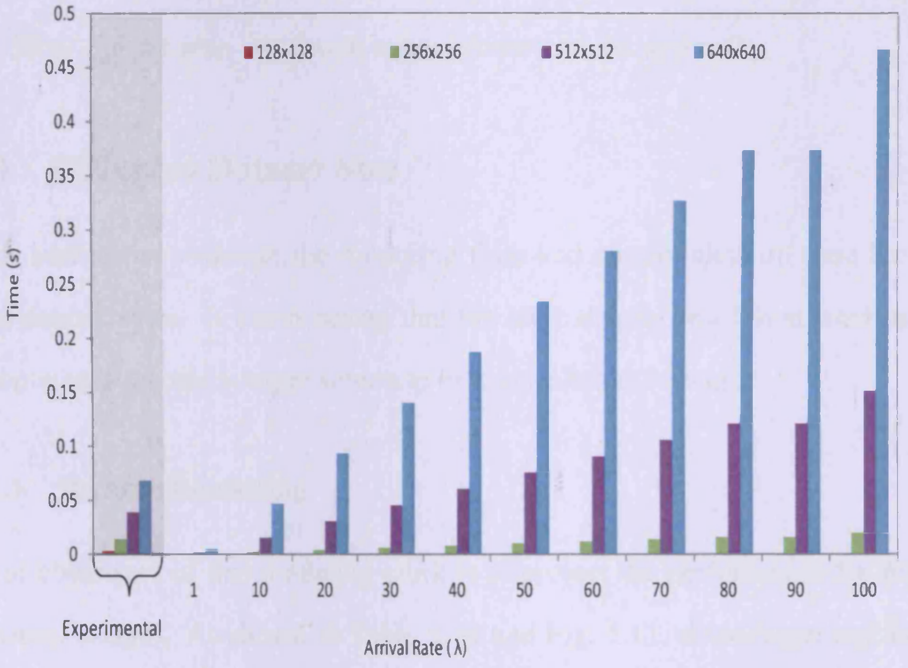


Figure 5.10: Server response times in seconds for different arrival rates for remote rendering.

other scenarios, for example, with different dataset sizes and network bandwidths.

The performance predictions in this section assume that the restriction of our system to use general-purpose Commercially-available Off-The-Shelf (COTS) processing and networking hardware is relaxed.

These types of "what-if" predications are key to answering important performance questions such as:

- What is the performance for different ranges of dataset size?
- What is that effect of a different network bandwidth?
- How can we plan any future enhancements to the system?

5.5.1 Different Dataset Size

In this section we estimate the rendering time and communication time for different dataset sizes. It worth noting that the large dataset resolution used in this section would require a larger screen to best visualize the images. [3]

5.5.1.1 Remote Rendering

One of objectives of this modeling work is to project the performance for higher resolution images. As shown in Table 5.10 and Fig. 5.11, these larger resolution scenarios will not run in at interactive rates as the maximum is 9 frames/seconds (fps). A comparison of the rendering time and communication time shows that each has a similar influence on the overall performance. Improving the rendering time by assigning a more powerfully rendering server, or parallelizing the rendering processing, would boost the frame rate in the 768 $\times$ 768 to a maximum of 20 fps, but for the higher resolutions cases the communication time would still limit the frame rate to a maximum of 10 fps. A more powerful network is essential for interactive frame rates for higher resolutions (1024 $\times$ 1024, 1152 $\times$ 1152, 1536 and 2048 $\times$ 2048).

[3]To best display the resulting frame buffers a 768 $\times$ 768 image requires a 15 inch monitor, 1024$\times$1024 a 17 inch monitor, 1152$\times$1152 a 19 inch monitor, and 1536$\times$1536 and 2048$\times$204821 a 21 inch monitor.

Data-Set Size	Estimated Rendering (ms)	Estimated Communication (ms)	Estimated Overall Time (ms) / FPS
768 × 768	66.99	50.30	117.30 / 9
1024 × 1024	119.10	98.40	217.50 / 5
1152 × 1152	150.74	144.00	294.73 / 3
1536 × 1536	267.97	254.27	522.23 / 2
2048 × 2048	476.38	329.56	805.94 / 1

Table 5.10: Estimated performance for remote rendering for different data-sets resolutions.

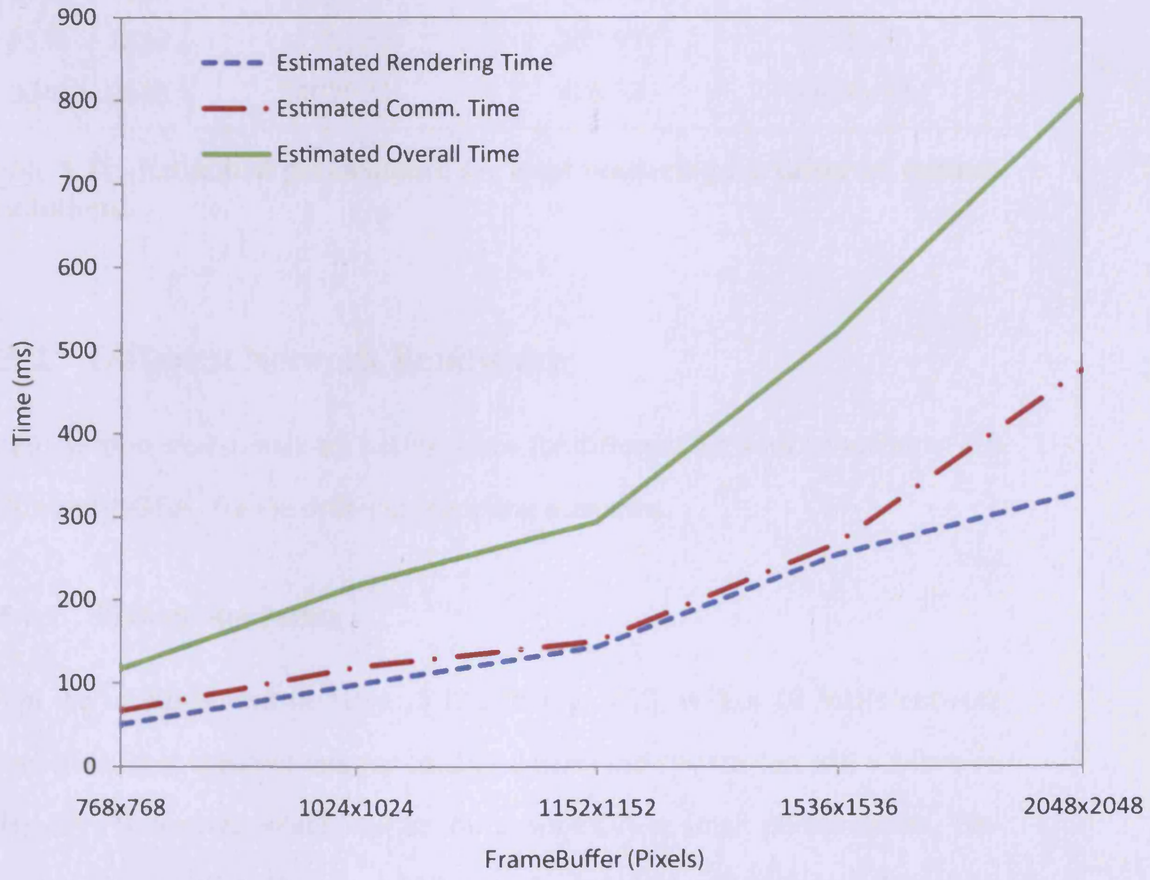


Figure 5.11: Estimated performance for remote rendering for different data-sets resolutions.

5.5.1.2 Local Rendering

As seen from Table 5.11 and Fig. 5.12 in the local rendering scenario the communication costs dominate the overall performance. To enhance this approach a higher-speed network or a dedicated network, is needed.

Data-Set Size	Estimated Communication (ms)	Estimated Rendering (ms)	Estimated Overall Time (ms)
768 × 768	11815.38	67	11882.37
1024 × 1024	21005.13	119.1	21124.22
1152 × 1152	26584.62	150.73	26735.35
1536 × 1536	47261.54	267.97	47529.50
2048 × 2048	84020.51	476.38	84496.90

Table 5.11: Estimated performance for local rendering for different dataset resolutions.

5.5.2 Different Network Bandwidth

In this section we estimate the performance for different network bandwidths (10 MB/s and 1 GB/s) for the different rendering scenarios.

5.5.2.1 Remote Rendering

From the results shown in Table 5.12 and Fig. 5.13, with a 10 MB/s network connection, it is apparent that for smaller dataset the system can still achieve an interactive frame rate, which will be useful for PDA or smart phone clients. However, with larger datasets the performance degrades, for example, to 2.5 fps for a 640 × 640 dataset. We can broadly conclude that 10 Mb/s and 100 Mb/s networks will both achieve interactive frame rates for 128 × 128 and 256 × 256 images.

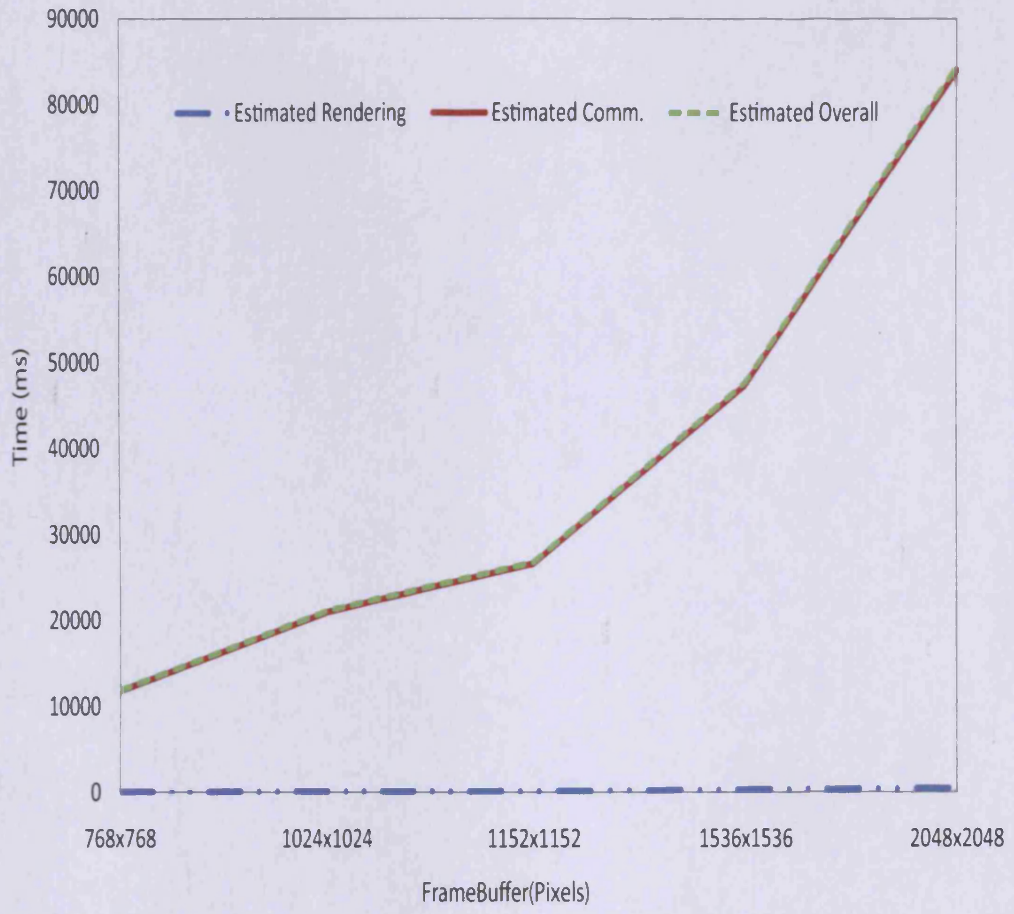


Figure 5.12: Estimated performance for local rendering for different dataset resolutions.

A 1 Gb/s network will not enhance the interactivity for smaller data-set (128×128 and 256×256), because although a higher frame rate is achieved this will not be noticeable to the eye. However, replacing the current 100 MB/s network with a 1 GB/s network is expected to result in interactive frame rates for 512×512 and 640×640 images.

Data-Set Size	Estimated 10MB ms / FPS	Measured 100MB ms / FPS	Estimated 100MB ms / FPS	Estimated 1GB ms / FPS
128x128	16.97 / 59	5.6 / 178.5	5.43 / 184	4.28 / 234
256x256	67.63 / 15	24.2 / 41	21.48 / 46.5	16.86 / 59
512x512	245.28 / 4	57.8 / 17	60.66 / 16	42.20 / 24
640x640	390.16 / 2.5	97.2 / 10	101.70 / 10	72.85 / 14

Table 5.12: Table of estimated remote rendering performance for different network bandwidths.

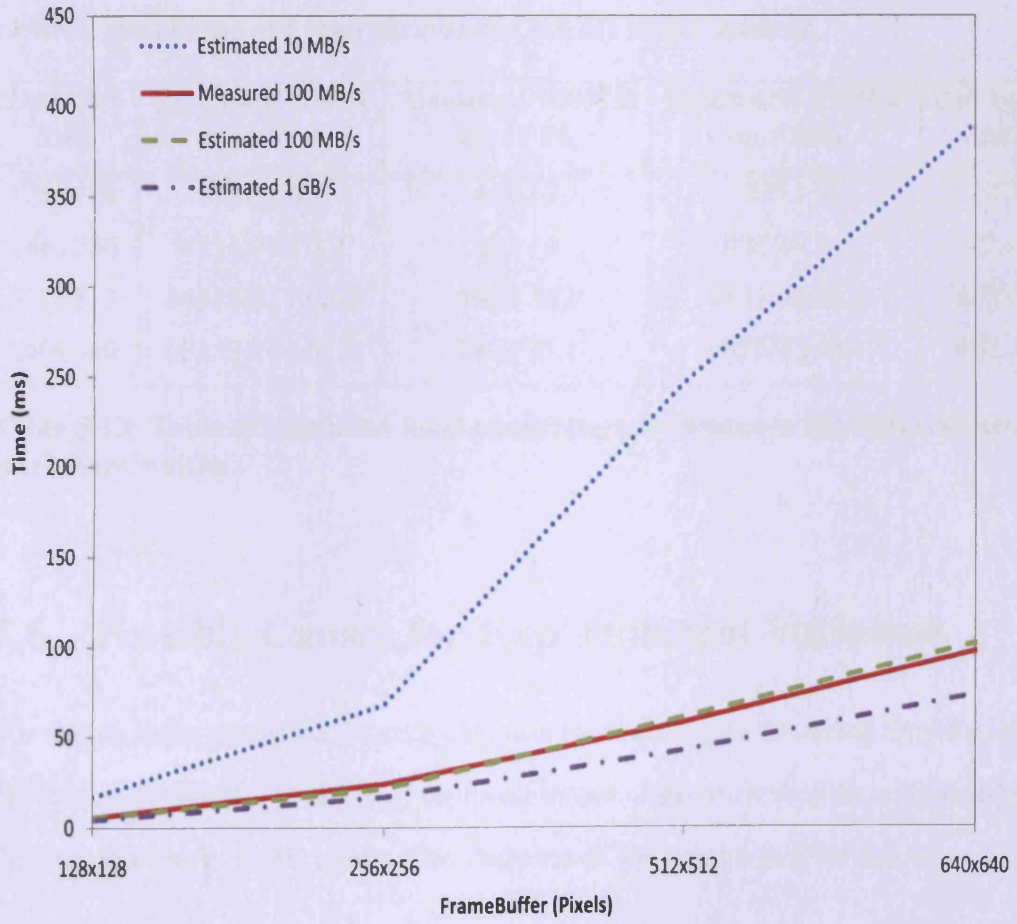


Figure 5.13: Remote rendering for different networks bandwidths.

5.5.2.2 Local Rendering

From the results shown in Table 5.13 and Fig. 5.14, with a 10 MB/s connection, the clients have to wait a longer period (3 to 63 seconds) before starting to navigate the dataset. While the 1 GB/s network dramatically enhances the interactivity for smaller datasets (128 $\times$ 128 and 256 $\times$ 256, the larger datasets still achieve only 2 to 3 frames per second. Replacing the current 100 MB/s network with a 1 GB/s network will not produce a interactive rate for 512 $\times$ 512 and 640 $\times$ 640 images. As a result, upgrading the network to 1 GB/s will be useful for smaller dataset viewers, such PDAs and smartphones, but not for larger datasets.

Data-Set Size	Estimated 10MB ms / FPS	Measured 100MB ms / FPS	Estimated 100MB ms / FPS	Estimated 1GB ms / FPS
128x128	3750 / 0.3	415 / 2	375 / 3	37.5 / 27
256x256	4214.74 / 0.2	482 / 2	421.47 / 2	42.15 / 24
512x512	44318.91 / 0.02	4901 /0.2	4431.89 /0.2	443.19 / 2.3
640x640	68253.21 / 0.01	7421 /0.1	6825.32 /0.1	682.53 / 1.5

Table 5.13: Table of estimated local rendering performance for different network bandwidths.

5.6 Possible Causes for Experimental Variation

The timing value presented for each data point is the average rendering time for ten different viewpoints, (V_x, V_y, V_z), repeated in five experimental trials representing the user movement in 3D space. The variance of the results may be due to:

- Interference from other programs or tasks. Other users may compete for shared resources such as processors, network bandwidth, and I/O band-

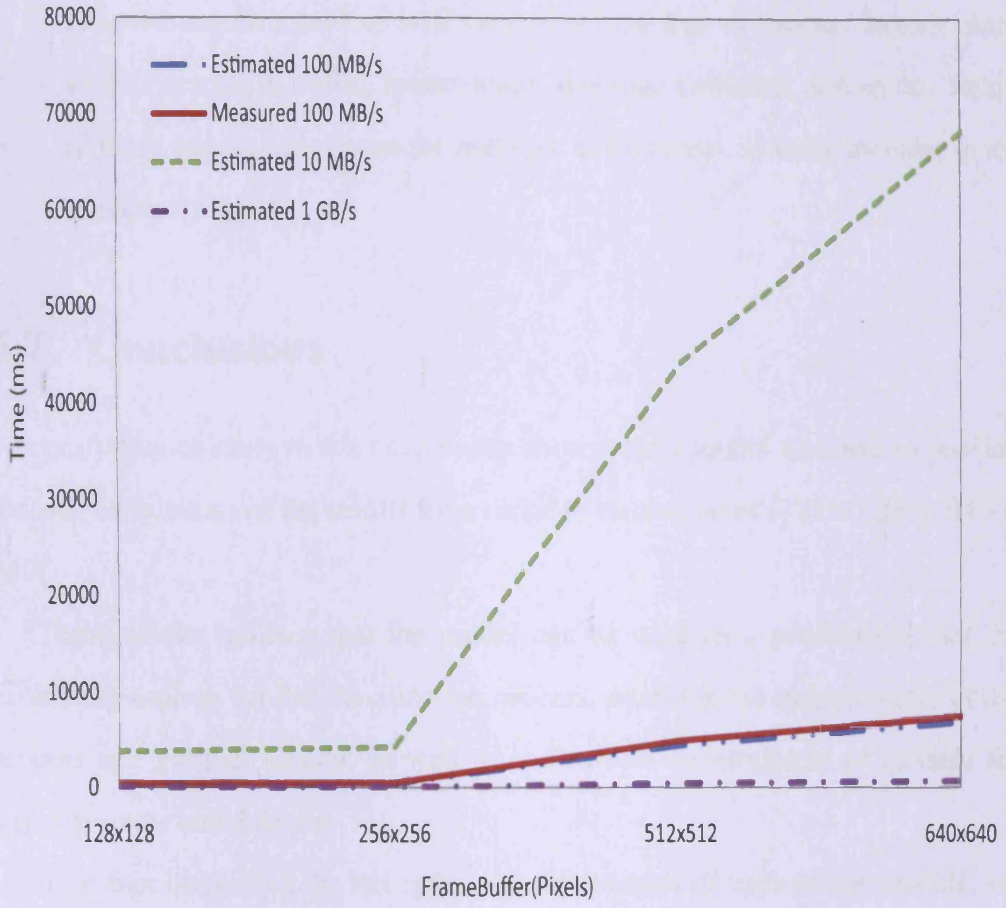


Figure 5.14: Local rendering for different networks bandwidths.

width. Note we run our experiments during off-peak times when no other users are directly logged on. However based, on the fact that these clients are open access other users may be remotely logged in and there might be background activities running which effect the overall performance.

- The server processor allocation. The operating system may use either of the two processors which will affect the repeatability of the execution time.

- Transient delays and variability. The amount of time it takes to send data between any two devices will vary over time due to various factors, such as fluctuations in traffic, router loads, message collision, and so on. Some of these factors may cause the message to be resent, thereby increasing the execution time.

5.7 Conclusions

The performance study in this chapter has shown that a model was able to provide realistic estimations of the results for a range of dataset sizes (128×128 to 640×640).

These results indicate that the model can be used as a predication tool for estimating timings for the visualization process, enabling the improvement of the process and product quality, as well as the further development of models for larger systems and datasets.

In further discussing the strengths and weaknesses of each of the models, we see that to be able to run the system for larger dataset resolution involves a trade-off between generality of hardware (the server and network) and dataset resolution. Larger dataset resolution cannot achieve interactive frame rates on current COTS resources.

The model has inaccuracies for small datasets because the rendering time of 1 to 3 ms is comparable to the timing resolution of 1 ms.

5.8 Future Work

The present results suggest that additional development is necessary to increase the accuracy for model prediction of the remote visualization system and to model performance across a broad range of circumstances. Examples of the modifications that need to be added to the current models involve variable service time, as we have assumed that all users visualize the same dataset, so the service time is for that particular dataset. Also we could add a multicore-based server which could be represented by a multi-server approach. Our system currently processes viewpoint queries from a set recorded by real user navigation. A mechanism for predicting the future navigation path could be used based on probabilistic prediction techniques, such as Kalman filtering [65]. We expect such techniques for pre-fetching future images would enhance the system performance in terms of interactivity and frame rates.

New experimental data will need to be acquired to elucidate this matter, and to provide information for the development of future performance models.

5.9 Chapter Summary

This chapter discussed a theoretical cost model for using the three different rendering scenarios in a remote visualization system. This set of models was compared with the experimental results presented in the previous chapter and enables a validation of these results. Furthermore, the models provide a prediction tool to estimate the performance in scenarios where some parameters are different from those examined experimentally. In addition, we have discussed the strengths and weaknesses of each of the models.

Chapter 6

Conclusions

Overview

This chapter contains a review of the work that has been detailed throughout this thesis. An overall critical analysis of the hypothesis presented in Chapter 1 is given. Furthermore, this chapter provides answers to the research questions investigated based on the objectives, and draws conclusions based on the main findings of the research. Then we summarize the main conclusions of the thesis, and the limitations of the work are presented.

Section 6.1 presents an overview of the issues involved in the design and evaluation of the system. Then Section 6.2 presents a critical evaluation of the system. We summarize the research contributions made in this thesis in Section 6.3, and describe the limitations of the work in Section 6.5.

6.1 Summary

Interactive distributed visualization has emerged as a discipline with numerous applications. However, many of the present approaches to interactive distributed visualization have limited performance since they are based on the traditional polygonal processing graphics pipeline. In contrast, image-based rendering uses multiple images of the scene instead of a 3D geometrical representation, and so has the key advantage that the final output is independent of the scene complexity and depends on the desired final image resolution. Furthermore, the discrete nature of the Light Field dataset maps well to a hybrid solution which can overcome the identified drawbacks. In this thesis we have described a method for efficiently and effectively transmitting visualization data to remote users/clients. Instead of downloading a complete replica of the Light Field dataset to each client, or remotely sending a single rendered view back from a central server to the user each time the user updates their viewing parameters, our on-demand strategy sends parts of the dataset based on the current client viewpoint. The on-demand approach shows stable performance as the number of clients increases because the load on the server and the network traffic are reduced.

6.2 Critical Evaluation

The research hypothesis we proposed initially in Chapter 1 was as follows:

3D collaborative visualization environments, created by distributed light field rendering techniques with tunable performance parameters, provide a generic, highly interactive, remote 3D visualization solution for large numbers of users, *enabling user controlled viewpoints for both real and synthetic data.*

The success of the research discussed in this thesis can be measured by the extent to which the objectives of the research mentioned in Chapter 1 have been achieved. Therefore, in order to validate this hypothesis we will look to the individual components/aims.

To fulfill the above hypothesis a system for 3D collaborative visualization was developed by using distributed light field rendering techniques aimed at the following:

- **Generic 3D remote visualization solution for large numbers of users**

  In order to accommodate end-users with varying degrees of expertise and different background knowledge of visualization our system hides many of the implementation details from the user by providing a high level of abstraction. Our system provides a 3D visualization environment for clients through the *Viewer Process* client module. The simple interface provides all the functionality necessary to explore the dataset for all viewpoints. The user can steer the visualization through the control of viewing parameters using different input devices (mouse/keyboard).

- **Platform-independent visualization environment**

  Most of the current visualization tools that run on multiple platforms are not truly platform-independent as different versions are provided for each platform. However, the use of the Qt software makes visualization applications cross-platform. Although Qt is available as freeware for Linux and Mac platforms, Windows users must pay for a licence. This is could be resolved by using the free version of Qt [2] to create an open-source software distribution, or to execute the software as a compiled binary. The software developed in this thesis is open-source and freely available, thus avoiding the licensing requirements of other similar tools and commercial products.

- **High interactivity and stability**

  To evaluate our system, we use interactivity metrics as a standard method to measure the response of the visualization system to user input. As shown in Fig. 4.5 the overall performance in frames per second for the different rendering scenarios demonstrates the client-side and server-side capabilities of the system.

  A second aspect of interactivity concerns the the stability of the performance measured in frames per second. Such knowledge indicates how interactivity might change when users explore different viewpoints for complex datasets. This information is also useful in predicting the performance for other dataset sizes. Figure 4.1 demonstrates the stability of image-based rendering approach.

- **Distributed light field rendering techniques** The system uses different distributed rendering transmission models: local, remote, and on-demand. Figures 4.6 and 4.7 illustrate the system behaviour under different scenarios, and shows which scenario performs better for a given situation in terms of enhancing the overall system performance. The on-demand approach shows higher stability in the performance as the number of clients increases because the load on the server, and the network traffic, are reduced.

- **Enabling user-controlled viewpoints for both real and synthetic data**

Other sources of image data could be fed into the system. These data could be generated using either a multi-array camera [6] or a gantry [9] for real objects, or it is created for a synthetic object using a modified ray tracing algorithm. Regardless of the source of the input dataset, the system should work on both real and synthetic data. Figure 3.2 shows the output for a real object (a dragon statue), and Fig. 3.3 shows the output for a synthetic object (a skeleton scan).

6.3 Contributions

The contributions of this thesis are as follows:

- **The design and implementation of a remote visualization system using light field rendering.** Chapter 3 proposes a design and implementation for building a collaborative visualization environment for geographically distributed users that involves complex datasets, using a commodity hardware and software environment. The design provides a novel combination of light field techniques and a transmission model which provides a general-purpose interactive solution for distributed collaborative visualization.

- **An on-demand transmission model for transmitting datasets based on the user/client viewpoint parameters.** Chapter 3 discusses the details of the on-demand model in which, instead of downloading a complete copy of the light field dataset to each client, or remotely sending a single rendered light field view back from a central server to the user each time a user updates their viewing parameters, both approaches are combined. In response to a viewing query, initially the client cache is checked for the required images, and if an image is not available it is retrieved from the server, and interpolated. The benefits of this approach is that excessive use of storage is avoided by downloading only partial datasets. Furthermore, it better utilizes the rendering capability of the clients and minimizes the server load, which will reduce the overall network traffic.

- **A performance study for system behaviours for three different transmission models under different viewpoints, dataset sizes, and viewers.** Chapter 4 presents a performance study of the effects of deploying an on-demand transmission model. In addition, it presents a study of the impact on performance of different factors, including the number of visualization clients, the dataset resolution, and the interpolation scheme.

- **A theoretical cost model for local rendering, remote rendering, and on-demand rendering.** These models are presented in Chapter 5, and provide a comparison with the experimental results presented in Chapter 4. Furthermore, these models models provide a predicative tool to project performance in terms of computation and communication time for parameters are different from those used in the performance experiments presented in Chapter 4.

6.4 Conclusions

We have described a software architecture for using an on-demand transmission model using light field rendering within a distributed collaborative visualization environment. The system makes use of commodity networking, hardware, and software for distributed visualization. We have discussed the existing local and remote transmission models, and compared them with our on-demand model.

Our results show that a light field rendering system enables constant frame rates independent of the scene complexity compared with geometry-based visualization, where the frame rate depends on the complexity of the scene for the chosen viewpoint. The efficiency of the system increases as the scene complexity increases. The applicability of the system covers a large number of application areas, where interactive collaboration could enhance and accelerate navigation and discovery.

Our system currently provides a generic on-demand remote rendering solution for users to access visualization services using computationally inexpensive light field rendering, allowing low-end devices, such as PDAs, to interactively explore 3D objects.

The on-demand approach shows stable performance as the number of clients increases because the load on the server and the network traffic are reduced. The main performance bottleneck arises from the limited transfer rate across the network, which can be improved with the availability of higher performance networks.

6.5 Limitations of the work

Overall, we believe the design of our distributed visualization system provides a good basis for future development, particularly in the following areas:

- **Centrality.** As with all single server systems there is a potential problem because the server is a single point of failure (SPOF). Any server failure will stop the whole system from working. One solution to enhance robustness is to use multiple and/or redundant servers.

- **Access control.** For collaboration and distributed systems security is one of the key issues. The identities or credentials used in one organisation might not be recognized in other organisations. The solution could be that either all organisations use certificates authorized by the same trusted Certificate Authority (CA), or a mapping mechanism can be used to map certificates across different organisations. The security across the boundaries of different organisations is outside the scope of the research presented in this thesis.

- **Synchronized Collaboration.** In our system each visualization client or viewer works independently. Clients send their viewing requests separately to the rendering server. The rendering server processes the requested views concurrently and sends the resulting framebuffers to the clients. This scenario represents collaboration in space and time (asynchronous collaboration) since participants could be located in different places at different times. In other situations users might want to share the same viewpoint simultaneously. This requires significant synchronization time between all processes to ensure that all views are consistent. This mode of use is called *synchronized collaboration.*

6.6 Chapter Summary

This chapter contains a review of the work that has been presented in this thesis. An overall critical analysis of the hypothesis presented in Chapter 1 has been given, together with the main highlights and findings of this research. We then draw the main conclusions of the thesis, and discuss the limitations of the work.

Chapter 7

Future Work

Overview

In this chapter, we briefly examine some of the future directions that the work of this thesis could be expanded into. These directions could be broadly classified into enhancing performance, experimentation, accessibility, collaboration experience, and security. A more detailed discussion of these issues is presented in Sections 7.1 to 7.7. Finally, a summary of the chapter is presented in Section 7.8.

7.1 Enhancing Performance

There are various directions for future research, especially if developers and users are willing to relax the conditions mentioned in the previous chapter, such as requiring the use of Commodity-Off-The-Shelf hardware and software.

7.1.1 Data Partitioning and Parallelism

With very large datasets (several GB) performance could be enhanced by partitioning the dataset between several PCs and using a master node to store an index of the location of each part of the dataset. A multiple-node server approach can also be used within the testbed to distribute the large datasets to multiple machines within a cluster according to their viewpoints. This will permit further performance studies to be undertaken for the on-demand approach for comparison with the current results.

7.1.2 Viewpoint Prediction

Our system currently processes viewpoints queries from a set recorded by real user navigation. Caching of images can be used to enable a client to download images into a local image cache for replay. The "momentum" of the camera position associated with interactive user navigation affords the opportunity for the client to predict, pre-fetch and cache the anticipated images on the server. A mechanism for predicting the future navigation path could be used based on probabilistic prediction techniques, such as Kalman filtering. We expect that the use of such techniques for pre-fetching future images would enhance the system performance in terms of interactivity and frame rate.

7.1.3 Dataset Compression

Compression could be used to enhance image data transmission rates. Although for certain dataset, such as medical dataset images, lossy compression is undesirable as every single detail is important, other applications could make use of such techniques, for example by using the Zlib library. However, other more powerful compression algorithms, such as [69], are computationally expensive and unsuitable for real-time computation

7.2 Experimental Extendibility

It would be interesting to enhance the distribution capability of our system by extending the experimental testbed to accommodate approximately 100 concurrent users. The Linux lab used in the work of this thesis is limited to 32 workstations, however, Cardiff's Condor pool could be used to expand the size of the testbed. One of the challenges in this case would be to overcome the scheduling of Condor as all the jobs have to processed through the Condor system.

7.3 Enhancing Accessibility

With the increasing number of different datasets and their associated attributes, attaching a meta-data description layer would efficiently enhance the time to extract and visualize a given dataset.

7.4 Enhancing Reliability

Expanding the model for to use a multi-server approach would add fault tolerant mechanisms to the current system, and it would be of interest to investigate the coordination of such a system.

7.5 Include Dynamic Live Data

Professor Takeshi Naemura from the University of Tokyo and Hibachi has presented a system that provides a real-time 3D visual experience by using an array of 64 video cameras and an integral photography display with 60 viewing directions [80, 98]. The live 3D scene in front of the camera array is reproduced in a full-colour, full-parallax autostereoscopic display with interactive control of viewing parameters. An interesting extension would be to expand our system to a distributed environment capable of handling and managing data management.

7.6 Enhance the Collaboration Experience

7.6.1 Video and chat utility

As the users may be located at different locations, video streaming and text chatting could be added to the visualization system to enhance user interaction. Such a scheme is be based on AccessGrid [26].

7.6.2 Collaboration Scenarios

A useful extension to the system would be to allow collaborators to interact in synchronized mode. Example of such collaboration are :

- Multiple Viewers and a Single Controller. In this case (see Fig. 7.1) the general aim is to provide a shared visualization view for all the participants provided by a visualization controller node. The visualization controller is the only participant with privileges to select view points. Clients in this case are passive viewers since they cannot change the viewing parameters. After selecting a new viewpoint, a request is sent to the rendering server to be processed, and the resulting framebuffer is multicast to the visualization clients. This represents space collaboration, where users could be located in geographically distributed places, and could be included in our system (where users each have their own viewpoint window) by providing another window showing the controller visualization. Another way is to check the availability of the controller; in this case it works as our system scenario.

- Multiple controllers. In this scenario (see Fig. 7.2) all the visualization participants are of the controller client type, and all have the same privileges to change the shared viewpoint. After a controller process selects a new view, this would be sent to all the other controllers. This requires significant synchronization between all processes to ensure that all views are consistent. It is a representation of space collaboration.

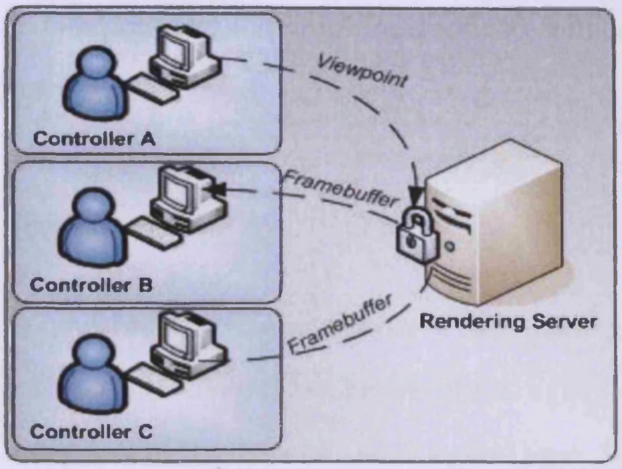


Figure 7.1: Multiple controllers scenario.

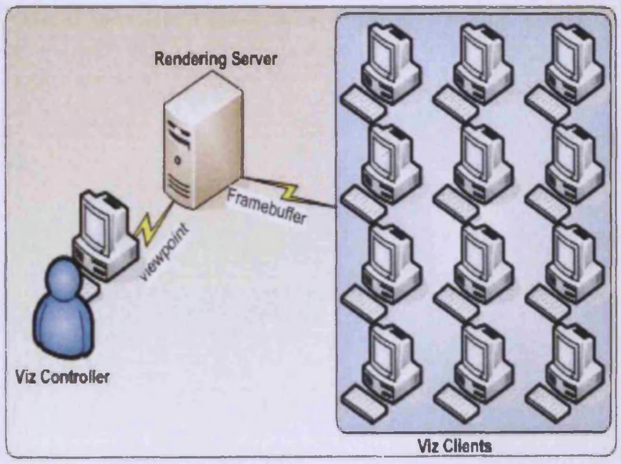


Figure 7.2: Multiple viewers and a single controller.

7.7 Enhancing Security

For collaboration in distributed systems security is one of the most important issues. The identities or credentials used in one organisation might not be recognized in other organisations. The solution could be that, either all organisations use certificates authorized by the same trusted Certificate Authority (CA), or a mapping mechanism is used to map certificates across different organisations. The security across the boundaries of different organisations is outside the scope of the research presented in this thesis.

7.8 Final Remark

In this chapter, we have briefly examined some of the future directions that the work could be expanded into. Overall, we believe the design of our visualization system provides a step forward in the area and is a good basis for future development. With future development as discussed above, this system would become more useful, and could be integrated with real users and applications.

Bibliography

[1] *David George Kendall Home Page*, http://www-groups.dcs.st-and.ac.uk/~history/Biographies/Kendall.html, last accessed April 2011.

[2] *Open Source Versions of Qt*, http://doc.qt.nokia.com/latest/opensourceedition.html, last accessed May 2011.

[3] *Amira*, http://http://www.amiravis.com/, last accessed October 2010.

[4] *Globus: Fundamental technologies needed to build computational grids.*, http:http://www.globus.org., last accessed October 2010.

[5] *The podcast, stackoverflow, episode 39*, https://stackoverflow.fogbugz.com/default.asp?W29026, last accessed May 2011.

[6] *The Stanford Multi-Camera Array*, http://graphics.stanford.edu/projects/array/, last accessed June 2010.

[7] *SGI VizServer*, http://www.sgi.com/products/software/vizserver/, last accessed June 2010.

[8] *Qt Library*, http://qt.nokia.com/, last accessed June 2010.

[9] *The Stanford Spherical Gantry*, http://window.stanford.edu/projects/gantry/, last accessed June 2010.

[10] *RealVNC Software*, http://www.realvnc.com/index.html, last accessed Januray 2011.

[11] *TightVNC Software*, http://www.tightvnc.com/, last accessed Januray 2011.

[12] *VTK: the Visualization Toolkit*, http://www.vtk.org/, last accessed July 2010.

[13] *Cactus Computational Toolkit*, `http://www.cactuscode.org/`, last accessed August 2010.

[14] *cURL*, `http://curl.haxx.se/`, last accessed June 2010.

[15] *flickr*, `http://www.flickr.com/`, last accessed October 2010.

[16] *NetMeeting*, `http://www.microsoft.com/downloads/details.aspx?FamilyID=26c9da7c-f778-4422-a6f4-efb8abba021e&displaylang=en#Overview`, last accessed Auguest 2010.

[17] *Office Live Meeting*, `http://office.microsoft.com/en-us/live-meeting/`, last accessed Auguest 2010.

[18] *Quicktime vr*, `http://developer.apple.com/legacy/mac/library/documentation/QuickTime/InsideQT_QTVR/insideqt_qtvr.pdf`, last accessed October 2010.

[19] *Virtual Gl Project*, `http://www.virtualgl.org/`, last accessed November 2010.

[20] *VisIt*, `https://wci.llnl.gov/codes/visit/`, last accessed Auguest 2010.

[21] *Yahoo Video*, `http://video.yahoo.com`, last accessed October 2010.

[22] *IRIS Explorer*, `http://www.nag.co.uk/visual/IE/iecbb/Product.html`, last accessed March 2010.

[23] *ParaView*, `http://www.paraview.org/`, last accessed Auguest 2010.

[24] *SciRun*, `http://software.sci.utah.edu/scirun.html`, last access March 2010.

[25] *VisMockUP*, `http://www.softscout.com/software/Engineering/Computer-Aided-Design-CAD/VisMockUp.html`, last accessed June 2010.

[26] *Accessgrid*, `http://www.accessgrid.org/`, last accessed February 2011.

[27] *ibmonitor Software*, `http://ibmonitor.sourceforge.net/`, last accessed June 2011.

[28] E. H. Adelson and J. R. Bergen, *The plenoptic function and elements of early vision*, Computational Models of Visual Processing (1991), 3–20.

[29] A. Al-Saidi, N. J. Avis, I. J. Grimstead, and O. F. Rana, *Distributed collaborative visualization using light field rendering*, CCGRID, 2009, pp. 609–614.

[30] Asma Al-Saidi, Nick J. Avis, Omer F. Rana, and Abdelhamid Abdesselam, *On-demand transmission model using image-based rendering for remote visualization*, UK e-Science All Hands Meeting, 7th -9th December 2009.

[31] Gabrielle Allen, Werner Benger, Thomas Dramlitsch, Tom Goodale, Hans-Christian Hege, Gerd Lanfermann, André Merzky, Thomas Radke, Edward Seidel, and John Shalf, *Cactus tools for grid applications*, Cluster Computing **4** (2001), 179–188.

[32] Ronnie T. Apteker, James A. Fisher, Valentin S. Kisimov, and Hanoch Neishlos, *Video acceptability and frame rate*, IEEE MultiMedia, **vol. 2** (1995), 32–40.

[33] Micah Beck, Terry Moore, and James S. Plank, *An end-to-end approach to globally scalable network storage*, Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications (New York, NY, USA), SIGCOMM '02, ACM, 2002, pp. 339–346.

[34] E.W. Bethel, *Visualization dot com*, IEEE Comput. Graph. Appl. **vol. 20** (2000), no. 3, 17–20.

[35] Richard E. Blahut, *Algebraic codes for data transmission*, 1 ed., Cambridge University Press, July 2002.

[36] K. Brodlie, J. Brooke, M. Chen, D. Chisnall, A. Fewings, C. Hughes, N. W. John, M. W. Jones, M. Riding, and N. Roard, *Visual supercomputing: Technologies, applications and challenges*, Computer Graphics Forum **24** (2005), no. 2, 217–245.

[37] K. W. Brodlie, D. A. Duce, J. R. Gallop, J. P. R. B. Walton, and J. D. Wood, *Distributed and collaborative visualization*, Computer Graphics Forum **23** (2004), no. 2, 223–251.

[38] Gordon C., *Modular visualization environments: past, present, and future*, SIGGRAPH Comput. Graph. **vol. 29** (1995), no. 2, pp. 3–4.

[39] S. M. Charters, *Visualization for eresearch: Past, present and future*, eResearch Australasia, 2008.

[40] S. E. Chen, *Quicktime vr: an image-based approach to virtual environment navigation*, SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques (New York, NY, USA), ACM, 1995, pp. 29–38.

[41] Daniel Cohen-Or and Eyal Zadicario, *Visibility streaming for network-based walkthroughs*, Graphics Interface (1998), 1–7.

[42] Matthias Kalle Dalheimer, *Qt vs. Java A comparison of Qt and Java for Large-Scale, industrial-strength gui development*, `http://turing.iimas.unam.mx/~elena/PDI-Lic/qt-vs-java-whitepaper.pdf`, last accessed October 2010.

[43] John Domingue, Blaine Price, and Marc Eisenstadt, *A framework for describing and implementing software visualization systems*, Proceedings of the conference on Graphics interface '92 (San Francisco, CA, USA), Morgan Kaufmann Publishers Inc., 1992, pp. 53–60.

[44] G. Scott Graham Kenneth C. Sevcik Edward D. Lazowska, John Zahorjan, *Quantitative system performance computer system analysis using queueing network models*, Prentice-Hall, New Jersey, 1984.

[45] Ian Foster and Carl Kesselman, *Globus: A metacomputing infrastructure toolkit*, International Journal of Supercomputer Applications **11** (1996), 115–128.

[46] *Frames per second*, `http://www.100fps.com/how_many_frames_can_humans_see.htm`, last accessed May 2011.

[47] Michael Friendly, *Milestones in the history of thematic cartography, statistical graphics, and data visualization*, (2009), `http://datavis.ca/milestones/`.

[48] J. A. Friesen and T. D. Tarman, *Remote high-performance visualization and collaboration*, IEEE Computer Graphics and Applications **vol. 20** (2000), no. 4, pp. 45–49.

[49] A. Glassner, *An introduction to ray tracing*, First edition, Morgan Kaufmann, 1989.

[50] S. J. Gortlet, R. Grzeszczuk, R. Szeliski, and M. F. Cohen, *The lumigraph*, Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, 1996, pp. 43–54.

[51] N. Greene, *Environment mapping and other applications of world projections*, IEEE Computer Graphics and Applications **vol. 6** (1986), no. 11, pp. 21–29.

[52] I. J. Grimstead, N. J. Avis, and D. W. Walker, *Automatic distribution of rendering workloads in a grid enabled collaborative visualization environment*, SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing (Washington, DC, USA), IEEE Computer Society, 2004, p. 1.

[53] ______, *Visualization across the pond: how a wireless pda can collaborate with million-polygon datasets via 9,000km of cable*, Web3D '05: Proceedings of the tenth international conference on 3D Web technology (New York, NY, USA), ACM, 2005, pp. 47–56.

[54] ______, *Rave: the resource-aware visualization environment*, Concurrency: Practice and Experience **vol. 21** (2008), no. 4, pp. 415–448.

[55] I.J. Grimstead, D. W. Walker, N. J. Avis, F. Kleinermann, and J. McClure, *3d anatomical model visualization within a grid-enabled environment*, Computing in Science and Engg. **9** (2007), no. 5, 32–38.

[56] I.J. Grimstead, D.W. Walker, and N.J. Avis, *Collaborative visualization: a review and taxonomy*, Distributed Simulation and Real-Time Applications, 2005. DS-RT 2005 Proceedings. Ninth IEEE International Symposium on, 10-12 2005, pp. 61 – 69.

[57] R. B. Haber and D. A. McNabb, *Visualization idioms: A conceptual model for scientific visualization systems*, Visualization in Scientific Computing.

[58] O. Hagsand, *Interactive Multiuser VEs in the DIVE System*, IEEE MultiMedia **vol. 3** (1996), no. 1, pp. 30–39.

[59] Wernert E.A. Hanson, A.J. and S.B. Hughes, *Constrained navigation environments*, Scientific Visualization Conference, 1997, 1997, p. 95.

[60] B. Hibbard, *Top ten visualization problems*, SIGGRAPH Comput. Graph. **33** (1999), no. 2, 21–22.

[61] Carl Kesselman Ian Foster, *The grid: Blueprint for a new computing infrastructure (the elsevier series in grid computing)*, 1st ed., Morgan Kaufmann, August 12 1998.

[62] J. Harting S. Jha S. M. Pickles R. L. Pinning J. M. Brooke, P. V. Coveney and A. R.Porter, *Computational steering in realitygrid*, in Proceedings of the UK e-Science, All Hands Meeting 2003, 2003.

[63] Bahram Javidi, *Three-dimensional television, video and display technology*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[64] D. Jin, H. Jian, B. Micah, L. Shaotao, M. Terry, and S. Stephen, *Remote visualization by browsing image based databases with logistical networking*, 2003, 105018534.

[65] R. E. Kalman, *A new approach to linear filtering and prediction problems*, Journal of Basic Engineering **82** (1960), no. 1, 35–45.

[66] L. Kleinrock, *Queueing systems*, vol. I : Theory, Wiley-Interscience, New York, USA, 1975.

[67] R. Kosara, F. Drury, L.E. Holmquist, and D.H. Laidlaw, *Visualization criticism*, Computer Graphics and Applications, IEEE **28** (2008), no. 3, 13 -15.

[68] Wim Lamotte, Eddy Flerackers, Frank Van Reeth, Rae Earnshaw, and Joao Mena De Matos, *Visinet: Collaborative 3d visualization and vr over atm networks*, IEEE Comput. Graph. Appl. **17** (1997), no. 2, 66–75.

[69] Eung-Seok Lee and Hyeong-Seok Ko, *Vertex data compression for triangular meshes*, Proceedings of the 8th Pacific Conference on Computer Graphics and Applications (Washington, DC, USA), PG '00, IEEE Computer Society, 2000, pp. 225–.

[70] M. Levoy and P. Hanrahan, *Light field rendering*, SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques (New York, NY, USA), ACM, 1996, pp. 31–42.

[71] J.D.C. Little, *A proof of the queueing formula $l = \lambda\, w$*, Operations Research **9** (1961), 383–387.

[72] Lici Lu., *Resource management for a campus computational grid.*, MasterŠs thesis, University of Adelaide, 2002.

[73] Ioana M. Martin, *Hybrid transcoding for adaptive transmission of 3d content*, In Proceedings of IEEE International Conference on Multimedia and Expo (ICME, Press, 2002, pp. 373–376.

[74] Wojciech Matusik and Hanspeter Pfister, *3d tv: a scalable system for real-time acquisition, transmission, and autostereoscopic display of dynamic scenes*, ACM Trans. Graph. **23** (2004), 814–824.

[75] B. H. McCormick, *Visualization in scientific computing*, SIGBIO Newsl. **10** (1988), 15–21.

[76] L. McMillan and G. Bishop, *Plenoptic modeling: an image-based rendering system*, SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques (New York, NY, USA), ACM, 1995, pp. 39–46.

[77] Gordon E Moore, *Cramming more components onto integrated circuits*, Electronics Magazine **38** (1965), 4.

[78] Jesper Mortensen, Pankaj Khanna, and Mel Slater, *Light field propagation and rendering on the gpu*, Proceedings of the 5th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa (New York, NY, USA), AFRIGRAPH '07, ACM, 2007, pp. 15–23.

[79] Jesper Mortensen, Pankaj Khanna, Insu Yu, and Mel Slater, *Real-time global illumination in the cave*, Proceedings of the 2007 ACM symposium on Virtual reality software and technology (New York, NY, USA), VRST '07, ACM, 2007, pp. 145–148.

[80] Takeshi Naemura, *The 64-array answer*, The Japan Journal **5** (2009), 24–25.

[81] S. Parker, W. Martin, P.-P. J.Sloan, P. Shirley, B. Smits, and C Hansen, *Interactive ray tracing*, In Symposium on interactive 3D graphics, 1999, pp. 119–126.

[82] S. Parker, M. Parker, Y. Livnat, P. P. Sloan, C. Hansen, and P. Shirley, *Interactive ray tracing for volume visualization*, IEEE Transactions on Visualization and Computer Graphics, **vol. 5** (1999), no. 3, pp. 238–250, 1077-2626.

[83] P. Peers, *Sampling reflectance functions for image-based relighting*, Ph.D. thesis, Katholieke Universiteit Leuven, 2006.

[84] F. Perrin, *Manufacturing: The video advantage*, first edition ed., Enigma Publishing Ltd, Autumn 1997, available through `http://www.mcb.co.uk/portfolio/nac/imds/books.htm`.

[85] James S. Plank, Alessandro Bassi, Micah Beck, Terence Moore, D. Martin Swany, and Rich Wolski, *Managing data storage in the network*, IEEE Internet Computing **5** (2001), 50–58.

[86] L. Prechelt, *An empirical comparison of seven programming languages*, Computer **33** (2000), no. 10, 23 –29.

[87] Ramesh Raskar, Amit Agrawal, Cyrus A. Wilson, and Ashok Veeraraghavan, *Glare aware photography: 4d ray sampling for reducing glare effects of camera lenses*, ACM Trans. Graph. **27** (2008), 56:1–56:10.

[88] L.R. Rabiner. R.E. Crochiere, *Multirate digital signal processing*, Prentice-Hall, 1983.

[89] T. Richardson, Q. Stafford-Fraser, K.R. Wood, and A. Hopper, *Virtual network computing*, Internet Computing, IEEE **2** (1998), no. 1, 33–38.

[90] Robert W. Scheifler and Jim Gettys, *The x window system*, ACM Trans. Graph. **5** (1986), 79–109.

[91] John Shalf and E. Wes Bethel, *The grid and future visualization system architectures*, IEEE Comput. Graph. Appl. **23** (2003), no. 2, 6–9.

[92] D. Shreiner, M. Woo, J. Neider, and T. Davis, *Opengl programming guide: The official guide to learning opengl*, Addison Wesley, 2007.

[93] H.-Y. Shum, S.-C. Chan, and S. B. Kang, *Image-based rendering*, First edition, Springer, September 2007.

[94] H.-Y. Shum and L.-W. He, *Rendering with concentric mosaics*, SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques (New York, NY, USA), ACM Press/Addison-Wesley Publishing Co., 1999, pp. 299–306.

[95] R. Sisneros, C. Jones, J. Huang, J Gao, B. Park, and N. Samatova, *A multi-level cache model for run-time optimization of remote visualization*, IEEE Transactions on Visualization and Computer Graphics **13** (2007), no. 5, 991–1003.

[96] Mel Slater, *The influence of rendering styles on participant responses in immersive virtual environments representing and validating digital business processes*, VISAPP (1), 2007, p. 2.

[97] William Stallings, *Snmp, snmpv2, snmpv3, and rmon 1 and 2*, 3 ed., Addison-Wesley Professional, Massachusetts, USA, January 1999.

[98] Yuichi Taguchi, Takafumi Koike, Keita Takahashi, and Takeshi Naemura, *Transcaip: A live 3d tv system using a camera array and an integral photography display with interactive control of viewing parameters*, IEEE Transactions on Visualization and Computer Graphics **15** (2009), 841–852.

[99] Eino-Ville Talvala, Andrew Adams, Mark Horowitz, and Marc Levoy, *Veiling glare in high dynamic range imaging*, ACM SIGGRAPH 2007 papers (New York, NY, USA), SIGGRAPH '07, ACM, 2007.

[100] Desney S. Tan, George G. Robertson, and Mary Czerwinski, *Exploring 3d navigation: combining speed-coupled flying with orbiting*, Proceedings of the SIGCHI conference on Human factors in computing systems (New York, NY, USA), CHI '01, ACM, 2001, pp. 418–425.

[101] Bernd F. Tomandl, Peter Hastreiter, Christof Rezk-Salama, Klaus Engel, Thomas Ertl, Walter J. Huk, Ramin Naraghi, Oliver Ganslandt, Christopher Nimsky, and Knut E. W. Eberhardt, *Local and Remote Visualization Techniques for Interactive Direct Volume Rendering in Neuroradiology1*, Radiographics **21** (2001), no. 6, 1561–1572.

[102] S. Christian Albright Wayne L. Winston, *Practical management science*, 2 edition ed., South-Western College Publishing, Massachusetts, USA, December 24 2002.

[103] B. Wilburn, *High performance imaging using arrays of inexpensive cameras*, Ph.D. thesis, 2004.

[104] B. Wilburn, M. Smulski, H.-H. K. Lee, and M. Horowitz, *The light field video camera*, Proceedings of Media Processors , SPIE Electronic Imaging 2002 (SPIE Electronic Imaging 2002, ed.).

[105] J. Wood, K. Brodlie, and J. Walton, *gviz - visualization middleware for e-science*, VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03) (Washington, DC, USA), IEEE Computer Society, 2003, p. 82.

[106] J. Wood, H. Wright, and K. Brodlie, *Collaborative visualization*, Proceedings of IEEE Visualization 1997 (Washington, DC, USA), VIS '97, IEEE Computer Society, 1997, pp. 253–259.

[107] Ilmi Yoon and Ulrich Neumann, *Ibrac: Image-based rendering acceleration and compression*, Eurographics 2000, Vol **19** (2000), 321–330.

[108] Shaohua Kevin Zhou and Rama Chellappa, *Illuminating light field: Image-based face recognition across illuminations and poses*, Automatic Face and

Gesture Recognition, Sixth IEEE International Conference on Automatic Face and Gesture Recognition (FG'04) (2004), 229.

Publications

Some of the work described in this thesis has been presented in the following papers:

- **Conferences**
  - Asma Al-Saidi, Nick J. Avis, Ian J. Grimstead, and Omer F. Rana, Distributed collaborative visualization using light Field rendering, CC-GRID, 2009, pp. 609–614.
  - Asma Al-Saidi, Nick J. Avis, Omer F. Rana, Abdelhamid Abdesselam, On-Demand Transmission Model Using Image-Based Rendering for Remote Visualization, UK e-Science All Hands Meeting, 7th-9th December 2009, Oxford ,UK.
- **Journals**
  - Asma Al-Saidi, David W. Walker and Omer F. Rana, "On-Demand Transmission Model for Remote Visualization Using Image-Based Rendering",Concurrency and Computation: Practice and Experience. (submitted)