EFFICIENT RENDERING OF COMPLEX SCENES ON HETEROGENEOUS PARALLEL ARCHITECTURES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

GÖKÇE YILDIRIM KALKAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

FEBRUARY 2014

Approval of the thesis:

EFFICIENT RENDERING OF COMPLEX SCENES ON HETEROGENEOUS PARALLEL ARCHITECTURES

submitted by **GÖKÇE YILDIRIM KALKAN** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen Dean, Graduate School of **Natural and Applied Sciences**	________________
Prof. Dr. Adnan Yazıcı Head of Department, **Computer Engineering**	________________
Prof. Dr. Veysi İşler Supervisor, **Computer Engineering Department, METU**	________________

Examining Committee Members:

Assoc. Prof. Dr. Halit Oğuztüzün Computer Engineering Department, METU	________________
Prof. Dr. Veysi İşler Computer Engineering Department, METU	________________
Assoc. Prof. Dr. Tolga Can Computer Engineering Department, METU	________________
Assist. Prof. Dr. Tolga Çapın Computer Engineering Department, Bilkent University	________________
Assist. Prof. Dr. Ahmet Oğuz Akyüz Computer Engineering Department, METU	________________

Date: ________________

ABSTRACT

EFFICIENT RENDERING OF COMPLEX SCENES ON HETEROGENEOUS PARALLEL ARCHITECTURES

Kalkan, Gökçe Yıldırım
Ph.D., Department of Computer Engineering
Supervisor : Prof. Dr. Veysi İşler

February 2014, 67 pages

In computer graphics, generating high-quality images at high frame rates for rendering complex scenes is a challenging task. A well-known approach to tackling this important task is to utilize parallel processing through distributing rendering and simulation tasks to different processing units.

In this thesis, several methods of distributed rendering architectures are investigated, and the bottlenecks in distributed rendering are analyzed. Based on this analysis, guidelines for distributed rendering in a network of computers are proposed.

Moreover, in the thesis, an efficient load balancing strategy is proposed for distributing the rendering of individual frames to different processing units in a network. In this distributed rendering heterogeneous system, there are computers equipped with multiple Graphical Processing Units (GPUs) with different rendering performances all in the same network with a server, which collects rendering performances of the GPUs in the different Image Generators (IGs) based on an effective load balancing. By means of the novel load balancing strategy, the thesis shows that such a system can increase the rendering performance of slow computers with the help of the fast ones.

Lastly, this model is extended to develop an adaptive hybrid model where (i) parts of a frame or a scene can be distributed and (ii) GPU-GPU and GPU-CPU distributions can be considered. This model can adjust itself to the changing loads of the GPUs and determine an efficient load balancing strategy for distributed rendering.

Keywords: Parallel Rendering, Scalable Visualization, Multiview Rendering, Heterogeneous Architectures

ÖZ

KARMAŞIK SAHNELERİN HETEROJEN PARALEL MİMARİLERDE ETKİN ÇİZİMİ

Kalkan, Gökçe Yıldırım
Doktora, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Veysi İşler

Şubat 2014, 67 sayfa

Bilgisayar grafiğinde karmaşık sahnelerin çizilmesi sırasında yüksek kaliteye sahip görüntülerin yüksek tazeleme hızında oluşturulması zor bir görevdir. Bu zor problem için mevcut en yaygın yaklaşımlardan bir tanesi, grafik çizim ve benzetim görevlerini farklı işlem ünitelerine dağıtarak paralel işlemektir.

Bu tez çalışmasında, öncelikli olarak, dağıtık çizim yöntemleri incelenmiştir, ve dağıtık çizimdeki darboğazlar incelenmiştir. Bu neticede, bir bilgisayar ağında dağıtık çizim yapmak için yönergeler önerilmiştir.

Ayrıca, bu tez çalışmasında, bir ağ içerisindeki farklı işlem ünitelerine birer birer çerçeveleri çizdirerek dağıtık çizim yapan, verimli bir yük dağılımı yöntemi önermiştir. Bu heterojen dağıtık çizim yönteminde, çizim yetenekleri ve performansları birbirinden farklı, birden çok Grafik İşlem Ünitesi (GİÜ) içeren bilgisayarlardan oluşan bir ağ bulunmaktadır ve bu ağda, GİÜ'lerin çizim performansını toplayan ve bunları hesaba katarak verimli bir yük dağıtımı yapan bir sunucu yer almaktadır. Özgün yük dağıtımı yöntemi sayesinde, tez çalışmasında, bu türden bir yöntemin, ağdaki hızlı bilgisayarların yardımı ile, çizimi yavaş bilgisayarların çizim performansının artırılabileceği gözterilmiştir.

Son olarak, bu yöntem genişletilmiş, melez ve adaptif bir model haline getirilmiştir, ki bu modelde, (i) bir sahnenin parçası veya tüm çerçeveler, ve (ii) GİÜ - işlemci dağıtımları yapılabilmektedir. Bu model, GİÜ'lerin değişen yüklerine kendisini uyarlayabilir ve dağıtık çizim için en uygun yük dağılımını belirleyebilir.

Anahtar Kelimeler: Paralel çizim, Ölçeklenebilir görselleştirme, Çok görüntülü çizim, Heterojen mimariler

To my family

ACKNOWLEDGMENTS

I wish to express my deepest gratitude to my supervisor Prof. Dr. Veysi İşler for his guidance, advice, criticism, encouragements and insight throughout the research.

I would like to thank management of Simsoft to provide the software and hardware environment for testing algorithms developed by this thesis.

This thesis is partially funded by the Ministry of Science, Industry and Technology under project number SANTEZ 00906.STZ.2011-1.

I would also like to thank my friends for their support during this study. Moreover, I would like to express my deepest appreciation to my husband and my family for their support and encouragement during the thesis.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vii

ACKNOWLEDGMENTS x

TABLE OF CONTENTS xi

LIST OF TABLES xv

LIST OF FIGURES xvi

LIST OF ABBREVIATIONS xviii

CHAPTERS

1 INTRODUCTION 1

1.1 Distributed Rendering 2

1.2 The Goal of the Thesis 3

1.3 Contributions of the Thesis 3

1.4 Outline of the Thesis 4

2 BACKGROUND AND LITERATURE SURVEY 5

2.1 Computer Graphics 5

2.2 Visualization of Big Data 7

2.3 Inter-frame Rendering 8

2.4 Intra-frame Rendering 8

2.4.1 Sort-first type rendering 9

2.4.2 Sort-last type rendering 9

2.4.3 Sort-middle type rendering 9

2.5 Parallel Rendering Software 10

2.5.1 WireGL 10

2.5.2 Chromium 10

2.5.3 OpenGL Multipipe SDK (MPK) 11

2.5.4 Equalizer 11

2.6 Bottlenecks of the Rendering Pipeline 13

2.7 Architecture of Graphical Processing Unit 13

2.8 Load Balancing Strategies 17

3 BOTTLENECKS OF DISTRIBUTED RENDERING 21

3.1 Rendering with a network of computers (Network Sharing Method - NSM) 21

3.2 Rendering locally among GPUs (GPU Sharing Method - GSM) 22

3.3 Results 22

3.3.1 Network Sharing Method 23

3.3.2 GPU Sharing Method 26

3.3.3 Guidelines 26

3.4 Conclusion 28

4 DISTRIBUTED RENDERING USING A NETWORK OF COMPUTERS 31

4.1 Formulation 31

4.2 A Distributed Solution 32

4.2.1 Determining k 33

4.2.1.1 Deriving number of frames an IG can help 33

4.2.1.2 Deriving number of frames an IG needs 33

4.2.2 Determining help frequencies 36

4.3 Consideration of Other Factors 36

4.3.1 Network Bandwidth 36

4.3.2 Rendered Triangles per Second 37

4.4 Results 37

4.5 Conclusion 40

5 A HYBRID SYSTEM FOR DISTRIBUTED RENDERING 43

5.1 Formulation 43

5.2 A Hybrid Solution 45

5.2.1 Determining k^T 46

5.2.1.1 Deriving number of frames an IG can help 46

5.2.1.2 Deriving number of frames an IG needs 46

5.2.2 Determining help frequencies 48

5.3 Results 50

5.3.1 Making the Solution Dynamic and Adaptive 52

5.4 Conclusion 53

6 CONCLUSION AND FUTURE WORK 55

6.1 Limitations and Future Work 56

REFERENCES 59

CURRICULUM VITAE 65

LIST OF TABLES

TABLES

Table 2.1 Comparison of the state of the art and this thesis. 19

Table 6.1 List of tasks that a full-fledged distributed rendering system should consider. 56

LIST OF FIGURES

FIGURES

Figure 1.1 Different distributed rendering types. 2

Figure 2.1 A snapshot from the first computer-animated movie on the altitude of a satellite rotating around the Earth [33]. 6

Figure 2.2 A snapshot from the first computer game - *Spacewar!* [38]. 6

Figure 2.3 A snapshot from image generators [15]. 7

Figure 2.4 Major components of WireGL (adapted from [20]). 10

Figure 2.5 A simple Chromium configuration (adapted from [21]). 11

Figure 2.6 OpenGL Multipipe SDK Application Structure (adapted from [34]). 11

Figure 2.7 Execution flow of an Equalizer application (adapted from [10]). . . 12

Figure 2.8 The bottlenecks in CPU, GPU and in between (adapted from [8]). . 14

Figure 2.9 The general rasterization pipeline (adapted from [36, 46]). 14

Figure 2.10 Old pipeline, before recent advanced in GPU architecture. 15

Figure 2.11 The general rasterization pipeline (adapted from [36, 46]). 16

Figure 2.12 Outline of a modern GPU composed of heterogeneous chips (adapted from [36]). 16

Figure 2.13 An illustration of static (a) and dynamic (b) load balancing (Source: [11]) 18

Figure 3.1 Flowchart of the scenario of Network Sharing Method. 22

Figure 3.2 Flowchart of the scenario of GPU Sharing Method in one IG. . . . 23

Figure 3.3 Experiment Environment. 24

Figure 3.4 A few snapshots from the rendering system. 25

Figure 3.5 The effect of Network Sharing Method on the IG receiving help. (a) With blocking the IG receiving help, (b) Non-blocking the IG receiving help. 27

Figure 3.6 The effect of Network Sharing Method on the IG giving help. . . . 28

Figure 3.7 The effect of GPU Sharing Method on the GPU receiving help. (a) With blocking the GPU receiving help, (b) Non-blocking the GPU receiving help. 29

Figure 3.8 The effect of GPU Sharing Method on the GPU giving help. . . . 30

Figure 4.1 The overview of the setup. 32

Figure 4.2 Effect of load balancing in the case of three fast IGs and one slow IG. 38

Figure 4.3 Effect of load balancing in the case of one fast IG and three slow IGs. 39

Figure 4.4 Effect of load balancing in the case of two fast IGs and two slow IGs. 40

Figure 5.1 The overview of the hybrid setup. 44

Figure 5.2 The effect of load balancing analyzed in different combinations of slow and fast IGs. The case with one slow IG and three fast IGs, where the second GPU of IG_2 helps IG_6, i.e., $H(IG_6) = \{IG_{22}\}$. 49

Figure 5.3 The effect of load balancing analyzed in different combinations of slow and fast IGs. The case with two fast IGs and two slow IGs, where $H(IG_8) = \{IG_{11}\}$ and $H(IG_5) = \{IG_{11}, IG_{22}\}$. 49

Figure 5.4 The effect of load balancing analyzed in different combinations of slow and fast IGs. The case with one fast IG and three slow IGs, where $H(IG_7) = \{IG_{22}\}$, $H(IG_6) = \{IG_{22}, IG_{52}\}$ and $H(IG_5) = \{IG_{52}\}$. . . 50

Figure 5.5 The effect of load balancing analyzed in different combinations of slow and fast IGs in the case of a smoky scene. The case with three fast IGs and one slow IG, where $H(IG_8) = \{IG_{51}, IG_{82}\}$ in a smoky scene. . 52

Figure 5.6 The effect of load changes on the execution period of the algorithm. 53

LIST OF ABBREVIATIONS

API	Application Programming Interface
CPU	Central Processing Unit
CRT	Cathode Ray Tube
FOV	Field-Of-View
GPU	Graphical Processing Unit
IG	Image Generator
LOD	Level-Of-Detail

CHAPTER 1

INTRODUCTION

With the desire to visualize huge data and to simulate complex events, or systems that are composed of either natural or man-made entities, rendering and simulation techniques and technologies have advanced rapidly in the last decade. Due to such advances, nowadays, computer games or simulations can be run on mediocre end-user personal computers with fidelity as high as expensive computer generated movies.

One important challenge is rendering high-resolution large-data composed of complex objects at high frame rates and in real-time. Despite the advances in graphics hardware, since the memory capacity on graphics hardware is limited so is the graphics computational power, visualization of huge data is very difficult or impossible on a single graphics system. For this purpose, many graphics systems that may or may not have the same processing power should be used in fulfilling the simulation of huge data. At this point, scalability of graphics systems and as such, parallel rendering plays an important role in order to be able to improve the performance of computer graphics software. By parallel rendering, what is meant is simply a concept to exploit multiple processing units in order to obtain sufficiently high performance [7]. In this thesis, heterogeneous systems [47] which include different types of processing units, CPUs and GPUs are used for parallel rendering. Heterogeneity is also obtained by means of load imbalances on computers during visualization of the scene which has different loads at different timestamps and different Field-Of-Views (FOVs).

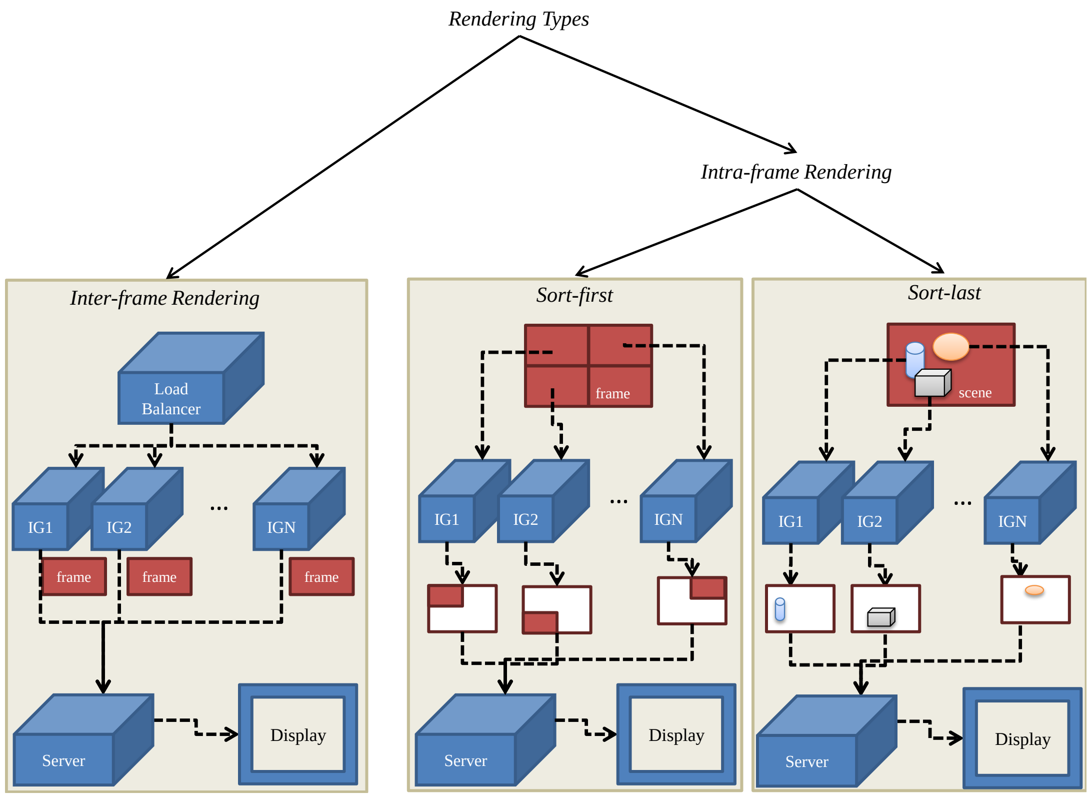


Figure 1.1: Different distributed rendering types.

1.1 Distributed Rendering

When the power of a processing unit is not sufficient, distribution becomes a necessity. As outlined in Figure 1.1, for rendering, there are two main approaches: inter-frame and intra-frame based distribution. In inter-frame rendering, individual frames are generated by the processing units and the generated frames are combined to a single display unit. On the other hand, in intra-frame rendering, parts of a frame or a scene are distributed.

According to what is distributed, intra-frame rendering can be of three types: sort-first, sort-middle and sort-last classes [49]. There are various APIs which utilize one of these approaches. Some of these are WireGL [20], Chromium [21], Multi OpenGL Multipipe SDK (MPK) [34] and Equalizer [10]. Among these APIs, Equalizer provides a much more scalable, flexible and compatible interface with less implementation overhead [17].

1.2 The Goal of the Thesis

The goal of this thesis is to optimize the resource management capabilities of a distributed framework. This includes the research issues of finding advanced load balancing strategies for different task sharing strategies for distributed rendering of both individual frames as well as parts of a single frame. For this purposed, the thesis investigates the possibilities of distributed rendering in a network of computers with single GPUs (computer-computer sharing) and in a network of computers with multiple GPUs (computer-computer sharing and GPU-GPU sharing).

1.3 Contributions of the Thesis

The following are the contributions of the thesis:

- **Analysis of the bottlenecks in distributed rendering:** The thesis investigates the several options for distributed rendering of huge data. Rendering load can be shared either between GPUs in a computer or among a set of computers connected by a network. The thesis, with proof of concept implementations, analyzes the bottlenecks in each case and proposes guidelines for developing distributed rendering. This part of the thesis is submitted as an article to a journal [24].
- **Distributed rendering in a network of computers:** The thesis proposes a novel load balancing strategy for distributing the workload among a network of computers. The load balancing is formalized as a matching problem and solved efficiently using a simple mechanism. In this part, individual frames are shared among the computers. The thesis shows that *fast* computers can help *slow* computers increase their rendering performances.
- **A hybrid framework for distributed rendering:** The thesis extends the previous contribution by also taking into account the sharing between GPUs. The sharing between GPUs is for the parts of frames whereas individual frames are helped by other computers. This contribution is in preparation for a journal submission [23], and submitted to an international conference [25].

1.4 Outline of the Thesis

In Chapter 2, the literature about parallel architectures and APIs is surveyed. The general approaches to parallel rendering are described in detailed and their advantages and disadvantages are discussed.

In Chapter 3, the bottlenecks in parallel rendering architectures are described and analyzed with a few implementations. Moreover, based on these analyses, guidelines for implementing such systems are provided.

Then, in Chapter 4, a model for distributed rendering of a scene is proposed. In the system, a network of computers shares the rendering by a novel load balancing scheme performed by a server. The load balancing method is a novelty of this thesis.

In Chapter 5, a hybrid architecture for distributed rendering is proposed. In this architecture, there exist a network of computers with multiple GPUs, and load balancing makes use of GPU-sharing as well as sharing between computers.

Finally, in Chapter 6, the thesis is concluded with a summary of the contributions, a discussion and a limitation of the proposed architectures as well as an outline of future research directions.

CHAPTER 2

BACKGROUND AND LITERATURE SURVEY

This chapter first gives a background on Computer Graphics and then reviews the existing approaches on distributed rendering architectures. As outlined in Figure 1.1, for rendering, there are two main approaches: inter-frame and intra-frame based distribution. In inter-frame rendering, individual frames are generated by the processing units and the generated frames are combined to a single display unit. On the other hand, in intra-frame rendering, parts of a frame or a scene are distributed.

2.1 Computer Graphics

Led by the computer gaming and movie industries, Computer Graphics is one of the biggest fields in Computer Science. It dates back to 1950s when Douglas T. Ross displayed a set of black-white pixels or lines, and later a cartoon character on CRT displays in the Whirlwind computer [6]. In this sense, Douglas T. Ross can be considered as the *father* of Computer Graphics, though the name is coined only later by William Fetter in Boeing [12].

One of the earliest landmarks in Computer Graphics was the first video game, *Spacewar!*, developed by Steve Russell [38] in 1961 - see Figure 2.2 for a snapshot. The other one was the first computer animated movie by Bell Telephone Laboratory for visualizing the altitude of a satellite rotating around the Earth [33] in 1963 - see Figure 2.1. Not later than these achievements, major companies became interested in Computer Graphics, and games were used for testing and benchmarking hardware.

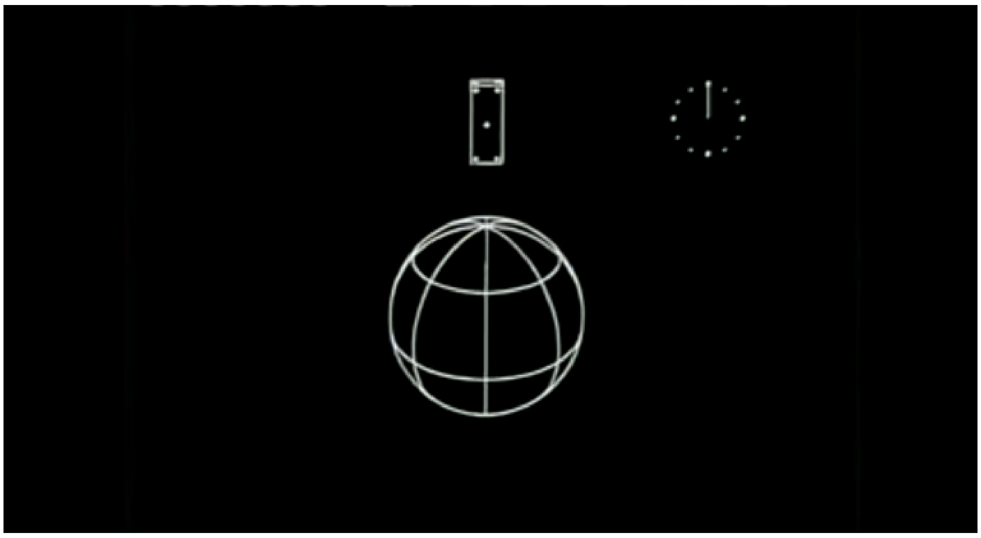


Figure 2.1: A snapshot from the first computer-animated movie on the altitude of a satellite rotating around the Earth [33].

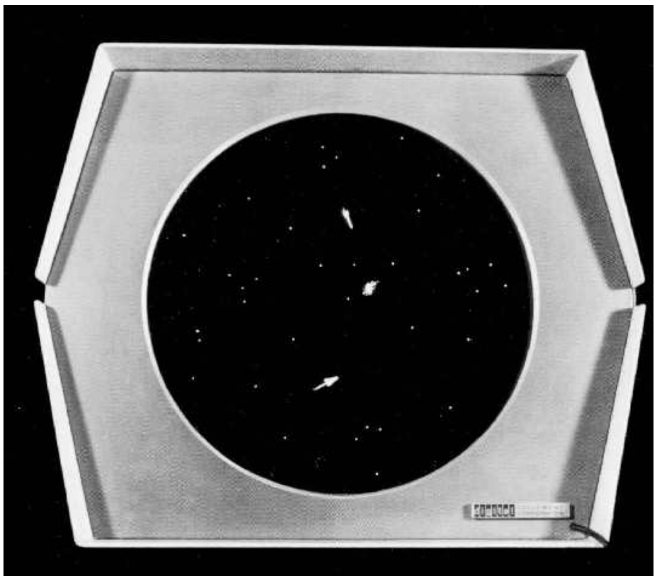


Figure 2.2: A snapshot from the first computer game - *Spacewar!* [38].

Figure 2.3: A snapshot from image generators [15].

With the advances in computer hardware especially in the 90s, Computer Graphics advanced rapidly, leading to the first full-length 3D computer-animated movie, Toy Story, and the first full-3D computer game, Quake. Since then, high fidelity, highly realistic games and computer-animated movies have been developed, and they have been leading the Computer Graphics techniques and hardware every since.

As well as animations and computer games, the need of training of some difficult and expensive systems such as aircrafts resulted in the generation of flight simulators including visual systems. One of the first visual systems as image generators to be used in flight simulators were developed by Evans & Sutherland company which is founded by Dr. David Evans and Dr. Ivan Sutherland in the campus of Utah University [44]. Currently, CAE [15] and Rockwell Collins [16] image generators are well known image generators which are used in flight simulators.

2.2 Visualization of Big Data

One of the most important applications of Computer Graphics is visualization. Even the first Computer Graphics application was the visualization of positions of a pen on a display.

With the ability to generate and store more data, visualization of huge data became an important need. The requirements for the visualization of large area database of the whole earth in flight simulators also supported the need for visualization of huge data. The visual systems in such simulators also consist of multi channels thus requiring multiview rendering of huge data. It is claimed that humans have produced five exabytes, i.e., five billion gigabytes, of information until 2003 since the earlier records of any available information [45]. Strikingly, the same amount is nowadays created every a few days [45]. To make sense of even subsets of such data, visualization became a crucial tool for humanity [19, 31]. However, since processing power on a single processing unit (whether CPU or GPU) is not sufficient for huge data, possibilities of distributing the graphic processing load among several processing units.

In simulations and games, loops mainly include input, processing and rendering parts. In input part, input from the user (trainee in training simulators or player in games) is gathered. Processing part includes sub-parts such as Artifial Intelligence, Physics Simulation and Networking. In modern systems, processing part can also be done on GPUs as well as CPUs. GPUs can be seen as multicore multiprocessors on which artificial intelligence or physics computations can be processed as General-Purpose Computation on GPUs (GPGPU) [22, 50]. Rendering part is the visualization stage of the scene to the user. This part is mainly done on GPUs. In this thesis, the rendering part is done on GPUs and other processing part such as physics computations are done on CPUs. GPUs help other GPUs in rendering the whole scene or part of the scene such as particles.

2.3 Inter-frame Rendering

In inter-frame rendering, distribution is done for complete frames of the scene. Different from inter-frame techniques, the overheads in composition of different parts of the frame or scene are prevented by means of inter-frame rendering [18].

2.4 Intra-frame Rendering

Intra-frame rendering is good for load balancing in the rendering of a scene on multiple GPUs. However, it requires a composition overhead varying according to sorting type. According to what is distributed, intra-frame rendering can be of three types: sort-first, sort-last and sort-middle [37].

2.4.1 Sort-first type rendering

In the sort-first approach, a camera view is divided into several subviews, or partitions, and each subview is assign to a processing unit [32] - see also Figure 1.1. The outputs of the processing units are easily tiled together by a server for a final display device [2, 49]. Because of its geometry transformation phase, it is expensive for distributed rendering. However, its network bandwidth usage is less than the others.

This approach is dependent on the size of the input dataset while it is irrespective of image resolution. Therefore, it is best to use this approach for applications whose final frame rate decreases with the increasing image resolution.

2.4.2 Sort-last type rendering

In the sort-last approach, the scene is partitioned into sets of objects or entities and these sets are shared among the processing units [29, 32] - see also Figure 1.1. The outputs of the processing units are only snapshots of only a part of the scene, and they are combined by a server possibly with a post-processing stage, where the depths of the individual objects are taken into account for a coherent snapshot of the scene [49]. The approach needs more bandwidth than the sort-first rendering approach, however, it has a better control of load balance in terms of object-space primitives.

This approach is dependent on image resolution while it while it is irrespective of input dataset. Therefore, this approach is well suited for applications whose final frame rate decreases with the increasing 3D input database.

2.4.3 Sort-middle type rendering

In the sort-middle approach, a hybrid of sort-last and sort-first is performed, where parts of the view as well as the objects in sub-views are distributed [32, 39].

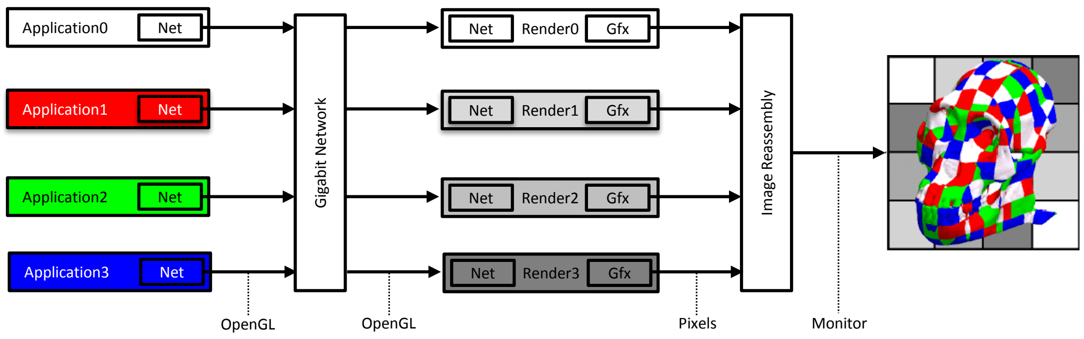


Figure 2.4: Major components of WireGL (adapted from [20]).

2.5 Parallel Rendering Software

In the literature, there are some parallel rendering systems designed for parallel architectures as summarized in the following subsections. However, most of these architectures, when used in distributed systems, are usually designed to work at high performance on very fast network infrastructures such as Myrinet and Infiniband.

2.5.1 WireGL

WireGL is a software compatible with OpenGL for distributed rendering in a network of workstations [20]. For distributed rendering, it uses the sort-first approach. Major components of WireGL are shown in Figure 2.4.

2.5.2 Chromium

Chromium is another software for distributed rendering in a network of workstations [21]. Unlike WireGL, Chromium allows both sort-first and sort-last type rendering architectures to be developed. Chromium also supports the use of stream processing units. However, Chromium's performance can be effected negatively because of the orientation of its stream [37]. A simple Chromium configuration can be seen in Figure 2.5.

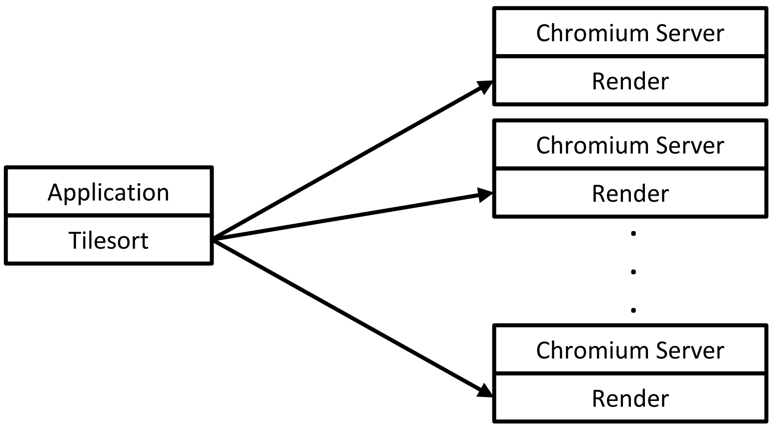


Figure 2.5: A simple Chromium configuration (adapted from [21]).

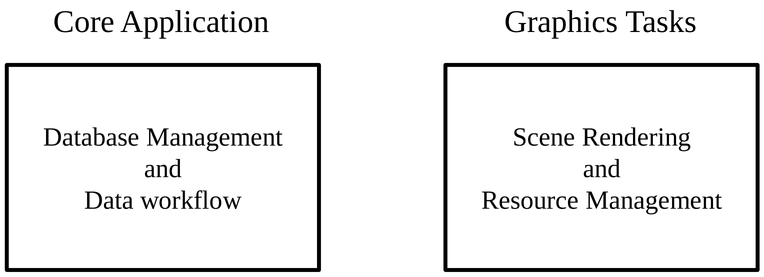


Figure 2.6: OpenGL Multipipe SDK Application Structure (adapted from [34]).

2.5.3 OpenGL Multipipe SDK (MPK)

OpenGL Multipipe SDK is an API designed to create complex immersive environments. Run-time configurability and run-time scalability are the main features of MPK [34]. Furthermore, it has integrated support for scalability graphics hardware, stereo and immersive environments. Figure 2.6 shows MPK's application structure.

2.5.4 Equalizer

Equalizer [9] is an OpenGL-based software which is flexible and capable. It can be used for large-scale distributed rendering in a network of computers with multiple CPUs and multiple GPUs as well as single CPU and single GPU desktop computers [10, 17]. Figure 2.7 shows the execution flow of an Equalizer application.

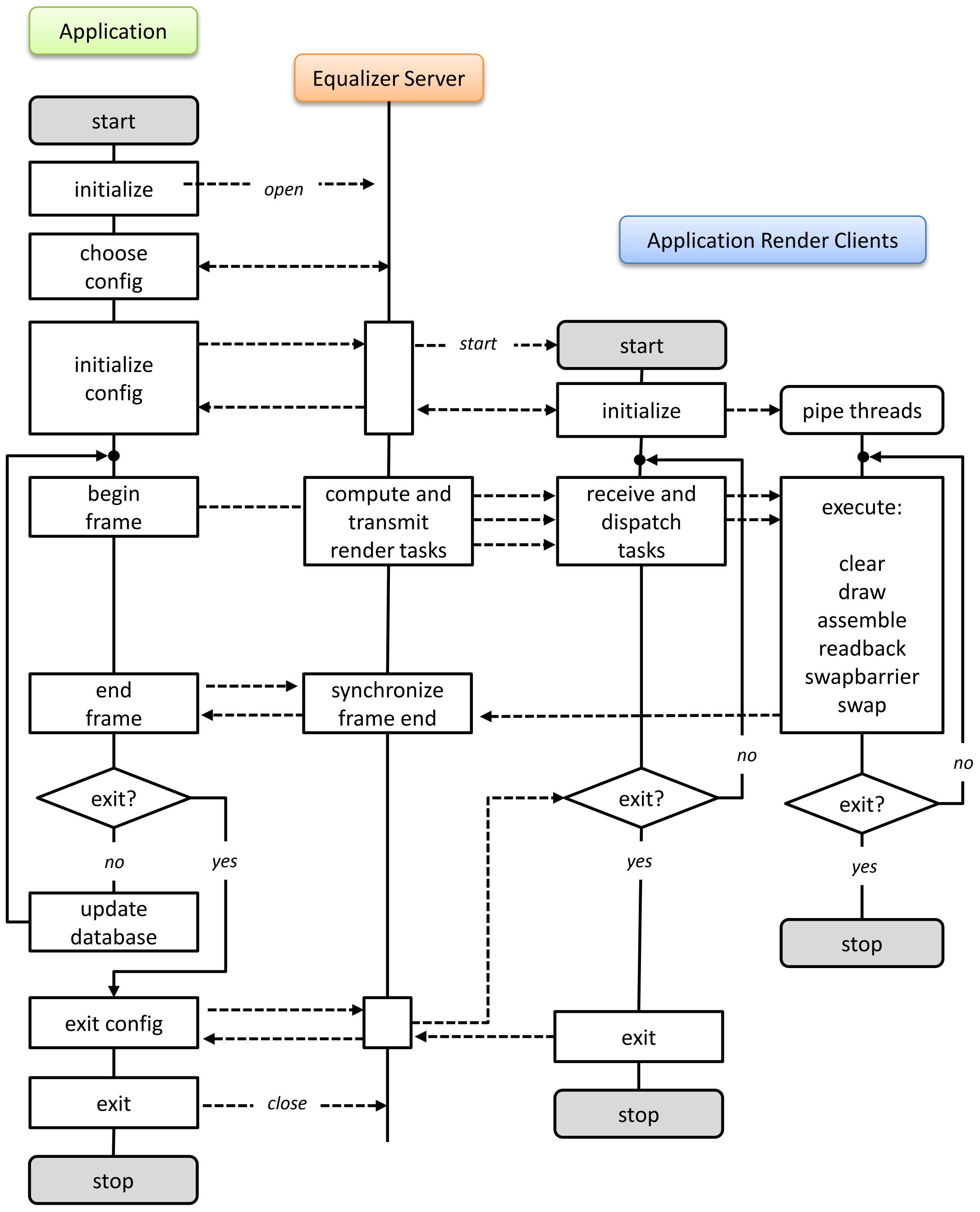


Figure 2.7: Execution flow of an Equalizer application (adapted from [10]).

2.6 Bottlenecks of the Rendering Pipeline

In a distributed rendering system, there are many workstations with different CPUs and GPUs connected together in a network. The overall performance of such a distributed system is severely affected by many factors, as shown in Figure 2.8. Below are the most important factors that a distributed rendering system will face and that will be investigated in Chapter 3:

- Cost of data transfer in a computer or between computers
- Discrepancies due to heterogeneous processing units, some of which are slower than others
- Redundant computations in the different rendering units
- Redundant storage since same scene data is replicated on each processing unit for faster access

2.7 Architecture of Graphical Processing Unit

Our aim in this thesis is to fully exploit the computing power available in current heterogeneous architectures and thus increase the rendering capacity and efficiency of the overall system performance.

To fully exploit the computing power available in especially modern GPUs, we have analyzed the architecture of the GPU in detail. The general rasterization pipeline can be seen in Figure 2.9 [36, 46]. In Geometry Processing Stage, geometry is transformed, more geometry is generated and per-vertex attributes are computed. In the rasterization stage, a primitive (e.g., a triangle) is set up and all samples inside the primitive are found. In Pixel Processing Stage, vertex attributes are interpolated and pixel color is computed.

Before the recent developments in GPU architecture, the pipeline had the following stages (Figure 2.10): Input Assembler, Vertex Shader, Rasterization, Pixel Shader and Output Merger.

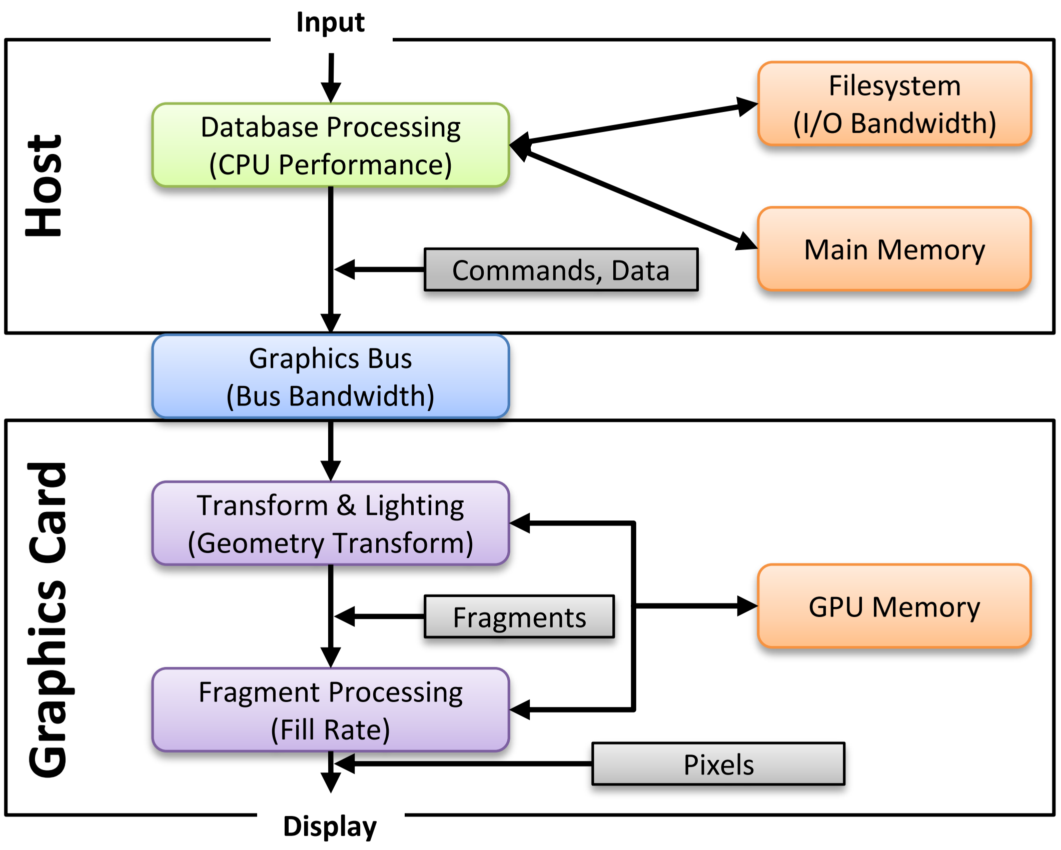

Input
Host
Database Processing (CPU Performance)
Filesystem (I/O Bandwidth)
Main Memory
Commands, Data
Graphics Bus (Bus Bandwidth)
Graphics Card
Transform & Lighting (Geometry Transform)
Fragments
GPU Memory
Fragment Processing (Fill Rate)
Pixels
Display

Figure 2.8: The bottlenecks in CPU, GPU and in between (adapted from [8]).

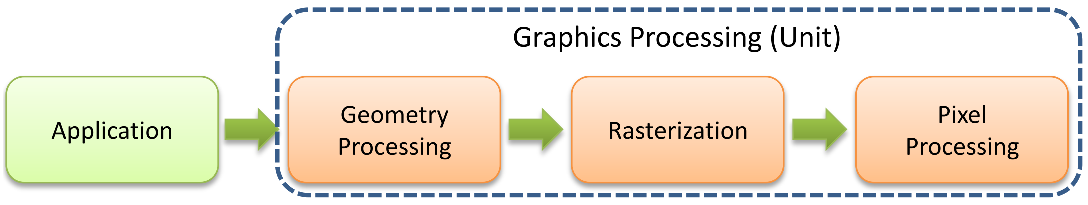


Figure 2.9: The general rasterization pipeline (adapted from [36, 46]).

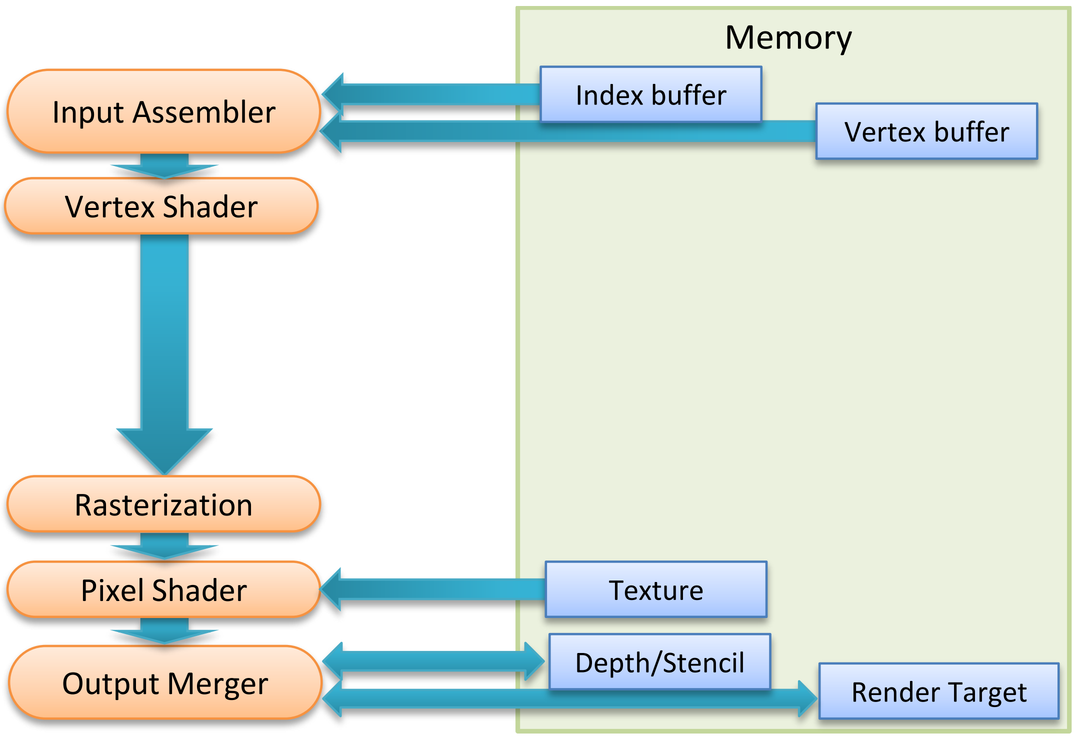


Figure 2.10: Old pipeline, before recent advanced in GPU architecture.

With the very new developments, the capabilities of shaders have increased and new stages and capabilities have been introduced (Figure 2.11). One of the new stages is the Geometry Shader which is especially used in generation of shadow volumes and cubemaps. Hull Shader, Tesselator and Domain Shader are newer stages which go between Vertex and Geometry Shaders.

A modern GPU itself can be seen as heterogeneous chip multi-processor especially highly tuned for graphics [36] as exemplified in Figure 2.12.

The following are the key ideas that should be considered in such a model [36]:

- Think of scheduling the pipeline as mapping tasks onto cores
- Preallocate resources before launching a task: Preallocation helps ensure forward progress and prevent deadlock
- Graphics is irregular: Dynamically generating, aggregating and redistributing tasks at irregular amplification points regains coherence and load balance
- Order matters: Carefully structure task redistribution to maintain ordering

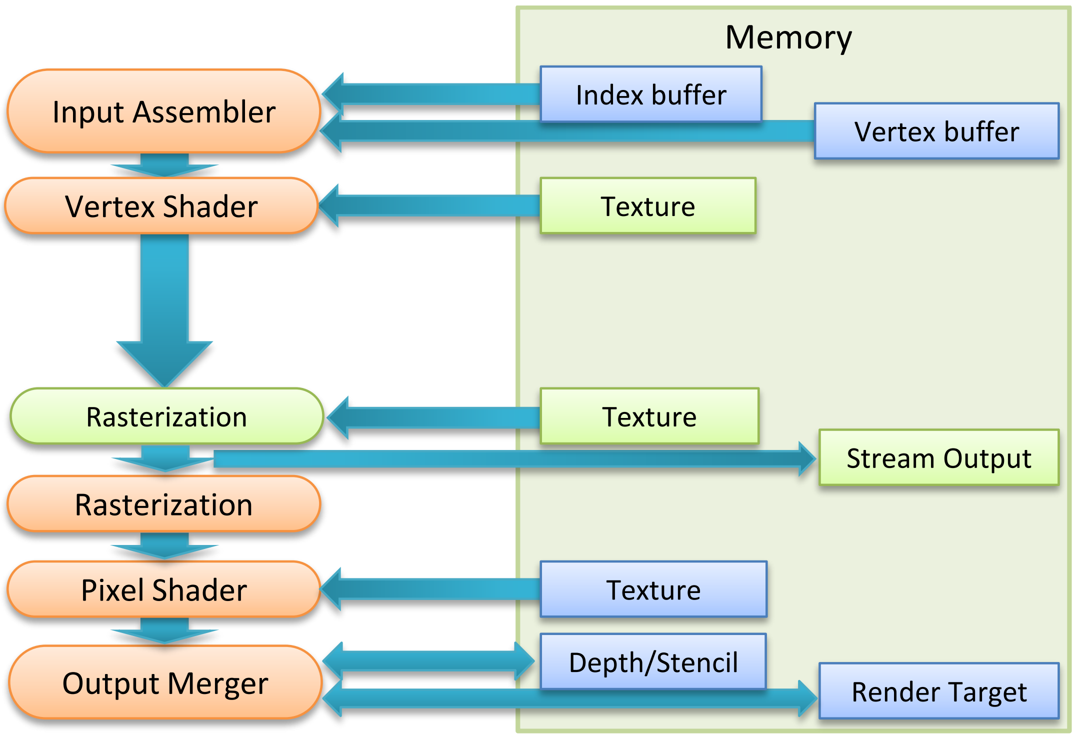


Figure 2.11: The general rasterization pipeline (adapted from [36, 46]).

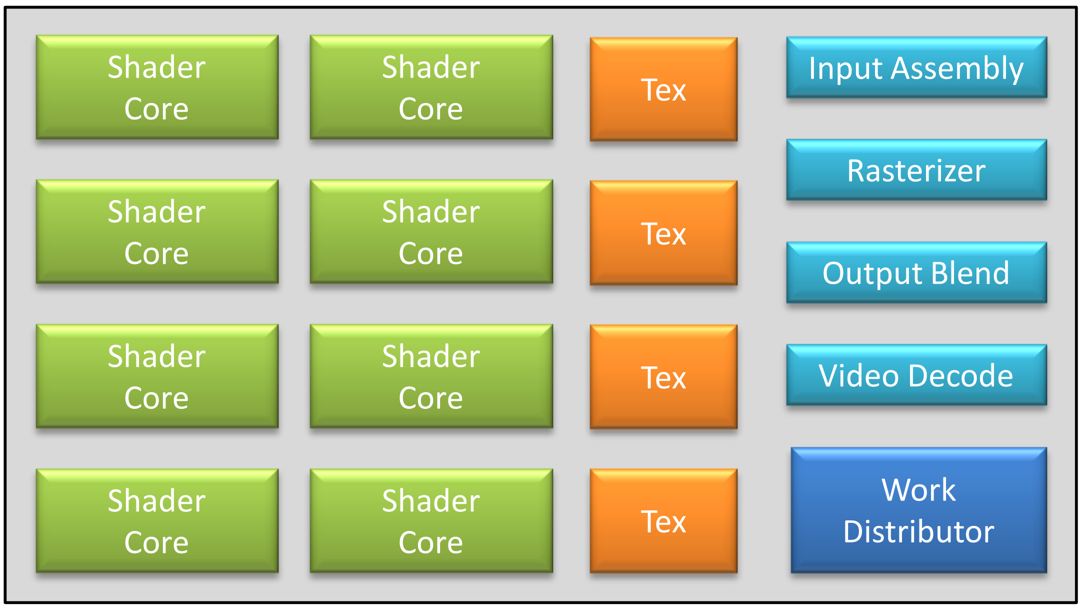


Figure 2.12: Outline of a modern GPU composed of heterogeneous chips (adapted from [36]).

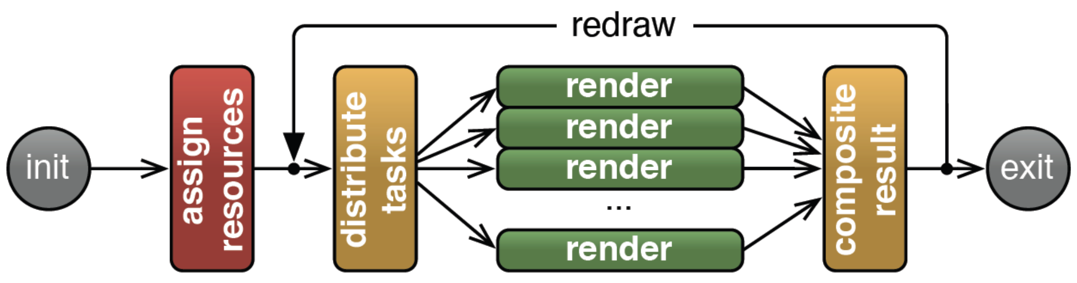
2.8 Load Balancing Strategies

As examined in the previous sections, different methods exist for distributed parallel rendering for both network of computers and multi processors in a local computer. The heterogeneous systems which render from different views of the scene on distributed computers including different number and types of processors end up with load imbalances. These load imbalances may occur because of the heterogeneous state of the rendered scene with multi-view aspects and/or physical differences on computer (and its processors') powers and/or network communication loads.

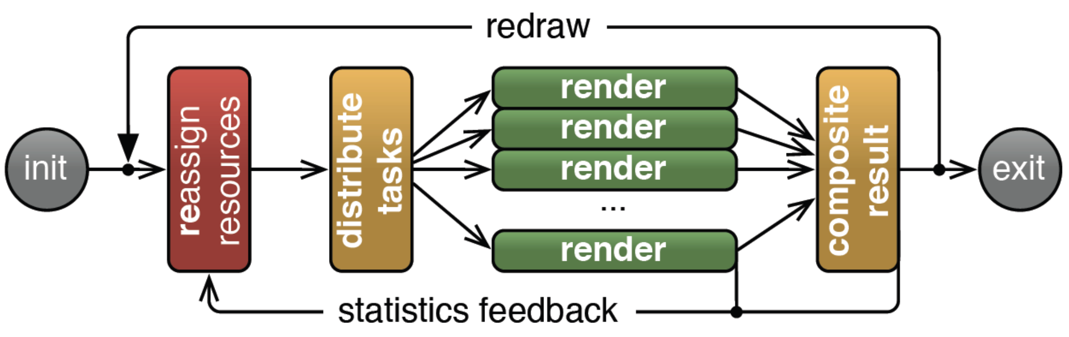
Load balancing strategies make distributed parallel systems differ from each other in terms of performance of interactive real-time applications. Load balancing algorithms use different algorithms for partitioning of load on the processors. Some are static, some are dynamic. Static load balancing algorithms usually partition processors or load at the beginning of application, and are not affected by the load imbalances while the application is running - see Figure 2.13(a) for an illustration. For this reason static load balancing algorithms cannot respond to load imbalances at runtime.

On the other hand, dynamic load balancing algorithms make partition load on processors at runtime, for this reason they can easily respond to load imbalances at runtime [1, 3, 5, 11, 13, 14, 26, 27, 28, 35, 40, 42, 48] - see Figure 2.13(b) for an illustration. In load balancing algorithms, the design issues should consider factors such as network bandwidth, computation loads and communication loads. Computation loads vary based on the computations for pre-processing and post-processing in sorting mechanisms to processing of network packets.

If we look at some of the methods in more detail; in Frederico et al., [13], a load balancing algorithm based on rendering times of previous frames is proposed for sort-first rendering systems including CPU-GPU-network components. In another study, Cederman and Tsigas [5] compare four dynamic load balancing methods which depend on the differences in processing strategy of tasks on multi-core GPUs. It is shown that lock-free methods have higher performance than blocking methods. In the work of Marchesin et al. [28], Level-Of-Detail (LOD) volume rendering is dynamically distributed on sort-last systems. Erol et al. [11] use a cross-segment load



(a)



(b)

Figure 2.13: An illustration of static (a) and dynamic (b) load balancing (Source: [11])

balancing strategy which is built on Equalizer. The method assigns available graphic resources which are shared to output displays. In the work by Ahrens and Painter [1], decreases the time of the compositing step in sort-last methods by using a compression algorithm based on a run-length encoding. Binotto et al. [3] propose a dynamic load-balancing method working on CPU and GPU and applies their method on solvers for Systems of Linear Equations in a Computational Fluid Dynamics application. A load balancing approach for in-situ visualization is proposed by Binotto et al. [3]. The work applies the approach for multiple GPUs in one computer.

Comparisons of some of these methods with the contributions of the thesis are shown in Table 2.1. As seen in the table, the hybrid method proposed in this thesis differs from the others in the way it uses a dynamic load balancing strategy using a hybrid approach of inter-frame and intra-frame rendering. This method also works both on a network of computers and multi-GPU systems.

Table 2.1: Comparison of the state of the art and this thesis.

Study	Distributed Rendering Type	Networked?	Multi-GPU in One Node?
[13]	intra-frame (sort-first)	yes	no
[5]	unknown	no	yes
[28]	intra-frame (sort-last)	yes	no
[11]	intra-frame (sort-first)	yes	yes
[3]	unknown	no	yes
[18]	inter-frame	no	yes
This thesis	inter-frame and intra-frame (sort-last)	yes	yes

CHAPTER 3

BOTTLENECKS OF DISTRIBUTED RENDERING

In distributed simulation systems, hardware speed and latency in communication are very crucial for the overall performance and design. In this chapter, these bottlenecks are described and analyzed in detail. Moreover, based on these analyses, guidelines for implementing such systems are provided. For this purpose, the chapter investigates two methods: The first method distributes load among a network of computers while the second method distributes load on the same computer among different GPUs.

3.1 Rendering with a network of computers (Network Sharing Method - NSM)

In this method, load is distributed among distributed computers. The distributed environment consists of the Central Control Computer and a number of Image Generators (IGs). An IG is an image generator computer which renders the scene. The Central Control Computer is the master computer which listens to the states of IG's in terms of refresh rate and decides to give commands IGs to help other IGs rendering their frames if their refresh rate is under a threshold. The flow is processed as shown in Figure 3.1. The Central Control Computer commands IG_1 to render the frame for IG_2. When IG_1 completes IG_2's frame, it sends the frame over fast network to IG_2. IG_2 receives the frame and renders. The decision for distributing load among IG's is given by central computing computer according to the current refresh rates of IGs.

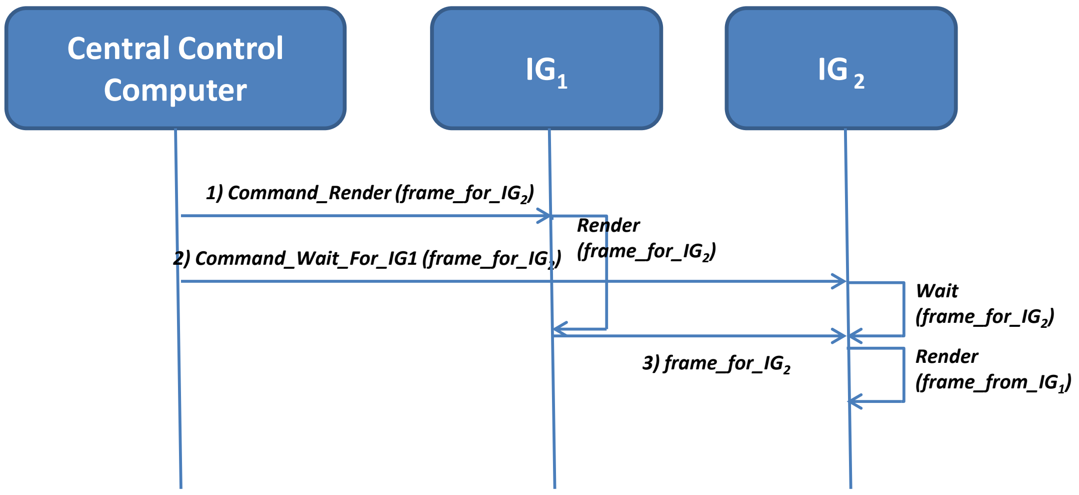


Figure 3.1: Flowchart of the scenario of Network Sharing Method.

3.2 Rendering locally among GPUs (GPU Sharing Method - GSM)

In this method, the load is distributed on the same computer among different Graphical Processing Units (GPUs). The IG application is responsible for rendering the scene. In this method, the flow is processed as shown in Figure 3.2. The IG application commands helper GPU (GPU$_2$) to render the frame which includes post-processing effects such as particles for the main GPU (GPU$_1$). GPU$_1$ is the GPU which is responsible for rendering the all scene. When GPU$_2$ completes the frame, it shares the frame with GPU$_1$ via memory of the IG. In the meantime, GPU$_1$ renders the main scene and blends the frame which includes post-processing effects such as particles rendered by GPU$_2$. In this method, the decision for load balancing is made locally in the IG by the IG application according to the state of GPU_1 in terms of refresh rate. When the refresh rate is under a threshold, IG application decides for main GPU to distribute its load to the helper GPU. The output of the main GPU is used as the IG output. The output of the helper GPU is only used as in input for the main GPU.

3.3 Results

The experiments were performed on a network of computers each of which contains two GTX 680 graphic cards (2GB RAM, 256 Bit), Intel i7 2700K processors and

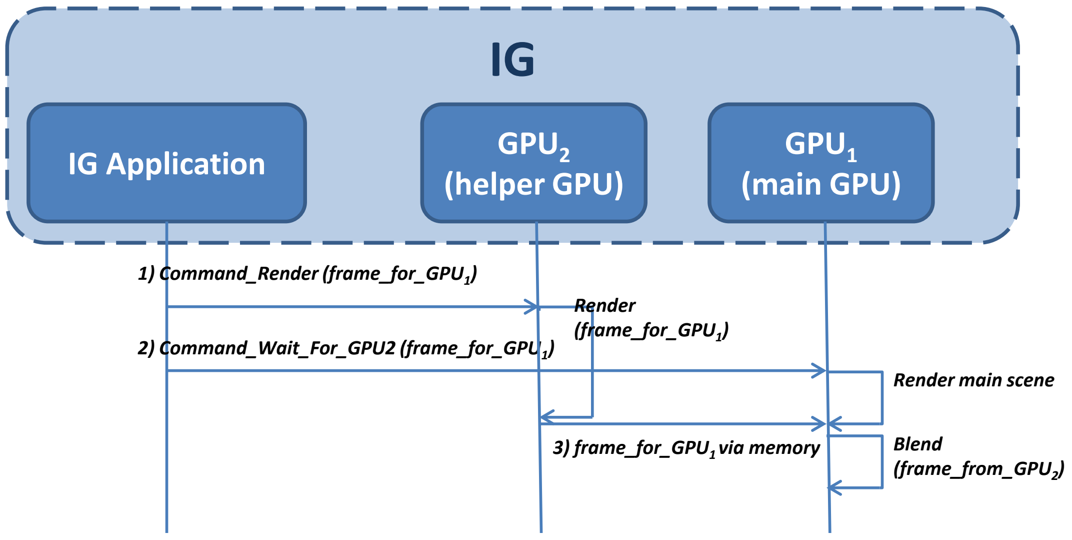


Figure 3.2: Flowchart of the scenario of GPU Sharing Method in one IG.

16GB main memory - see Figure 3.3 for an outline of the experimental environment. The scene was selected as a large terrain in which some dust is scattered as particles in some regions of the terrain (see Figure 3.4 for a detailed snapshot). Each IG controls a window which has a camera view from the same viewpoint with contiguous field of views so that we can have a large field of view for the scene.

3.3.1 Network Sharing Method

In the network, there is significant latency due to transmission of a packet, which can lead to incoherence in displayed frames. Two solutions to this problem are: (i) Block the IG receiving help until the next frame in sequence arrives. (ii) Send the frames to the slow IG latest at $t - \Delta t_N$, where Δt_N is the delay due to network transmission and the related processing. In this section, we will have a look at both. In any case, for load distribution to be worth the effort for an IG (IG_1) that needs help, the time for rendering one frame should be costing at least the time for a packet to travel on network and the rendering time for the helper IG (IG_2):

$$\Delta t_{IG_1} > \Delta t_{IG_2} + \Delta t_N, \tag{3.1}$$

where Δt_{IG_1} and Δt_{IG_2} are frame rendering times for IG_1 and IG_2, respectively. The decision for the Central Control Computer should be based on the frame rendering times of IG_1 and IG_2 according to the criteria in Equation 3.1.

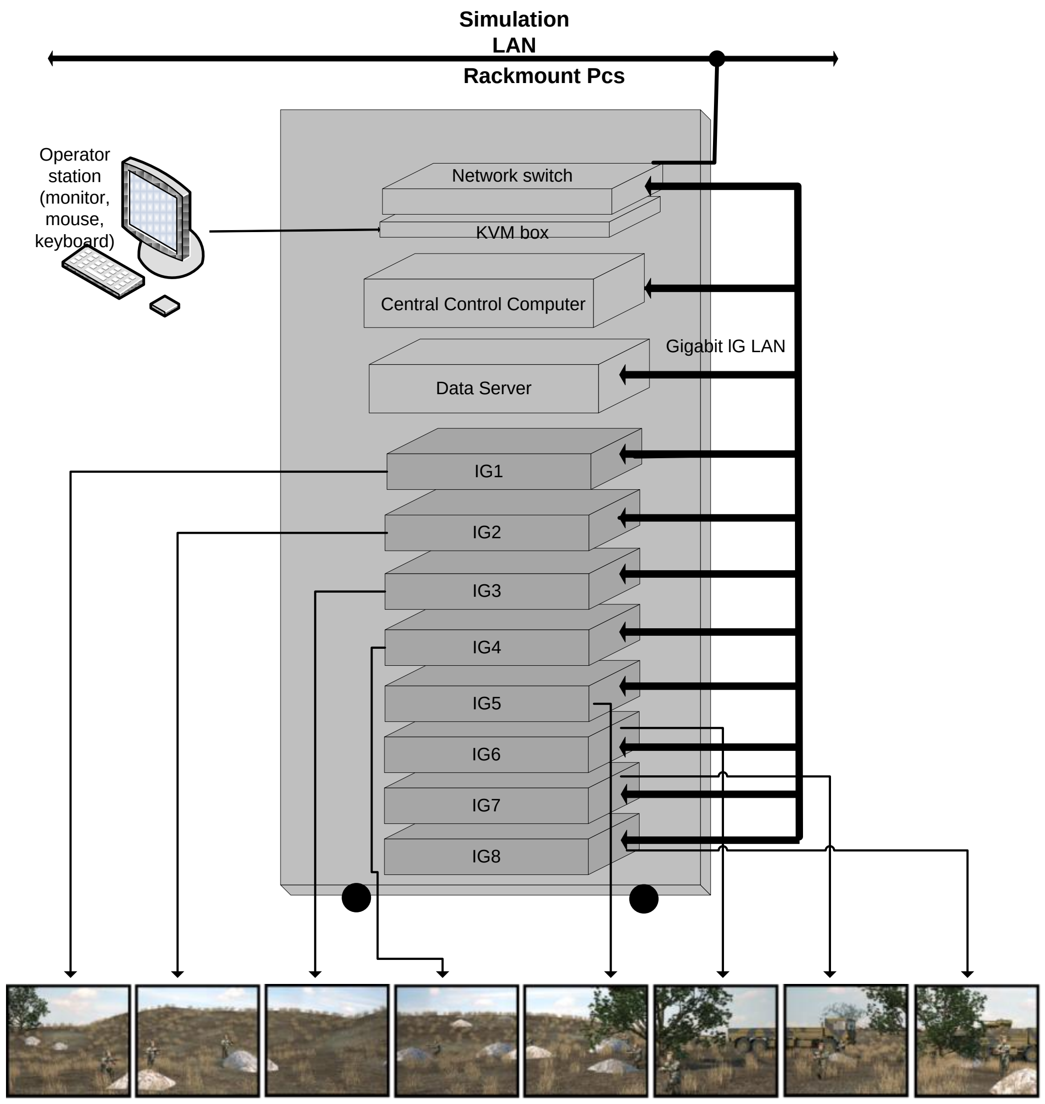


Figure 3.3: Experiment Environment.

In addition, for an application that needs to achieve 25 fps, the time for the helper IG (IG_2) to render one frame for the other IG should be smaller than the time for rendering its own frame (40ms in the case of 25 fps) minus the time it takes helper IG to render the other IG's frame:

$$\Delta t_{IG} + \Delta t_{OF} < 40ms, \tag{3.2}$$

where Δt_{IG} shows the time of rendering its own frame for a helper IG, and Δt_{OF} is the time it takes a helper IG to render the other IG's frame. Based on this result, we can conclude that the decision for the Central Control Computer should also be based on the frame render times of helper IGs according to the criteria in Equation 3.2.

In Figure 3.5, we analyze the effect of the received help on the refresh rate of the IG

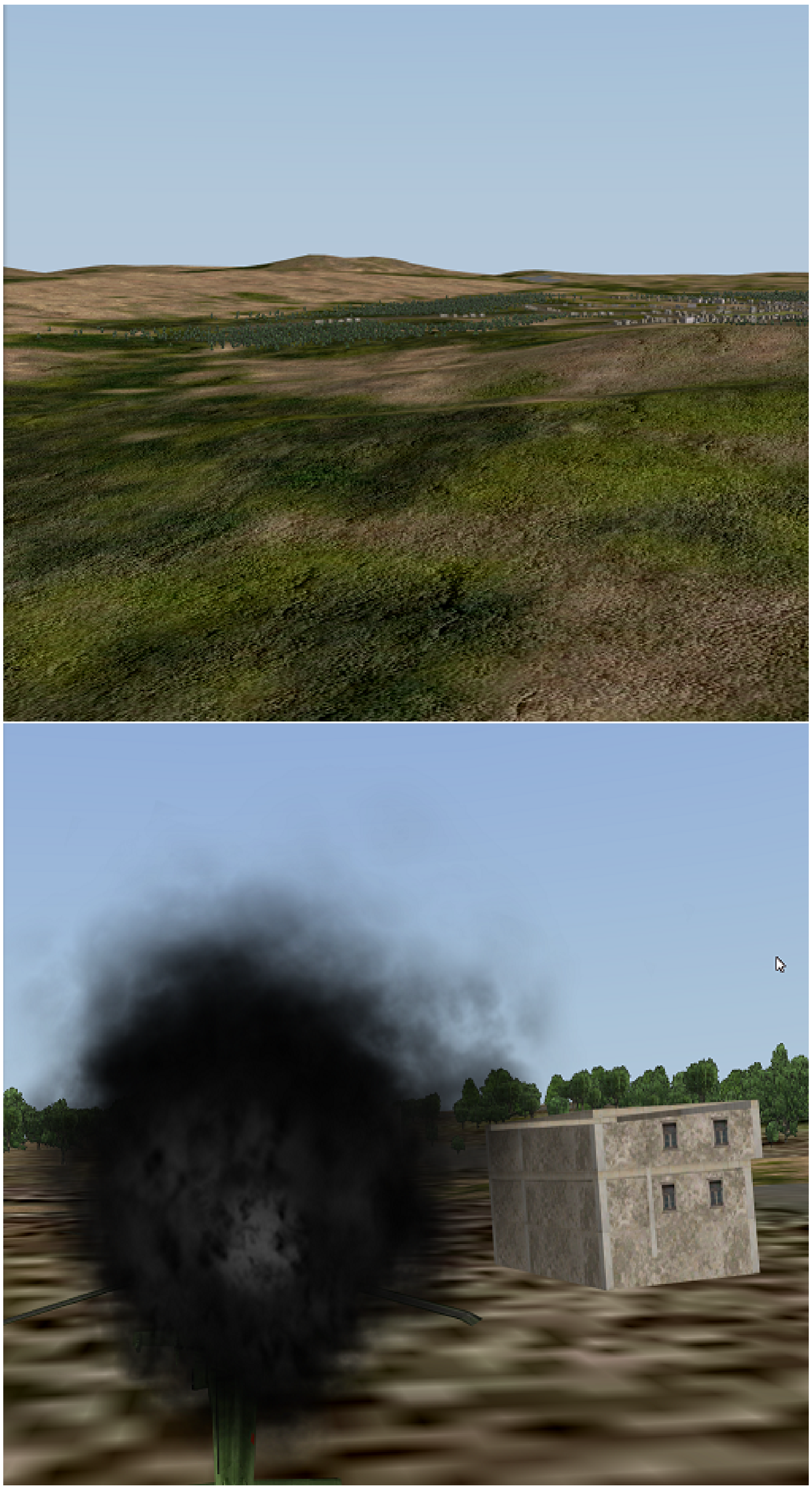


Figure 3.4: A few snapshots from the rendering system.

receiving help. In these results, the helping GPU sends a frame every 50ms. In the blocking case, we see that, if the IG is fast enough, the received help cannot increase its refresh rate because getting help means waiting for a packet from the network, which costs more than rendering the frame itself. In the non-blocking case, however, whatever how fast the IG is, receiving help increases its refresh rate. However, the amount of increase decreases when the speed increases.

In Figure 3.6, we see the effect of the Network Sharing Method on the refresh rate of the helping IG. We see that the helping IG is only slightly affected. This is due to the fact that the IG prepares and sends frames over the network in a separate thread than the one rendering the scene.

3.3.2 GPU Sharing Method

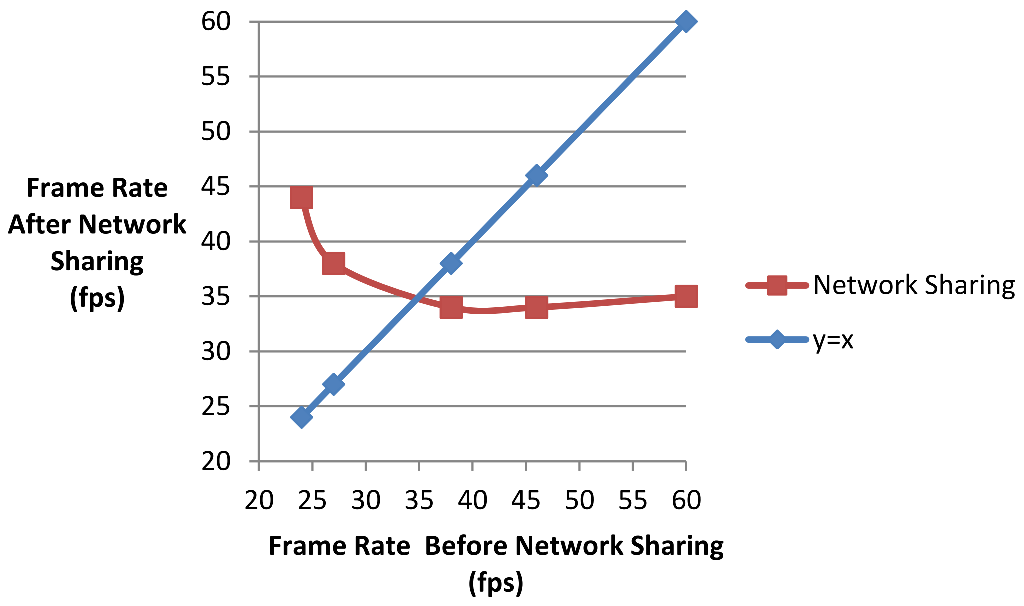
In Figure 3.7, we analyze the effect of the received help on the refresh rate of the GPU receiving help. The helping GPU sends every frame to the helpee GPU. We observe that, whatever the slow GPU's frame rate is, the help from the fast GPU leads to approximately similar frame rates, both in the blocking and the non-blocking cases. This is due to the fact that shared memory allows very fast transfer of frame to the helpee GPU and the helpee GPU does not get much chance to render a frame itself. Therefore, Figure 3.7 shows us that GPU-GPU sharing has a limit, no matter the original speed of the helpee GPU.

In Figure 3.8, we see the effect of the GPU Sharing Method on the refresh rate of the helping GPU. We see that the helping GPU is very much affected. This is due to the fact that the time spent for the helping GPU to copy the rendered data to the shared memory is roughly three times than rendering a scenes.

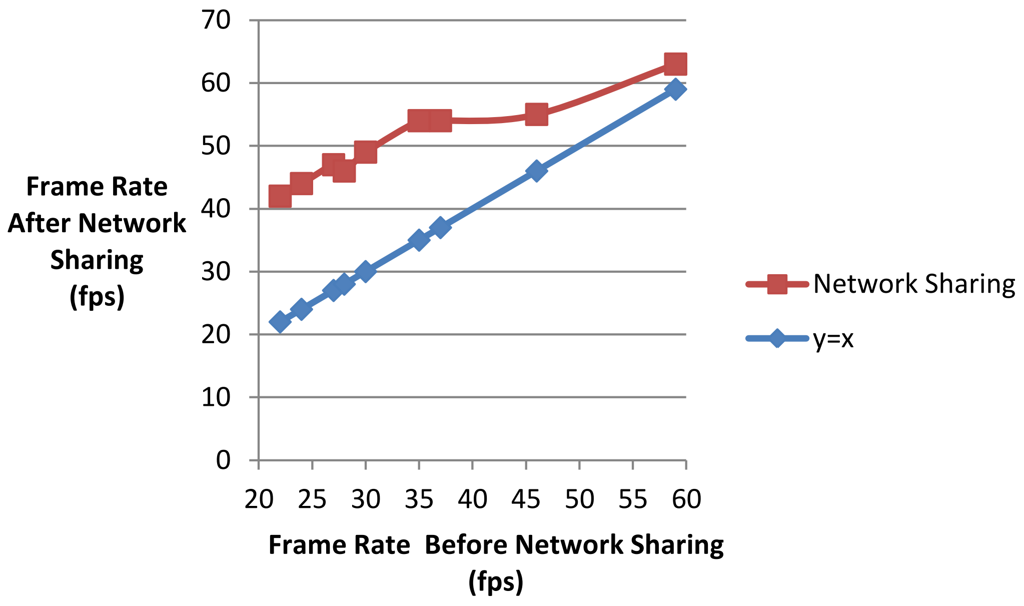
3.3.3 Guidelines

Based on the results provided in this section, we provide the following guidelines:

1. Although the distributed rendering literature has mostly focused on intra-frame load distribution strategies, inter-frame rendering is still plausible despite the



(a) Blocking



(b) Non-blocking

Figure 3.5: The effect of Network Sharing Method on the IG receiving help. (a) With blocking the IG receiving help, (b) Non-blocking the IG receiving help.

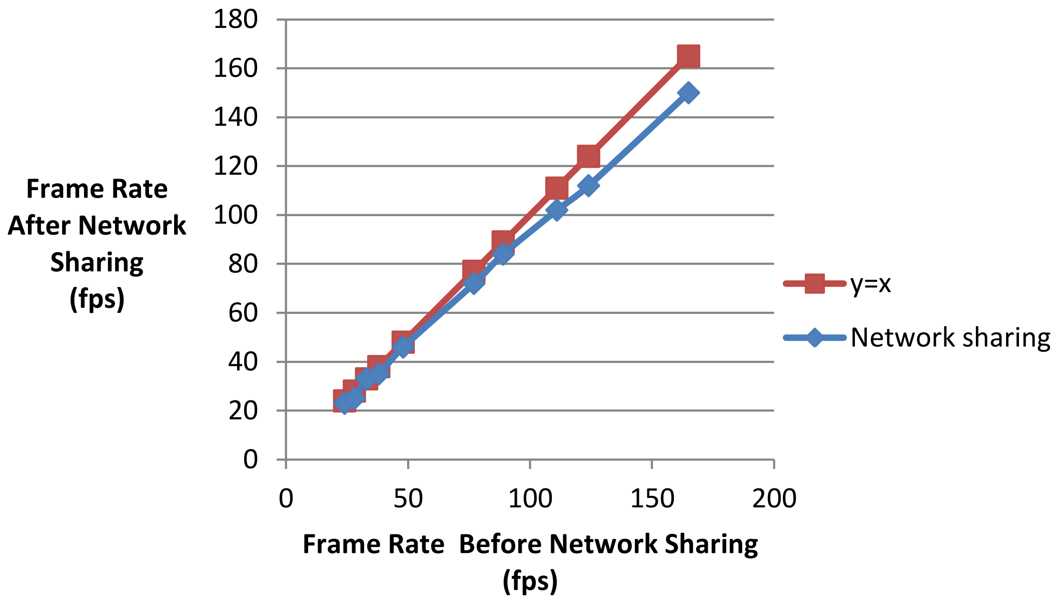


Figure 3.6: The effect of Network Sharing Method on the IG giving help.

network latency.

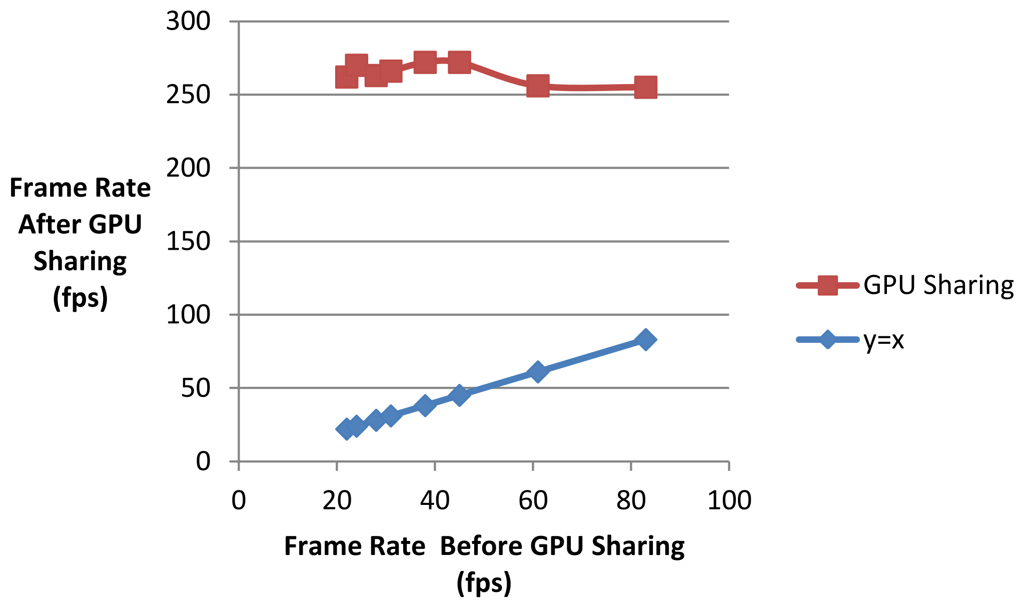
2. For overcoming the network latency problem, several strategies can be adopted based on the demands of the rendering problem. One is blocking the rendering node while waiting for the frame from another IG, and the other approach is non-blocking the IG receiving help, and taking care of the frame coherence issue by making sure that the frames are sent at least Δt_N before the IG finishes rendering its own frame.

3. GPU-GPU sharing is much faster than IG-IG sharing. The latency problem, though less severe, exists for GPU-GPU communication as well. The same solutions proposed for IG-IG communication apply to GPU-GPU latency.

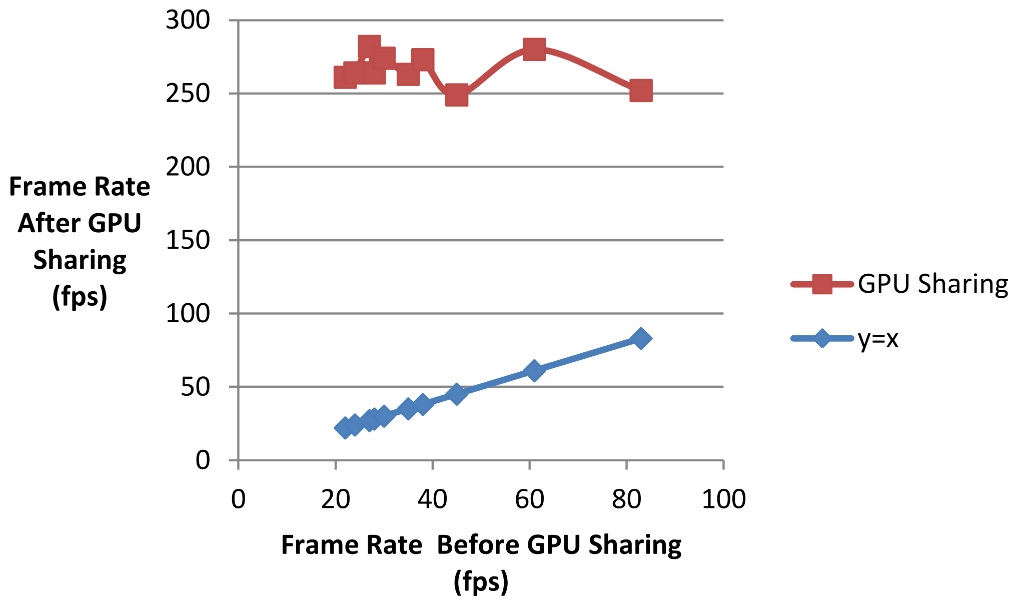
4. For enhanced distributed rendering, IG-IG and GPU-GPU distributed rendering should both be utilized.

3.4 Conclusion

The chapter has investigated the bottlenecks in parallel and distributed rendering systems with simulations. It has shown that in a locally distributed rendering system, the



(a) Blocking



(b) Non-blocking

Figure 3.7: The effect of GPU Sharing Method on the GPU receiving help. (a) With blocking the GPU receiving help, (b) Non-blocking the GPU receiving help.

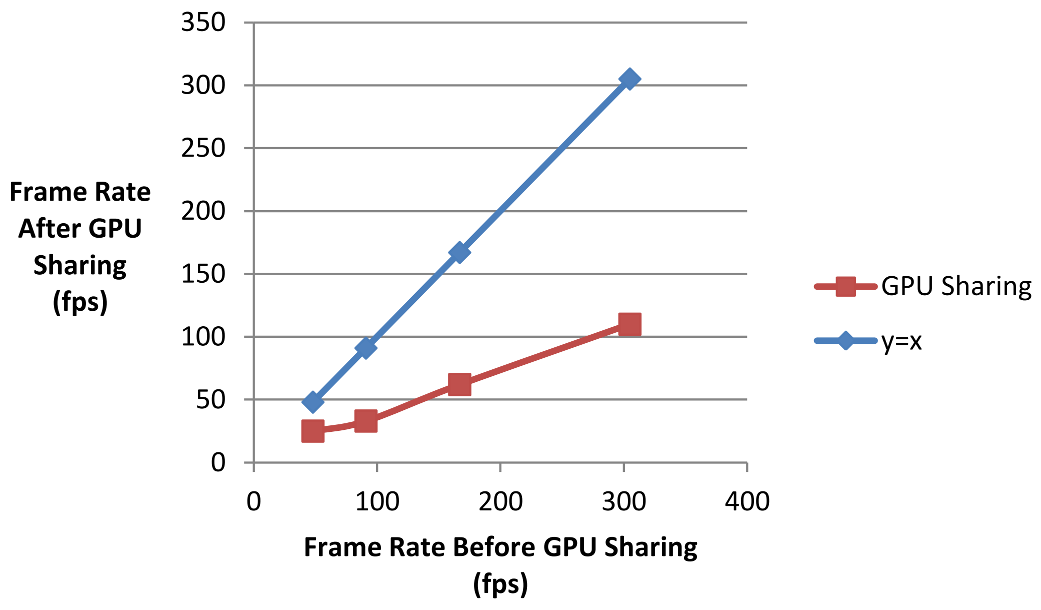


Figure 3.8: The effect of GPU Sharing Method on the GPU giving help.

transfer from one GPU to the other needs to go over the CPU and the memory, which is a limiting factor.

Moreover, for distributed rendering using a network of computers, the network speed is a bottleneck. We argue that, under these bottlenecks, rendering can be distributed provided that the rendering speed of a processing unit is slow enough to compensate for the time delay for the data transfer, either in the computer or in the network.

CHAPTER 4

DISTRIBUTED RENDERING USING A NETWORK OF COMPUTERS

In this chapter, a method for distributed rendering of huge data among a network of computers is described. In the network, there are computers with different rendering performances all in the same network with a server, which collects rendering performances and performs load balancing. The chapter shows that such a system can increase the rendering performance of slow computers with the help of the fast ones.

The proposed solution in this chapter depend on the inter-frame rendering approach. In inter-frame rendering, distribution among computers is done for complete frames of the scene. The load balancing strategy proposed is dynamic and adaptive to new conditions which result from load imbalances at run-time.

4.1 Formulation

In our problem, we have a set of image generators (IGs), $I = \{IG_1, \ldots, IG_n\}$ which are connected to each other and a server over a network (Figure 4.1). The server functions as the load balancer for the IGs. Each IG_i has an estimated frame rate f_i and the goal is to redistribute the load between the IGs such that frame rate for each IG is higher than a predefined minimum value, i.e.,

$$f_i > \tau, i = 1, ..., n. \tag{4.1}$$

Among the IGs, some has frame rates lower than τ and some higher than τ; in other

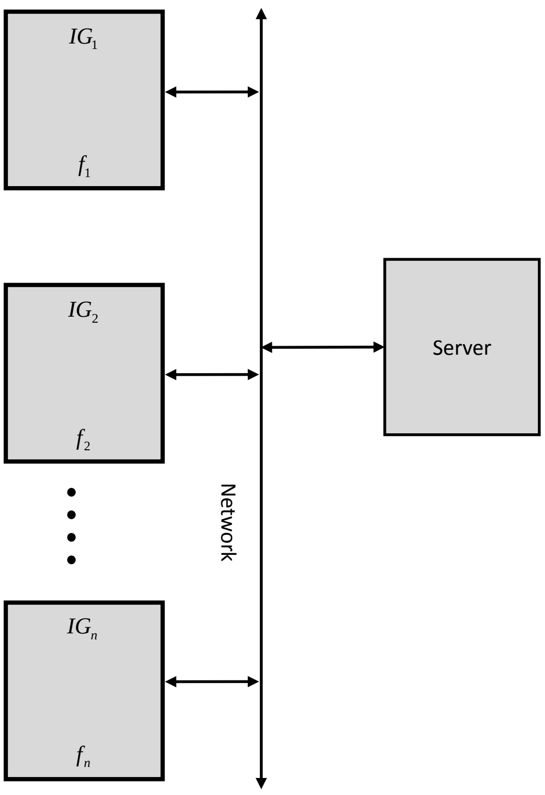


Figure 4.1: The overview of the setup.

words, we can partition I into slow IGs (S) and fast IGs (F), i.e.,

$$I = S \cup F, \tag{4.2}$$

where S and F are defined as:

$$S = \{IG_i | IG_i \in I, f_i < \tau\}, \tag{4.3}$$

and

$$F = \{IG_i | IG_i \in I, f_i > \tau\}. \tag{4.4}$$

4.2 A Distributed Solution

The solution presented in this chapter can be summarized as follows:

1. Sort S based on the IG frame rates in <u>increasing</u> order. Let S_{sorted} denote the sorted *slow* IGs.
2. Sort F based on the IG frame rates in <u>decreasing</u> order. Let F_{sorted} denote the sorted *fast* IGs.

3. Let IG_s be the top IG in S_{sorted}. Determine k IGs from F_{sorted} that will help IG_s. Repeat this step for each IG in S_{sorted}:

```
**for all** $IG_s \in S_{sorted}$ **do**
    - choose $k$ IGs from $F_{sorted}$ to help, and add them to $H(IG_s) = \{IG_i, ..., IG_h\}$
    - remove the $k$ IGs from $F_{sorted}$
**end for**
```

4. For each IG in S_{sorted}, calculate the frequency with which the IGs in $H(IG_s)$ will help.

4.2.1 Determining k

Let us say that IG_s has frame rate $f_s < \tau$, and its deficiency from the minimum number of frames per second, i.e., $\tau - f_s$, is going to be helped by IGs from F_{sorted}. For determining k, we can employ Algorithm 1.

4.2.1.1 Deriving number of frames an IG can help

In Algorithm 1, c_f, the number of frames that IG_f can help, is calculated from its frame rate (f_f), the frame rate threshold (τ) and the (time) cost of sending a frame to the network (Δt_f). The main idea is that the fast IG can only discard frames (in a second) provided that its updated frame rate is above threshold. Below is the derivation of c_f.

$$\{f_f\} - \{\text{ \# frames that I can send } (c_f)\} \times \{\text{ Time loss for sending data}\} \geq \{\text{ Threshold } (\tau)\} \quad (4.5)$$

$$\rightarrow f_f - c_f \times \frac{\{\text{ Time for sending a single frame } (\Delta t_f)\}}{\{\text{ Time for a single frame }\}(1/f_f)} \geq \tau \quad (4.6)$$

$$\rightarrow f_f - c_f \times \frac{\Delta t_f}{1/f_f} > \tau \rightarrow c_f \leq \frac{f_f - \tau}{\Delta t_f \times f_f} \quad (4.7)$$

$$\Rightarrow c_f = \left\lfloor \frac{f_f - \tau}{\Delta t_f \times f_f} \right\rfloor . \quad (4.8)$$

Algorithm 1 The algorithm for determining the number of helping IGs for a slow IG.

```
Input   $F_{sorted}, S_{sorted}$   :  The list of fast and slow IGs
Output  $H(IG_s), l_f^s$           :  Helping IGs and helping frequencies

for all $IG_f \in F_{sorted}$ do
    $f_f \leftarrow$ frame rate for $IG_f$
    $c_f \leftarrow \left\lfloor \frac{f_f - \tau}{1 + \Delta t_f f_f} \right\rfloor$  /* Number of frames $IG_f$ can give away */
end for
for all $IG_s \in S_{sorted}$ do
    $H(IG_s) \leftarrow \emptyset$
    $f_s \leftarrow$ frame rate for $IG_s$
    $k \leftarrow 0$
    $g_s \leftarrow \left\lceil \frac{\tau - f_s}{1 - f_s \Delta t_s} \right\rceil$
    for all $IG_f \in F_{sorted}$ do
        if $c_f < 0$ then
            /* This IG is not fast enough, skip it */
            continue
        end if
        $k \leftarrow k + 1$
        $H(IG_s) \leftarrow H(IG_s) + IG_f$
        if $g_s > c_f$  /* Take this IG and look for more helpers */ then
            $g_s \leftarrow g_s - c_f$
            $l_f^s \leftarrow \frac{f_f}{c_f}$
            $c_f \leftarrow 0$
        else
            /* With this IG, I have got all the help I need. */
            $c_f \leftarrow c_f - g_s$
            $l_f^s \leftarrow \frac{f_f}{g_s}$
            $g_s \leftarrow 0$
            break
        end if
    end for
end for
```

4.2.1.2 Deriving number of frames an IG needs

In Algorithm 1, the minimum number of frames (g_s) that an IG needs is calculated from is calculated from its frame rate (f_s), the frame rate threshold (τ) and the (time) cost of receiving a frame from the network (Δt_s). g_s can be derived from the following two constraints:

- **First constraint:** The only way a slow IG can spare time for getting help from other IGs and displaying those frames is not rendering a number of frames:

$$\Big\{ \text{\# frames that IG will stop rendering } (x) \Big\} \times \Big\{ \text{Time for rendering a frame } (1/f_s) \Big\} \tag{4.9}$$

$$\geq \Big\{ \text{\# received frames} \Big\} \times \Big\{ \text{cost of the received frame} \Big\} \tag{4.10}$$

$$\rightarrow x\frac{1}{f_s} \geq g_s \times \Delta t_s \tag{4.11}$$

$$\rightarrow x\frac{1}{f_s} - g_s \times \Delta t_s \geq 0. \tag{4.12}$$

- **Second constraint:** The remaining frames that the slow IG renders and the ones that it receives from others should be above the threshold:

$$\Big\{ \text{\# frames that I continue rendering } (f_s - x) \Big\} \times \Big\{ \text{Received frames } (g_s) \Big\} \geq \tau \tag{4.13}$$

$$\rightarrow f_s - x + g_s \geq \tau \tag{4.14}$$

$$\rightarrow f_s - x + g_s + T \geq 0 \tag{4.15}$$

Constraints 1 and 2 in Equations 4.12 and 4.15 can be combined as follows:

- Multiply Equation 4.15 by $1/f_s$ to get g_s:

$$g_s \geq \frac{\tau - f_s}{1 - f_s \times \Delta t_s} \tag{4.16}$$

$$\Rightarrow \text{minimum possible } g_s = \left\lceil \frac{\tau - f_s}{1 - f_s \times \Delta t_s} \right\rceil. \tag{4.17}$$

- Multiple Equation 4.12 by Δt_s to get x:

$$x \geq f_s \Delta t_s \frac{\tau - f_s}{1 - f_s \times \Delta t_s} \tag{4.18}$$

$$\Rightarrow x \approx f_s \Delta t_s g_s. \tag{4.19}$$

4.2.2 Determining help frequencies

We are given the helping IGs $H(IG_s) = \{IG_{s_1}, \ldots, IG_{s_m}\}$, which have frame rates $f_{s_1}, \ldots, f_{s_m}$ higher than τ, for a slow IG IG_s with frame rate $f_s < \tau$.

Assume that IG_f with $f_f \geq \tau$ helps a slow IG, IG_s, with $f_s < \tau$. Also assume that IG_f helps IG_s with c_f^s number of frames per second. IG_s needs a frame from IG_f regularly, i.e., every l^{th} frame, which can be calculated easily by:

$$l = \left\lfloor \frac{f_f}{c_f^s} \right\rfloor, \tag{4.20}$$

where $c_f^s < c_f^{max}$ is the number of frames that IG_f gives away to IG_s.

4.3 Consideration of Other Factors

In our proposed algorithm, the frame rate is considered as a load balancing criteria. Along with frame rate, other criteria such as network bandwidth and rendered triangles per second can be used in determining helping and helpee IGs and help frequencies.

4.3.1 Network Bandwidth

In the application of algorithm, when the frequencies of sent images over network increase, the network bandwidth may not be sufficient for handling the transmission. In a 1-Gb network, it is possible to send at most 30 colored images at 1024×1024 resolution. However, the number of images sent depends on the switch capacity and load. In our algorithm, in determining the helping frequencies, the following criteria should be considered:

$$\frac{f_f}{l} < 30, \tag{4.21}$$

where l is the frequency at which a fast IG helps a slow IG, and f_f is the frame rate of the fast IG. In case the fast IG helps more than one IG, on the other hand, the total number of frames that the fast IG can help should be limited by 30.

The highest number of images that can be sent over network can be increased by decreasing the size of images using compression algorithms. There are various lossless compression algorithms [4, 30, 41, 43] such as JPG, TGA (using RLE compression), TIF (using LZW compression which is an extension of LZ77 and LZ78 algorithms [51, 52]) and PNG. Variances of these algorithms exist. JPG gives very good results in compression up to compression ratios of 5:1. This results in the increase of maximum number of frames that can be sent over network, which means $30 \times 5 = 150$ frames in the case of a compression ratio of 5:1. Actually, while the compression ratio is decreased, the time for compression and decompression times are introduced as overheads which increase Δt_f and Δt_s in our algorithm. And this ends up in reducing the frame rates and so requiring higher helping frequencies. This trade-off can be considered as a criteria in determining help frequencies.

4.3.2 Rendered Triangles per Second

The load which is introduced during visualization depends on different factors. One of these is the rendered triangles per second. Rendered triangles per second depends on various parameters such as the camera position, camera orientation, rendered LODs and the number of objects. When the number of rendered triangles per second needs to be increased, this causes a decrease in the frame rate. For this reason, trying to decrease the number of rendered triangles by some methods such as using LOD is a load balancing scheme in its way. There is a relation of frame rate with rendered triangles per second. However, frame rate is not only affected by the number of rendered triangles per second, but also other processes such as physical and network computations. Thus, the number rendered triangles per second may not be considered as a unique constraint, but it can be used complimentary to and together with other constraints.

4.4 Results

The experiments were performed on the same cluster used for the experiments in the chapter 3. The network on which the computers are connected to each other is a 1-

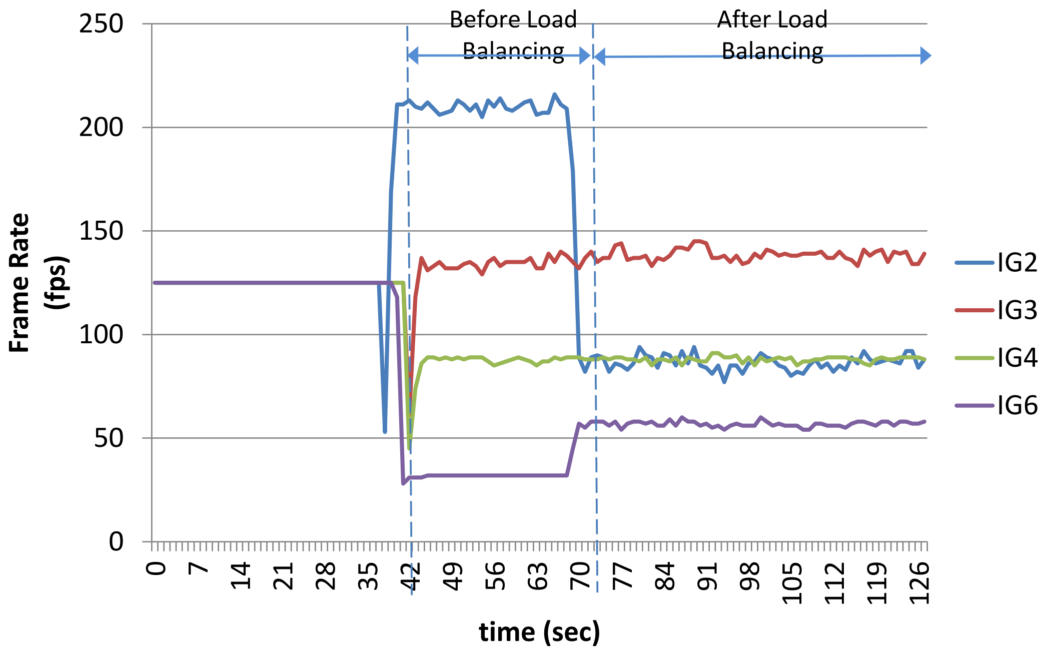


Figure 4.2: Effect of load balancing in the case of three fast IGs and one slow IG.

Gb Ethernet, i.e., not a very fast network. The heterogeneous system was obtained adding extra different amounts of artificial scene load to different IGs. The IGs are commanded by the Central Control Computer to load the scene including terrain, models and other effects. The results below were obtained including the loading phase and after loading phase which also includes states when load balancing strategy is on/off.

According to the results in (Figure 4.2), there are three fast IGs (IG_2, IG_3 and IG_4) which have higher frame rates than threshold and one slow IG (IG_6) which has lower frame rate than threshold. After applying our load balancing algorithm described above, IG_2, which is the fastest one, helps the slow IG (IG_6). It can be seen that the help frequencies supplied by IG_2 are enough for IG_6 so that IG_6 becomes a faster IG, which does not need any extra help from other IGs (IG_3 and IG_4).

The results in (Figure 4.3) show that there is one fast IG (IG_2) which has higher frame rates than threshold and three slow IGs (IG_5, IG_6 and IG_7) which have lower frame rates than threshold. After applying our load balancing algorithm described above, IG_2, which is the fastest one, helps two slow IGs (IG_6 and IG_7) so that these two slow IGs become faster. However, for the fast IG, IG_2, it is not possible to help IG_5

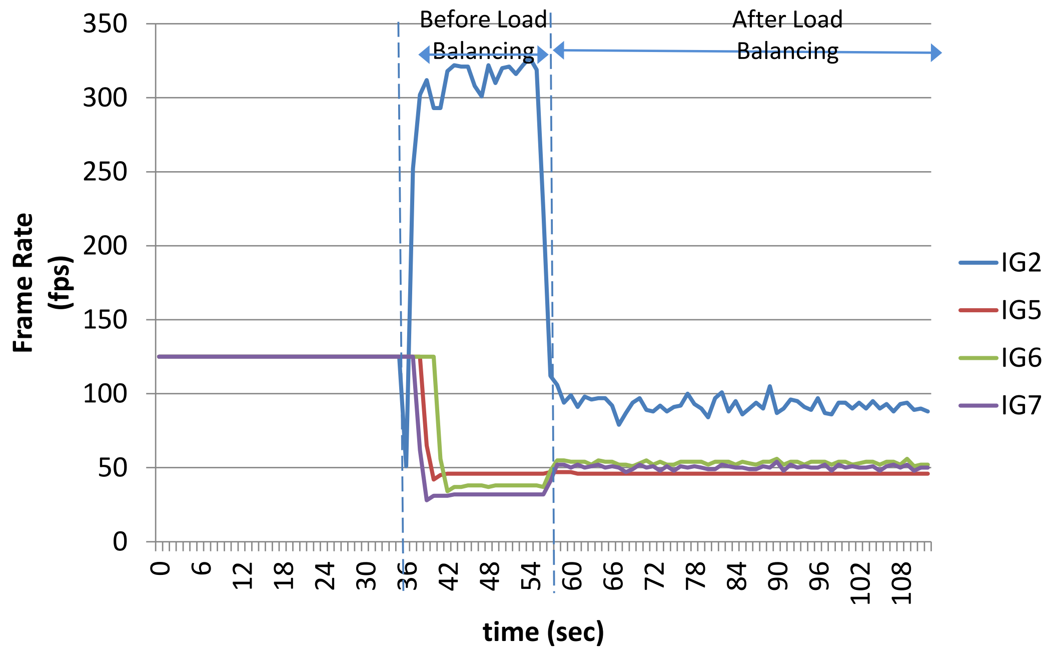


Figure 4.3: Effect of load balancing in the case of one fast IG and three slow IGs.

otherwise its threshold becomes lower than the threshold.

The results in (Figure 4.4) show that there are two fast IGs (IG_1 and IG_2) which have higher frame rates than threshold and two slow IGs (IG_5 and IG_8) which have lower frame rates than threshold. After applying our load balancing algorithm described above, IG_1, which is the fastest one, first helps the slowest IG (IG_8) so that IG_8 becomes faster. The second slower IG, IG_5, gets help also from the fastest IG_1, however, help from IG_1 is not enough for IG_5 to achieve frame rate higher than threshold after IG_1 has started helping IG_8. To achieve frame rate higher than threshold, IG_5 gets help from IG_2 which is one of the fastest IGs.

Moreover, the results show that the network bandwidth criteria does not create a bottleneck since the number of frames needed to be sent calculated according to the help frequencies do not exceed the maximum number of frames that can be sent according to the network bandwidth limit. However, in cases of distributed systems including many more number of computers over wide area networks, network bandwidth criteria becomes important and should be considered regarding the trade-off between compression ratio and compression and decompression process durations.

By the results, it is shown that a fast IG can help more than one slow IG, unless the

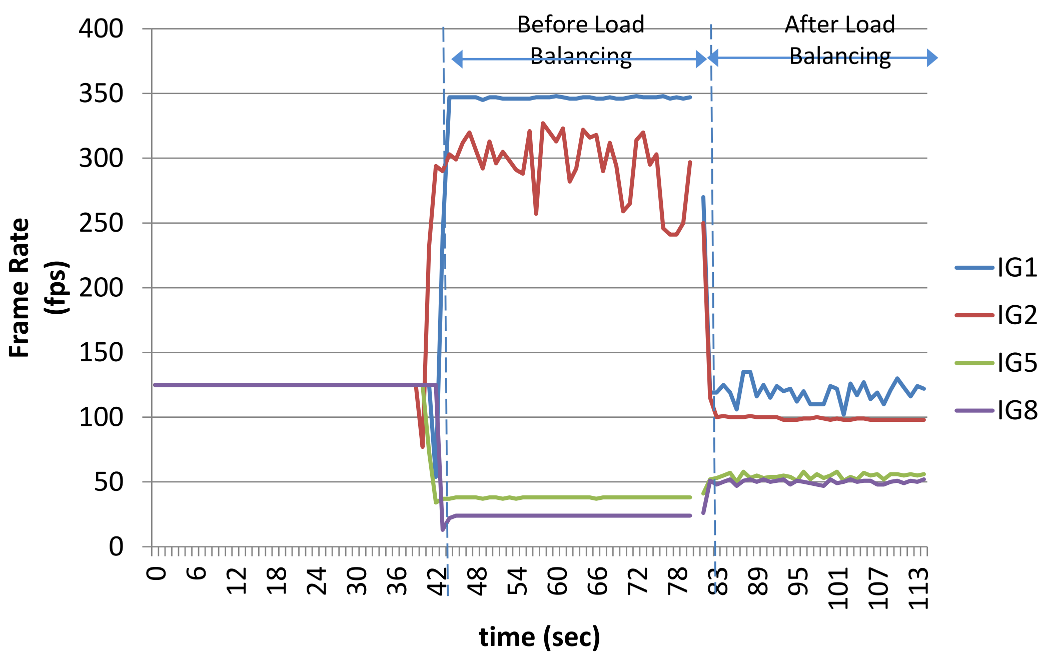


Figure 4.4: Effect of load balancing in the case of two fast IGs and two slow IGs.

frame rate of the fast IG does not decrease under the threshold. On the other hand, a slow IG can get help from more than one fast IG to achieve a high frame rate than the threshold.

4.5 Conclusion

The distribution solution depends on the principle of fast IGs helping slow IGs to help them increase their frame rates over the threshold. To achieve this goal, it is possible for more than one fast IGs to help more than one slow IGs. The results show that such our distribution solution can increase the rendering performance of slow IGs with the help of the fast ones, meeting the goal.

Our distribution rendering solution depends on inter-frame approaches. It works on a network of connected computers and a server. Different from other works which require advanced very fast network, an ordinary fast network infrastructure which can be obtained at low prices is enough for our algorithm to succeed.

The solution also differs from other works in the way that it is adapted to new conditions. New conditions here mean variance in load during the rendering of the scene, variance in load of each computer or variance on network delays. Server, which works as a centralized load balancer, provides this adaptation using the periodic statistical values gathered from each computer.

CHAPTER 5

A HYBRID SYSTEM FOR DISTRIBUTED RENDERING

In this chapter, a hybrid method for distributed rendering of huge data is proposed. In the distributed rendering system, there is a network of computers with multiple GPUs. In this system, a GPU can either help another GPU in the same IG or another IG in the network. Moreover, both inter-frame type of distributed rendering as well as sort-last type intra-frame distributed rendering are considered. In other words, there are two aspects of hybridity: (i) locally and network-wide distributed rendering, and (ii) inter-frame and intra-frame distributed rendering.

In the network, there are computers equipped with multiple GPUs with different rendering performances all in the same network with a server, which collects rendering performances of the GPUs in the different IGs and performs load balancing. The chapter shows that such a system can increase the rendering performance of slow computers with the help of the fast ones.

5.1 Formulation

Similar to the problem formulation in Section 4.1, in the hybrid problem, there is a set of image generators (IGs), $I = \{IG_1, \ldots, IG_n\}$ with multiple GPUs which are connected to each other and a server over a network (Figure 5.1). The server functions as the load balancer for the IGs. Each GPU$_{ji}$, where j denotes that the GPU is on IG_j, has an estimated frame rate f_{ji} and the goal is to redistribute the load between the IGs and the GPUs such that frame rate for each IG is higher than a predefined minimum

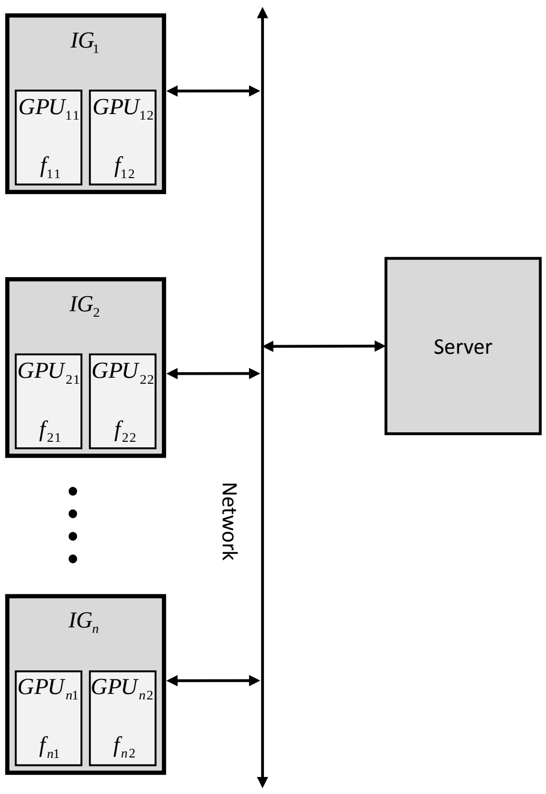


Figure 5.1: The overview of the hybrid setup.

value, i.e.,

$$f_{ji} > \tau, j = 1, ..., n, \text{ and } i = 1, ..., m. \tag{5.1}$$

In our case, m is two due to the setup used for the experiments; however, the algorithms are applicable for $m > 2$ too. For an IG_j, GPU_{j1} is called the primary GPU whereas GPU_{j2} is called the secondary GPU.

There are different rendering tasks $\{T_1, ..., T_o\}$ (such as multitexturing, occlusion culling, particles, fog simulation, illumination, vegetation) that a GPU can take care of, and for each task, the frame rate is different. Let f_{ji}^T denote the frame rate of GPU_{ji} for performing task T. One can easily estimate the frame rate, call it $\hat{f}_{ji}$ for GPU_{ji} as the minimum for the tasks:

$$\hat{f}_{ji} = \min\left(\{f_{ji}^T\}_{T=1}^{T=o}\right). \tag{5.2}$$

Then, considering the minimum $\hat{f}_{ji}$ for $l = 1, ..., o$ for primary GPUs, we can partition the set of IGs I into slow IGs (S) and fast IGs (F), i.e.,

$$I = S \cup F, \tag{5.3}$$

where S and F are defined as:

$$S = \left\{GPU_{i1} \mid IG_i \in I,\ \hat{f}_{i1} < \tau\right\}, \tag{5.4}$$

and

$$F = \left\{G \mid IG_i \in I, G \in \{GPU_{i1}, GPU_{i2}\},\ \hat{f}_G > \tau\right\}. \tag{5.5}$$

In other words, the primary GPUs which are slow can receive help from all the GPUs (primary or secondary) that are fast. For the sake of simplicity and in order to be able to use the same way of thinking and derivations in Chapter 4, each GPU will be considered as an IG in the rest of the chapter without loss of generality.

5.2 A Hybrid Solution

The solution presented in this chapter can be summarized as follows:

1. Sort S based on <u>their gain</u> in <u>increasing</u> order. Let S_{sorted} denote the sorted *slow* GPUs. The gain of a GPU_i is equal to its frame rate ($\hat{f}_i$) minus a factor of the cost of helping the slow GPU. Note that the cost of a secondary GPU helping a primary one is much less than that of helping another IG in the network.
2. Sort F based on <u>their gain</u> in <u>decreasing</u> order. Let F_{sorted} denote the sorted *fast* GPUs.
3. Let GPU_s be the top GPU in S_{sorted}. For each task T, determine k^T IGs from F_{sorted} that will help IG_s. Repeat this step for each IG in S_{sorted}:

```
for all $IG_s \in S_{sorted}$ do
    for all task $T$ do
        - choose $k^T$ IGs from $F_{sorted}$ to help, and add them to $H^T(IG_s) = \{IG_i, ..., IG_h\}$
        - update the gain values of the $k^T$ IGs from $F_{sorted}$
    end for
end for
```

4. For each IG in S_{sorted}, calculate the frequency with which the IGs in $H^T(IG_s)$ will help.

5.2.1 Determining k^T

Let us say that the primary GPU of IG_s has frame rate $\hat{f}_{s1} < \tau$, and its deficiency from the minimum number of frames per second, i.e., $\tau - \hat{f}_{s1}$, is going to be helped by IGs from F_{sorted}. For determining k^T for a task T, we can employ Algorithm 2, which is a slight modification of Algorithm 1.

5.2.1.1 Deriving number of frames an IG can help

In Algorithm 2, c_f, the number of frames that IG_f can help, is calculated from its frame rate (f_f), the frame rate threshold (τ) and the (time) cost of sending a frame to the network (Δt_f) as follows:

$$c_f = \left\lfloor \frac{f_f - \tau}{\Delta t_f \times f_f} \right\rfloor . \tag{5.6}$$

The derivation of c_f is the same as presented in Section 4.2.1.1.

5.2.1.2 Deriving number of frames an IG needs

In Algorithm 2, the minimum number of frames (g_s^T) that an IG needs is calculated from its frame rate (f_s^T) for task T, the frame rate threshold (τ) and the (time) cost of receiving a frame for task T (Δt_s). g_s^T can be derived from the following two constraints:

- **First constraint:** The only way a slow IG can spare time for getting help from other IGs and displaying those frames is not rendering a number of frames:

$$\Big\{ \text{# frames that IG will stop rendering } (x) \Big\} \times \Big\{ \text{Time for rendering a frame } (1/f_s) \Big\} \tag{5.7}$$

$$\geq \Big\{ \text{# received frames} \Big\} \times \Big\{ \text{cost of the received frame} \Big\} \tag{5.8}$$

$$\rightarrow x\frac{1}{f_s} \geq g_s^T \times \Delta t_s \tag{5.9}$$

$$\rightarrow x\frac{1}{f_s} - g_s^T \times \Delta t_s \geq 0. \tag{5.10}$$

Algorithm 2 The algorithm for determining the number of helping IGs for a slow IG for a task in the hybrid case - this is a slight modification of Algorithm 1.

Input	F_{sorted}, S_{sorted}	:	The list of fast and slow IGs
Output	$H^T(IG_s), l^T_{f\to s}$	:	Helping IGs and helping frequencies

```
 1: for all $GPU_f \in F_{sorted}$ do
 2:     $f_f \leftarrow$ frame rate for $GPU_f$ as estimated in Equation 5.2
 3:     $c_f \leftarrow \left\lfloor \frac{f_f - \tau}{1+\Delta t_f f_f} \right\rfloor$ /* Number of frames $GPU_f$ can give away */
 4: end for
 5: for all $IG_s \in S_{sorted}$ do
 6:     $H_T(IG_s) \leftarrow \emptyset$
 7:     $f^T_s \leftarrow$ frame rate for $IG_s$
 8:     $k^T \leftarrow 0$
 9:     $g^T_s \leftarrow \left\lceil \frac{\tau - f^T_s}{1 - f^T_s \Delta t_s} \right\rceil$
10:     for all $GPU_f \in F_{sorted}$ do
11:         if $c_f < 0$ then
12:             /* This GPU is not fast enough, skip it */
13:             continue
14:         end if
15:         $k^T \leftarrow k^T + 1$
16:         $H_T(IG_s) \leftarrow H_T(IG_s) + GPU_f$
17:         if $g^T_s > c_f$ /* Take this GPU and look for more helpers */ then
18:             $g^T_s \leftarrow g^T_s - c_f$
19:             $l^T_{f\to s} \leftarrow \frac{f_f}{c_f}$
20:             $c_f \leftarrow 0$
21:         else
22:             /* With this GPU, I have got all the help I need. */
23:             $c_f \leftarrow c_f - g^T_s$
24:             $l^T_{f\to s} \leftarrow \frac{f_f}{g^T_s}$
25:             $g^T_s \leftarrow 0$
26:             break
27:         end if
28:     end for
29: end for
```

- **Second constraint:** The remaining frames that the slow IG renders and the ones that it receives from others should be above the threshold:

$$\{\text{ \# frames that I continue rendering } (f_s - x)\} \times \{\text{ Received frames } (g_s^T)\} \geq \tau \tag{5.11}$$

$$\rightarrow f_s - x + g_s^T \geq \tau \tag{5.12}$$

$$\rightarrow f_s - x + g_s^T + \tau \geq 0 \tag{5.13}$$

Constraints 1 and 2 in Equations 5.10 and 5.13 can be combined as follows:

- Multiply Equation 5.13 by $1/f_s$ to get g_s^T:

$$g_s \geq \frac{\tau - f_s}{1 - f_s \times \Delta t_s} \tag{5.14}$$

$$\Rightarrow g_s^T = \left\lceil \frac{\tau - f_s}{1 - f_s \times \Delta t_s} \right\rceil . \tag{5.15}$$

- Multiple Equation 5.10 by Δt_s to get x:

$$x \geq f_s \Delta t_s \frac{\tau - f_s}{1 - f_s \times \Delta t_s} \tag{5.16}$$

$$\Rightarrow x \approx f_s \Delta t_s g_s^T . \tag{5.17}$$

5.2.2 Determining help frequencies

We are given the helping IGs $H^T(IG_s) = \{GPU_{s_1}, \ldots, GPU_{s_m}\}$ for task T, which have frame rates $\hat{f}_{s_1}, \ldots, \hat{f}_{s_m}$ higher than τ, for a slow IG IG_s with frame rate $f_s^T < \tau$ for task T.

Assume that GPU_f with $f_f \geq \tau$ helps a slow IG, IG_s, with $f_s^T < \tau$. Also assume that GPU_f helps IG_s with $c_{f \to s}$ number of frames per second. IG_s needs a frame from GPU_f regularly, i.e., every $l_{f \to s}^{th}$ frame, which can be calculated easily by:

$$l_{f \to s} = \left\lfloor \frac{f_f}{c_{f \to s}} \right\rfloor , \tag{5.18}$$

where $c_{f \to s} < c_f$ is the number of frames that GPU_f gives away to IG_s.

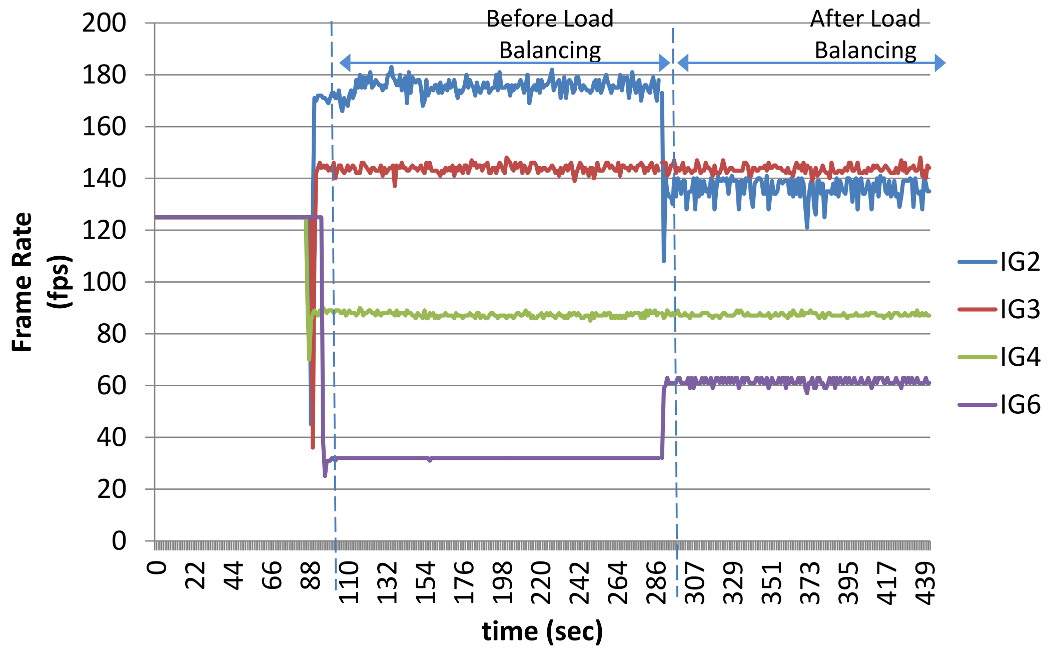


Figure 5.2: The effect of load balancing analyzed in different combinations of slow and fast IGs. The case with one slow IG and three fast IGs, where the second GPU of IG_2 helps IG_6, i.e., $H(IG_6) = \{IG_{22}\}$.

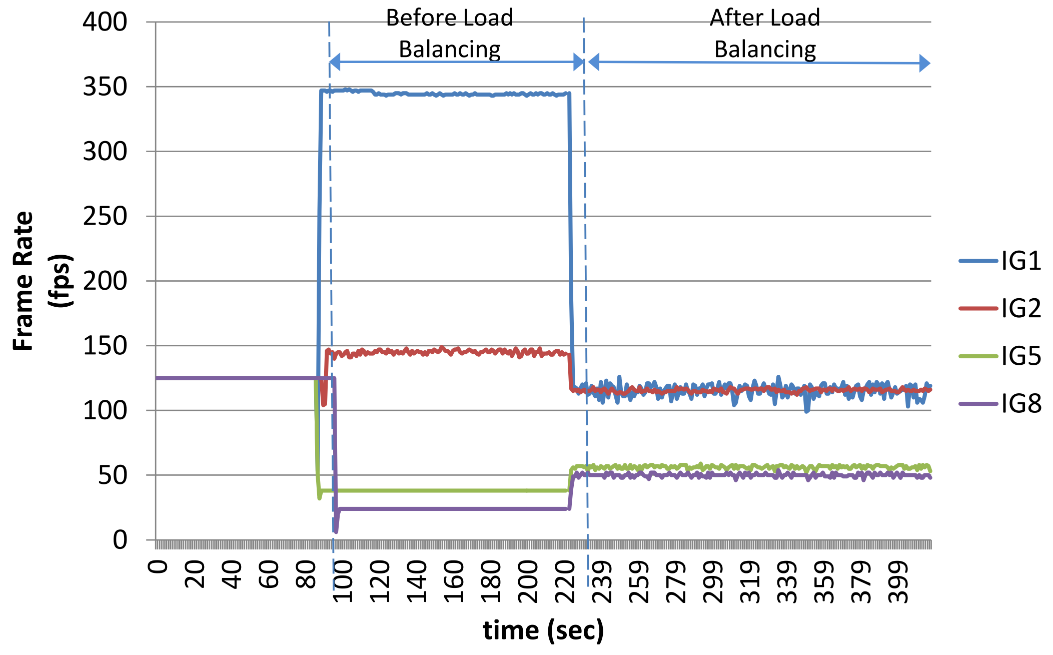


Figure 5.3: The effect of load balancing analyzed in different combinations of slow and fast IGs. The case with two fast IGs and two slow IGs, where $H(IG_8) = \{IG_{11}\}$ and $H(IG_5) = \{IG_{11}, IG_{22}\}$.

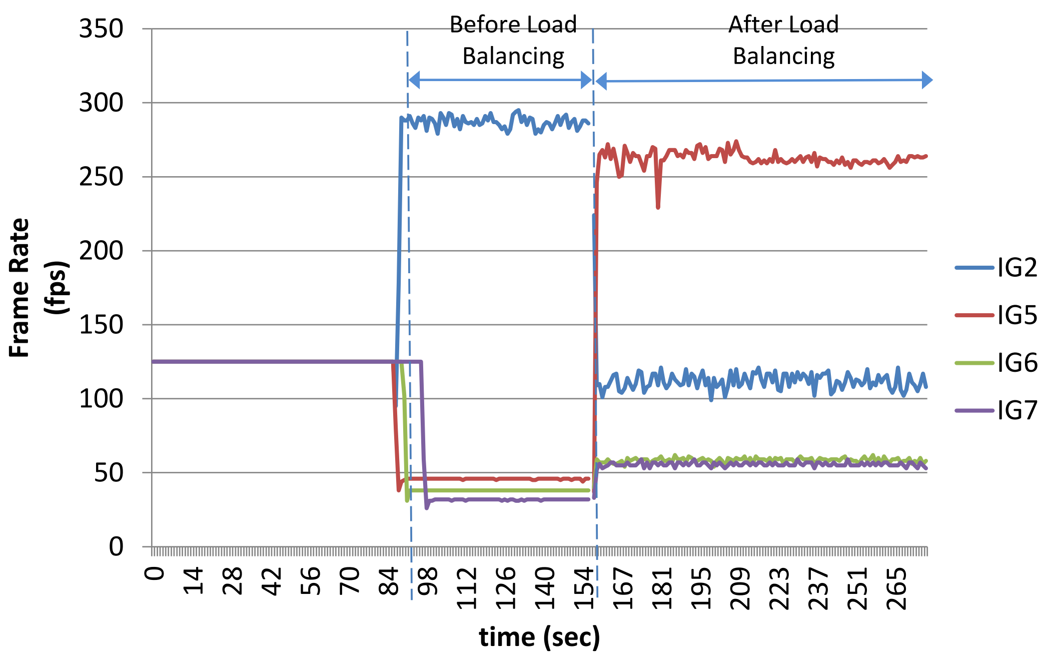


Figure 5.4: The effect of load balancing analyzed in different combinations of slow and fast IGs. The case with one fast IG and three slow IGs, where $H(IG_7) = \{IG_{22}\}$, $H(IG_6) = \{IG_{22}, IG_{52}\}$ and $H(IG_5) = \{IG_{52}\}$.

5.3 Results

The experiments were performed on the same cluster used for the experiments in the Chapter 3. The network on which the computers are connected to each other is a 1-Gb Ethernet, i.e., not a very fast network. Like in the previous chapter, the heterogeneous system was obtained adding extra different amounts of artificial scene load to different IGs. The IGs are commanded by the Central Control Computer to load the scene including terrain, models and other effects. The results below were obtained including the loading phase and after loading phase which also includes states when load balancing strategy is on/off.

In this section, the performance of the IGs before and after load balancing for the following combinations of slow and fast IGs are investigated, as shown in Figures 5.2, 5.3 and 5.4 - the cases are presented in increasing order of 'complexity':

- Figure 5.2 shows a case where there is one slow IG and three fast IGs. In this case, the second GPU of IG_2 helps IG_6, i.e., $H(IG_6) = \{IG_{22}\}$. Through load balancing, with the help of IG_6, the frame rate of IG_2 increases. Other IGs are unaffected by this load balancing process.

Although the second GPU of IG_2 is the helper, the frame rate of IG_2 decreases. The reason for this decrease is that, while sending frames of the second GPU over the network, the CPU becomes less available for the main GPU, effectively decreasing the frame rate.

- Figure 5.3 depicts a case of two fast IGs and two slow IGs, in which case the following assignments are performed:

$$H(IG_8) = \{IG_{11}\}, \quad \text{and} \tag{5.19}$$

$$H(IG_5) = \{IG_{11}, IG_{22}\}. \tag{5.20}$$

  In this assignment, a fast IG helps two slow IGs.

- In Figure 5.4, we have a case with one fast IG and three slow IGs. With load balancing, the proposed algorithm makes the following assignments:

$$H(IG_7) = \{IG_{22}\}, \tag{5.21}$$

$$H(IG_6) = \{IG_{22}, IG_{52}\}, \quad \text{and} \tag{5.22}$$

$$H(IG_5) = \{IG_{52}\}. \tag{5.23}$$

  In this case, one of the IGs, IG_5, helps both itself (i.e., the second GPU helps the main GPU) and another IG in the network. Moreover, IG_6 receives help from two IGs, and IG_2 helps two IGs.

In the preceding explanations, the results from inter-frame load balancing in both networked and local systems are shown. In Figure 5.5, load balancing effects are shown in a scene which includes extra load using smoke particles. In this case, IG_8 is helped by IG_{51} for the main scene part of the frame, and is helped by IG_{82}, i.e. by its second GPU, for the smoke particles part of the frame. Here, in the case where the rendering of the smoke particles is too expensive, the positive effect of intra-frame rendering is seen as the increase of frame rate. Although the composition of main scene (coming from IG_{51}) and particles (coming from IG_{82}) is expensive, since the

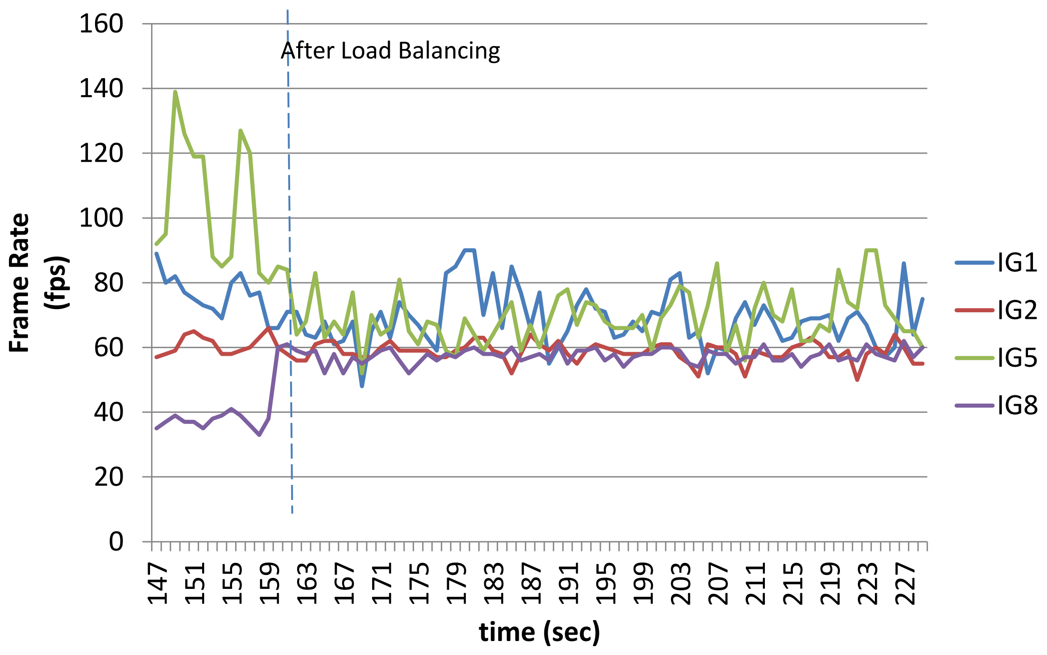


Figure 5.5: The effect of load balancing analyzed in different combinations of slow and fast IGs in the case of a smoky scene. The case with three fast IGs and one slow IG, where $H(IG_8) = \{IG_{51}, IG_{82}\}$ in a smoky scene.

rendering of particles and the main scene is much more expensive, it is the optimized configuration which our load balancing strategy decides on.

These results in this section show that, for different slow and fast IG combinations in a network of computers with multiple GPUs, the proposed load balancing algorithm can determine suitable assignment of fast-slow IGs and GPUs.

5.3.1 Making the Solution Dynamic and Adaptive

The solution proposed in this thesis is dynamic and adaptive, i.e. it uses the dynamic statistics coming from IGs. By means of this behavior, the solution can be easily adapted to the changes in the load of the scene and/or physical changes in network speed, etc. This is achieved by executing the algorithm every p seconds. However, in order to prevent the unnecessary computation load, the execution period p of the algorithm is incremented by one second if the load balancing assignments do not change, or decremented if there is a change. As seen in the graph in Figure 5.6, the execution period is incremented monotonously when there is no change in load or

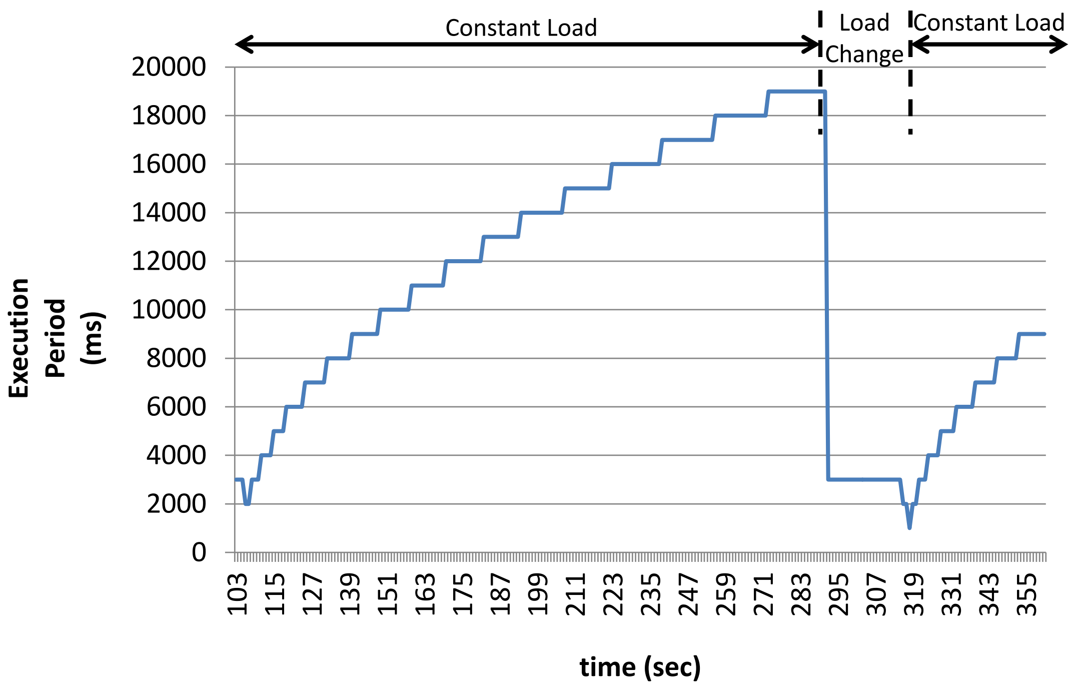


Figure 5.6: The effect of load changes on the execution period of the algorithm.

other physical states, however, it is decremented suddenly if there is a sudden change in the load to be adapted to the changes. In the decision of execution period, different techniques can be used such as proposed in [22].

5.4 Conclusion

In this chapter, a hybrid system is proposed. In this system, there is both an inter-frame-based distributed rendering as well as an intra-frame rendering used. For intra-frame rendering, smoke simulation particles are shared among the computers, which makes it a sort-last rendering system. In addition, in the hybrid system, a computer receives help either locally from its secondary GPU or from other computers in the network. Therefore, there are two aspects of hybridity: (i) inter-frame and intra-frame rendering, (ii) local (GPU-GPU) and network based rendering. By means of this efficient load balancing strategy, rendering performance of slow computers are increased with the help of the fast ones, so that the slow computers are no longer bottlenecks in heterogeneous systems.

CHAPTER 6

CONCLUSION AND FUTURE WORK

With the desire to visualize huge data or simulate complex scenes in high resolution, it has become a necessity to use parallel and distributed rendering techniques or architectures for fast, real-time, interactive simulation systems. Existing approaches either share individual frames (inter-frame methods) or parts of a single frame (sort-first, sort-last and sort-middle methods), and generally, they use advanced hardware connected together in very expensive ultra fast networks.

In this thesis, an inter-frame-based method (Chapter 4) is proposed. It is shown in Chapter 4 that, in a network of heterogeneous computers some which are fast and some of which are slow, the fast computers can help the slow ones improve the rendering speed. Moreover, in Chapter 5, a hybrid system is proposed. In this system, there is both an inter-frame-based distributed rendering as well as an intra-frame rendering used. For intra-frame rendering, smoke simulation particles are shared among the computers, which makes it a sort-last rendering system. In addition, in the hybrid system, a computer receives help either locally from its secondary GPU or from other computers in the network. Therefore, there are two aspects of hybridity: (i) inter-frame and intra-frame rendering, (ii) local (GPU-GPU) and network based rendering.

In addition, the thesis has investigated the bottlenecks in parallel and distributed rendering systems with simulations in Chapter 3. It is shown that in a locally distributed rendering system, the transfer from one GPU to the other needs to go over the CPU and the memory, which is a limiting factor. Moreover, for distributed rendering using a network of computers, the network speed is a bottleneck. Chapter 3 discussed that, under these bottlenecks, rendering can be distributed provided that the rendering

Table 6.1: List of tasks that a full-fledged distributed rendering system should consider.

Task Type	Task descriptions
Rendering Tasks	multitexturing, occlusion culling, particles, fog simulation, illumination, vegetation, anti-aliasing ...
Artificial Intelligence Tasks	crowd simulation, behavior simulation, traffic simulation ...
Physics and Animation Tasks	vehicle dynamics, collision detection, cloud motion, character animation ...
Simulation Management Tasks	replay, simulation logic
Preprocessing Tasks	paging

speed of a processing unit is slow enough to compensate for the time delay for the data transfer, either in the computer or in the network.

By means of our proposed efficient load balancing strategy, rendering performance of slow computers are increased with the help of the fast ones, so that the slow computers are no longer bottlenecks in heterogeneous systems.

6.1 Limitations and Future Work

The current work relies on the assumption that the packets sent by a helper travel *uniformly* in the network. In other words, they can be received and read at regular intervals by the help computer. The effects of this assumption can be relaxed by: (i) using faster yet more expensive computer networks, and (ii) by a mechanism to synchronize precisely the clocks on each computer and label time for each frame. However, this may impose unnecessary waiting times for fast processing units, decreasing the overall performance of the system.

In a full-fledged distributed rendering system, the *things* that can be distributed should not be limited to frames or particles. In general, the tasks listed in Table 6.1, i.e., rendering tasks, artificial intelligence tasks, physics and animation tasks, simulation management tasks and preprocessing tasks should all be considered. Note that these tasks are very different in nature and require designing such a system from scratch with distributed rendering in mind.

Moreover, in cases of distributed systems including many more number of computers over wide area networks, network bandwidth criteria should be considered regarding the trade-off between compression ratio and compression/decompression process durations as discussed in 4.3.1.

REFERENCES

[1] J. Ahrens and J. Painter. Efficient sort-last rendering using compression-based image compositing. In *Proceedings of the 2nd Eurographics Workshop on Parallel Graphics and Visualization*, pages 145–151, 1998.

[2] E. Bethel, G. Humphreys, B. Paul, and J. D. Brederson. Sort-first, distributed memory parallel visualization and rendering. In *Proceedings of the 2003 IEEE symposium on parallel and large-data visualization and graphics*, page 7. IEEE Computer Society, 2003.

[3] A. P. D. Binotto, C. E. Pereira, and D. W. Fellner. Towards dynamic reconfigurable load-balancing for hybrid desktop platforms. In *IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, pages 1–4. IEEE, 2010.

[4] R. S. Brar and B. Singh. A survey on different compression techniques and bit reduction algorithm for compression of text-lossless data. *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(3):579–582, 2013.

[5] D. Cederman and P. Tsigas. On dynamic load balancing on graphics processors. In *Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*, pages 57–64. Eurographics Association, 2008.

[6] Whirlwind computer. Wikipedia. `http://en.wikipedia.org/wiki/Whirlwind_(computer)`. [Online; Last access: 26-January-2014].

[7] T. W. Crockett. An introduction to parallel rendering. *Parallel Computing*, 23(7):819–843, 1997.

[8] S. Eilemann. An analysis of parallel rendering systems. `http://www.equalizergraphics.com/documents/ParallelRenderingSystems.pdf`, 2006. [Online; Last access: 26-January-2014].

[9] S. Eilemann. Project equalizer: A project description of equalizer. `http://www.equalizergraphics.com/`, 2006. [Online; Last access: 26-January-2014].

[10] S. Eilemann, M. Makhinya, and R. Pajarola. Equalizer: A scalable parallel rendering framework. *IEEE Transactions on Visualization and Computer Graphics*, 15(3):436–452, 2009.

[11] F. Erol, S. Eilemann, and R. Pajarola. Cross-segment load balancing in parallel rendering. In *Proceedings of the Eurographics Conference on Parallel Graphics and Visualization*, pages 41–50, 2011.

[12] W. Fetter. Wikipedia page. `http://en.wikipedia.org/wiki/William_Fetter).` [Online; Last access: 26-January-2014].

[13] A. Frederico, C. Waldemar, C. Renato, and C. J. Luiz. A load-balancing strategy for sort-first distributed rendering. In *Proceedings of the 17th Brazilian Symposium on Computer Graphics and Image Processing*, pages 292–299. IEEE, 2004.

[14] S. Frey and T. Ertl. Load balancing utilizing data redundancy in distributed volume rendering. In *Proceedings of the 11th Eurographics conference on Parallel Graphics and Visualization*, pages 51–60. Eurographics Association, 2011.

[15] CAE Image Generators. Company website. `http://www.cae.com`. [Online; Last access: 26-January-2014].

[16] Rockwell Collins Image Generators. Company website. `http://www.rockwellcollins.com`. [Online; Last access: 26-January-2014].

[17] U. Gun. Interactive editing of complex terrains on parallel graphics architectures. M.sc. thesis, Middle East Technical University, Department of Computer Engineering, 2009.

[18] R. Hagan and Y. Cao. Multi-gpu load balancing for in-situ visualization. In *International Conference on Parallel and Distributed Processing Techniques and Applications*, 2011.

[19] D. Hoffer. What does big data look like? visualization is key for humans. Wired, `http://www.wired.com/insights/2014/01/big-data-look-like-visualization-key-humans/).` [Online; Last access: 26-January-2014].

[20] G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett, and P. Hanrahan. Wiregl: a scalable graphics system for clusters. In *SIGGRAPH*, volume 1, pages 129–140, 2001.

[21] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 693–702. ACM, 2002.

[22] M. Joselli, M. Zamith, E. Clua, R. Leal-Toledo, A. Montenegro, L. Valente, B. Feijo, and P. Pagliosa. An architeture with automatic load balancing for real-time simulation and visualization systems. *JCIS - Journal of Computational Interdisciplinary Sciences*, 1(3):207–224, 2010.

[23] G. Yildirim Kalkan and V. Isler. A hybrid architecture for distributed rendering of huge data. *International Journal of Parallel Programming*, 2014 (in preparation).

[24] G. Yildirim Kalkan and V. Isler. Bottlenecks in distributed real-time visualization of huge data on heterogeneous systems. *Journal of Visualization*, 2014 (submitted).

[25] G. Yildirim Kalkan and V. Isler. An efficient load-balancing strategy for parallel rendering of complex scenes on distributed heterogeneous systems. *Computer Graphics International, Australia*, 2014 (submitted).

[26] H. Liu, P. Wang, S. Li, X. Cai, and L. Zeng. Research on multi-cpu and multi-gpu scalable parallel rendering on shared memory architecture. In *Proceedings of the 2012 International Conference on Computer Application and System Modeling*. Atlantis Press, 2012.

[27] Y. H. Low, S. Jain, S. J. Turner, W. Cai, W. J. Hsu, and S. Y. Huang. Load balancing for conservative simulation on shared memory multiprocessor systems. In *Proceedings Fourteenth Workshop on Parallel and Distributed Simulation*, pages 139–146, 2000.

[28] S. Marchesin, C. Mongenet, and J.-M. Dischler. Dynamic load balancing for parallel volume rendering. In *Proceedings of the 6th Eurographics Conference on Parallel Graphics and Visualization*, pages 43–50. Eurographics Association, 2006.

[29] S. Marchesin, C. Mongenet, and J.-M. Dischler. Multi-gpu sort-last volume visualization. In *Proceedings of the 8th Eurographics Conference on Parallel Graphics and Visualization*, EG PGV'08, pages 1–8, 2008.

[30] D. Mateika and R. Martavičius. Analysis of the compression ratio and quality in medical images. *Information Technology and Control*, 35(4):530—-536, 2006.

[31] T. Meek. Big-data visualization is worth a thousand spreadsheets. Forbes, `http://www.forbes.com/sites/netapp/2013/12/11/big-data-visualization/)`. [Online; Last access: 26-January-2014].

[32] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs. A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications*, 14(4):23–32, 1994.

[33] First Computer Animated Movie. Youtube. `http://www.youtube.com/watch?v=ULoethEU2OY)`. [Online; Last access: 26-January-2014].

[34] OpenGL Multipipe™ SDK White Paper. Document number: 007-4516-003. Sgi Techpubs Library.

[35] P. Peschlow, T. Honecker, and P. Martini. A flexible dynamic partitioning algorithm for optimistic distributed simulation. In *Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation*, pages 219–228. IEEE Computer Society, 2007.

[36] J. Ragan-Kelley. Keeping many cores busy: Scheduling the graphics pipeline. SIGGRAPH Course: Beyond Programmable Shading.

[37] J. Rodrigues, A. R. Balan, L. Zaina, and A. J. M. Traina. A survey on distributed visualization techniques over clusters of personal computers. *INFOCOMP Journal of Computer Science*, 8(4):79—-90, 2009.

[38] S. Russell. The first computer game. `http://en.wikipedia.org/wiki/Spacewar_(video_game))`. [Online; Last access: 26-January-2014].

[39] R. Samanta, T. Funkhouser, K. Li, and J. P. Singh. Hybrid sort-first and sort-last parallel rendering with a cluster of pcs. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 97–108. ACM, 2000.

[40] R. Samanta, J. Zheng, T. Funkhouser, K. Li, and J. P. Singh. Load balancing for multi-projector rendering systems. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 107–116. ACM, 1999.

[41] K. Sayood. *Introduction to data compression*. Newnes, 2012.

[42] R. Schlagenhaft, M. Ruhwandl, C. Sporrer, and H. Bauer. Dynamic load balancing of a multi-cluster simulator on a network of workstations. In *ACM SIGSIM Simulation Digest*, volume 25, pages 175–180. IEEE Computer Society, 1995.

[43] R. Sim. Survey of fast compression algorithms. `http://www.technofumbles.com/weblog/2011/04/22/survey-of-fast-compression-algorithms-part-1-2/`. [Online; Last access: 26-January-2014].

[44] Evans Sutherland Visual Systems. Company website. `http://www.es.com`. [Online; Last access: 26-January-2014].

[45] D. Turek. The case against digital sprawl. Businessweek, `http://www.businessweek.com/articles/2012-05-02/the-case-against-digital-sprawl)`. [Online; Last access: 26-January-2014].

[46] S. Tzeng, A. Patney, and J. D. Owens. Task management for irregular-parallel workloads on the gpu. In *Proceedings of the Conference on High Performance Graphics*, pages 29–37. Eurographics Association, 2010.

[47] H. T.-H. Wong. *Architectures and limits of GPU-CPU heterogeneous systems.* M.sc. thesis, University of British Columbia, 2008.

[48] B. Wylie, C. Pavlakos, V. Lewis, and K. Moreland. Scalable rendering on pc clusters. *Computer Graphics and Applications, IEEE*, 21(4):62–69, 2001.

[49] P. Yin, X. Jiang, J. Shi, and R. Zhou. Multi-screen tiled displayed, parallel rendering system for a large terrain dataset. *International Journal of Virtual Reality*, 5(4):47–54, 2006.

[50] M. P. M. Zamith, E. W. G. Clua, A. Conci, A. Montenegro, R. C. P. Leal-Toledo, P. A. Pagliosa, L. Valente, and B. Feij. A game loop architecture for the gpu used as a math coprocessor in real-time applications. *Computers in Entertainment (CIE)*, 6(3):42, 2008.

[51] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

[52] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530—-536, 1978.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Yıldırım Kalkan, Gökçe

Nationality: Turkish

Date and Place of Birth: April, 1979, Trabzon

Marital Status: Married

Phone: 0 312 210 00 46

Fax: 0 312 210 00 47

EDUCATION

Degree	**Institution**	**Year of Graduation**
M.Sc.	Dept. of Computer Engineering, METU	2005
B.S.	Dept. of Computer Engineering, METU	2002
High School	Trabzon Anadolu Lisesi	1997

PROFESSIONAL EXPERIENCE

Year	**Place**	**Enrollment**
8 years	Simsoft	Director & Co-Founder
3,5 years	Meteksan	Software Engineer

PUBLICATIONS

Journal Publications and Book Chapters

- Gökçe Yıldırım Kalkan, Veysi İşler, "A hybrid architecture for distributed rendering of huge data", International Journal of Parallel Programming, 2014 (in preparation).
- Gökçe Yıldırım Kalkan, Veysi İşler, "Bottlenecks in distributed real-time visualization of huge data on heterogeneous systems", Journal of Visualization, 2014 (submitted).
- Gökçe Yıldırım, Hacer Yalım Keleş, Veysi İşler, "Three Dimensional Smoke Simulation on Programmable Graphics Hardware", International Conference on Parallel Computational Fluid Dynamics, 21-24, May 2007, Turkey (Also published as the Book Chapter in Parallel Computational Fluid Dynamics 2007, Lecture Notes in Computational Science and Engineering, Volume 67, 2009).

Conference Publications

- Gökçe Yıldırım Kalkan, Veysi İşler, "An Efficient Load-Balancing Strategy for Parallel Rendering of Complex Scenes on Distributed Heterogeneous Systems", Computer Graphics International, Australia, 2014 (submitted).
- Mehmet Sadık Akyol, Gökçe Yıldırım, "Gerçek Zamanlı Dinamik Sensör Simülasyonu", Ulusal Savunma Uygulamaları Modelleme ve Simülasyon Konferansı, 2013.
- Gökçe Yıldırım, Veysi İşler, "Efficient Rendering Of Complex Scenes On Heterogeneous Parallel Architectures", Eurasiagraphics 2012, International Workshop on Computer Graphics, Animation and Game Technologies,4-5 May 2012, Işık University, İstanbul.
- Gökçe Yıldırım, İsmail Bıkmaz, Kürşat Çağıltay, Veysi İşler, "Savunma Sistemleri Tasarımında İnsan Bilgisayar Etkileşimi Çalışmalarının Rolü", Ulusal Savunma Uygulamaları Modelleme ve Simülasyon Konferansı, 1, (2007), s.200-208.

- İsmail Bıkmaz, Gökçe Yıldırım, Veysi İşler, Kürşat Çağıltay, Kayhan İmre, “Ciddi Oyun ve Simülasyon Altyapısı”, Ulusal Savunma Uygulamaları Modelleme ve Simülasyon Konferansı, 1, (2007), s.457-466.