An Architecture Design for a Real-Time Web-Based Visualization in the Grid Environment

Bhargavi Sura

Follow this and additional works at: https://scholarsjunction.msstate.edu/td

Recommended Citation

https://scholarsjunction.msstate.edu/td/481
AN ARCHITECTURE DESIGN FOR A REAL-TIME WEB-BASED VISUALIZATION 
IN THE GRID ENVIRONMENT

By

Bhargavi Sura

Approved:

Tomasz A. Haupt  
Research Associate Professor of  
Engineering Research Center  
(Major Advisor)

Edward A. Luke  
Assistant Professor of Computer Science  
and Engineering  
(Committee Member)

Edward B. Allen  
Assistant Professor and  
Graduate Coordinator,  
Department of Computer Science  
and Engineering  
(Committee Member)

Robert P. Taylor  
Interim Dean of the College  
of Engineering
Name: Bhargavi Sura

Date of Degree: December 11, 2004

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Dr. Tomasz Haupt

Title of Study: AN ARCHITECTURE DESIGN FOR A REAL-TIME WEB-BASED VISUALIZATION IN THE GRID ENVIRONMENT

Pages in Study: 102

Candidate for Degree of Master of Science

Situations like war, terrorist attacks, fire accidents, floods, storms, etc., which threaten human life and property, demand immediate action to decrease the damage caused by them. A system is needed that predicts the future events based on what has happened and notifies the concerned personnel. The situation could be better understood in less time if the data is represented as colored, shaded and moving images rather than as numbers. Such a system requires a real-time Web-based visualization system with easy and secure access to grid resources, presenting easy-to-read graphics through a simple interface provided by a Web browser, and responding to user actions immediately. The Web and grid environments impose severe performance constraints such as communication time, latency of the network, etc., making it highly difficult to have a highly responsive real-time visualization. This work aims in finding an appropriate design that satisfies the above requirements. It also aims in understanding the limitations of a distributed environment
for real-time applications and finding ways to overcome those limitations. A three-tier architecture is proposed, implemented, and tested to find the bottlenecks of the distributed environment. Relevant design principles are applied to a case study eliminating or minimizing the bottlenecks until the case study system satisfies all the requirements. The case study is the Fire-Smoke system, simulating the propagation of fire in the ex-USS Shadwell test area emulating a submarine. This system is re-implemented from a stand-alone system in OpenGL to a real-time Web-based visualization system using Java3D and J2EE technologies.
DEDICATION

To my mom.
ACKNOWLEDGMENTS

First and foremost, I would like to thank my major professor and thesis advisor, Dr. Tomasz Haupt for sharing his ideas with me and helping me in all ways. I gratefully acknowledge him for his enthusiastic supervision and for reading my numerous draft manuscripts. Next, I thank my committee members Dr. Edward Allen and Dr. Edward Luke for their suggestions and encouragement.

My special thanks to Anand Kalyanasundaram and Shravan Durvasula for helping me with this work. I would also like to thank Dmitry Shulga whose work is used in developing the test-case system and the ECS team who developed the ECS grid portal that is used in this work.

This research was funded by the Naval Research Laboratories and I gratefully acknowledge their financial support. I also thank Engineering Research Center and Center for Advanced Vehicular Systems for providing the infrastructure to conduct the experiments.

This thesis also owes its existence to my family, specially my mom who constantly encouraged and supported me in doing my Masters. I also thank my friend Archana Chilukuri for her support and tolerance, which meant that I always had someone to talk with. Finally, I would also like to thank Neli Fairfield for reviewing the English grammar of this thesis draft.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEDICATION</td>
<td>ii</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>vii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>viii</td>
</tr>
<tr>
<td>LIST OF ABBREVIATIONS</td>
<td>x</td>
</tr>
<tr>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>I. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Visualizations for Real-Time Systems</td>
<td>1</td>
</tr>
<tr>
<td>1.1.1 Limitations of Stand-alone Systems</td>
<td>3</td>
</tr>
<tr>
<td>1.1.2 Visualizations for Distributed Real-Time Systems</td>
<td>4</td>
</tr>
<tr>
<td>1.2 Visualizations for Grid-Based Real-Time Systems</td>
<td>5</td>
</tr>
<tr>
<td>1.3 Hypothesis</td>
<td>6</td>
</tr>
<tr>
<td>1.4 Research Questions</td>
<td>7</td>
</tr>
<tr>
<td>1.5 Contribution</td>
<td>7</td>
</tr>
<tr>
<td>1.6 Thesis Structure</td>
<td>8</td>
</tr>
<tr>
<td>II. LITERATURE REVIEW</td>
<td>9</td>
</tr>
<tr>
<td>2.1 Distributed Visualization</td>
<td>9</td>
</tr>
<tr>
<td>2.1.1 Types of Distributed Visualization</td>
<td>12</td>
</tr>
<tr>
<td>2.1.2 Client-based Visualization</td>
<td>14</td>
</tr>
<tr>
<td>2.1.2.1 FAST</td>
<td>15</td>
</tr>
<tr>
<td>2.1.2.2 Vis5D</td>
<td>16</td>
</tr>
<tr>
<td>2.1.2.3 VizWiz</td>
<td>17</td>
</tr>
<tr>
<td>2.1.2.4 MolVis</td>
<td>17</td>
</tr>
<tr>
<td>2.1.2.5 Visualization at RIKZ</td>
<td>18</td>
</tr>
<tr>
<td>2.1.2.6 Particle Tracing System</td>
<td>20</td>
</tr>
</tbody>
</table>
CHAPTER 2.1.3 Server-based Visualization .................................. 20
2.1.3.1 Vis-a-Web ................................................. 20
2.1.3.2 Remote Visualization using the World Wide Web ... 21
2.1.3.3 COVISE ...................................................... 23
2.1.3.4 NOVICE .................................................. 23
2.1.4 Real-Time Distributed Simulations .............................. 24
2.2 Technologies Supporting Distributed Visualization .............. 25
2.2.1 J2EE, Java3D, and Web portals ................................ 26
2.2.2 Grid Computing ............................................. 27
2.3 Other Related Work .................................................. 30
2.3.1 Fire Dynamics Simulator ....................................... 30
2.3.2 Prototype of Distributed Fire-Smoke Model ................. 32
2.4 Summary .............................................................. 32

III. VISGREN REQUIREMENTS AND ARCHITECTURE DESIGN ........ 34
3.1 ViSGrEn Requirements .............................................. 34
3.1.1 User Requirements ............................................ 34
3.1.1.1 For Web-browser based ................................ 35
3.1.1.2 For Collaboration ....................................... 36
3.1.1.3 For High Interactivity ................................... 36
3.1.2 System Requirements .......................................... 38
3.1.2.1 Scalable .................................................. 38
3.1.2.2 Fault Tolerant and Reliable ............................ 38
3.1.2.3 Hardware Requirements ................................. 38
3.2 ViSGrEn Architecture Design ...................................... 39

IV. IMPLEMENTATION, EVALUATION, AND OPTIMIZATIONS ............ 42
4.1 Testbed for ViSGrEn ................................................ 42
4.2 ViSGrEn Features .................................................... 43
4.2.1 Creating Simulations ......................................... 43
4.2.2 Plugging-in Different Geometries ............................ 44
4.2.3 Replaying Simulations ....................................... 44
4.2.4 Fault-tolerance for Lost Data Packets ....................... 45
4.3 Implementation of the Baseline System ............................ 47
4.4 Evaluation of Baseline System and Optimizations ............... 50
4.4.1 Startup Time .................................................. 51
4.4.1.1 Getting Data from the Database ......................... 51
<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4.1.2 Optimization: Preprocessing and Caching the Database</td>
<td>53</td>
</tr>
<tr>
<td>4.4.1.3 Rendering Time</td>
<td>56</td>
</tr>
<tr>
<td>4.4.1.4 Optimization: Progressive Rendering</td>
<td>56</td>
</tr>
<tr>
<td>4.4.2 Interactivity and Response Time</td>
<td>59</td>
</tr>
<tr>
<td>4.4.2.1 Optimization: Client-based Rendering</td>
<td>59</td>
</tr>
<tr>
<td>4.4.3 Simulation Visualization Time</td>
<td>59</td>
</tr>
<tr>
<td>4.4.3.1 Optimizations: Reducing Simulation Visualization Time</td>
<td>62</td>
</tr>
<tr>
<td>4.4.4 Summary of Evaluation</td>
<td>66</td>
</tr>
<tr>
<td>V. EXPERIMENTAL RESULTS</td>
<td>68</td>
</tr>
<tr>
<td>5.1 Issues Encountered</td>
<td>71</td>
</tr>
<tr>
<td>5.1.1 Java3D with Swing Bug</td>
<td>71</td>
</tr>
<tr>
<td>5.1.2 Timers</td>
<td>71</td>
</tr>
<tr>
<td>5.2 Performance of ViSGrEn</td>
<td>72</td>
</tr>
<tr>
<td>5.2.1 Startup Time</td>
<td>72</td>
</tr>
<tr>
<td>5.2.1.1 Getting Data from the Database</td>
<td>72</td>
</tr>
<tr>
<td>5.2.1.2 Rendering Time</td>
<td>73</td>
</tr>
<tr>
<td>5.2.1.3 Total Startup Time</td>
<td>77</td>
</tr>
<tr>
<td>5.2.2 Interactivity and Response Time</td>
<td>79</td>
</tr>
<tr>
<td>5.2.2.1 Rotations and Translations</td>
<td>79</td>
</tr>
<tr>
<td>5.2.2.2 Changing Object States</td>
<td>80</td>
</tr>
<tr>
<td>5.2.3 Simulation Visualization Time</td>
<td>83</td>
</tr>
<tr>
<td>5.2.3.1 Creating Simulations</td>
<td>83</td>
</tr>
<tr>
<td>5.2.3.2 Replaying Pre-run Simulations</td>
<td>87</td>
</tr>
<tr>
<td>5.3 Validation of Hypothesis</td>
<td>92</td>
</tr>
<tr>
<td>5.4 Summary of Results</td>
<td>93</td>
</tr>
<tr>
<td>VI. CONCLUSIONS AND FUTURE WORK</td>
<td>95</td>
</tr>
<tr>
<td>6.1 Conclusions</td>
<td>95</td>
</tr>
<tr>
<td>6.2 Future Work</td>
<td>97</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>100</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Startup Time for Peterson1</td>
</tr>
<tr>
<td>5.2</td>
<td>Interactivity and Response Time for Peterson1 with Suppression System</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Fire-Smoke Stand-Alone System Showing Simulation and GUI</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Fire-Smoke System Features</td>
<td>3</td>
</tr>
<tr>
<td>2.1 Visualization Pipeline</td>
<td>10</td>
</tr>
<tr>
<td>2.2 Network-Oriented Visualization</td>
<td>11</td>
</tr>
<tr>
<td>2.3 Client Based Visualization</td>
<td>13</td>
</tr>
<tr>
<td>2.4 Server Based Visualization</td>
<td>14</td>
</tr>
<tr>
<td>2.5 Variation of Server Based Visualization</td>
<td>15</td>
</tr>
<tr>
<td>2.6 System Architecture of MolVis</td>
<td>18</td>
</tr>
<tr>
<td>2.7 Vis-a-web System</td>
<td>22</td>
</tr>
<tr>
<td>2.8 Three-tier Architecture of Web Portal</td>
<td>28</td>
</tr>
<tr>
<td>2.9 Architecture of Visual Exploration System</td>
<td>29</td>
</tr>
<tr>
<td>2.10 Snapshot of FDS</td>
<td>31</td>
</tr>
<tr>
<td>3.1 System Architecture for ViSGrEn</td>
<td>41</td>
</tr>
<tr>
<td>4.1 Facility to Plug-in Different Geometries</td>
<td>44</td>
</tr>
<tr>
<td>4.2 List of Pre-run Simulations Available for Replay</td>
<td>46</td>
</tr>
<tr>
<td>4.3 Sampling in Time or Space</td>
<td>46</td>
</tr>
<tr>
<td>4.4 Java3D Scene Graph Structure for Fire-Smoke System</td>
<td>49</td>
</tr>
<tr>
<td>FIGURE</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>4.5</td>
<td>Time for Getting Data from the Database</td>
</tr>
<tr>
<td>4.6</td>
<td>Preprocessing and Caching the Database</td>
</tr>
<tr>
<td>4.7</td>
<td>Time for Rendering the Geometry</td>
</tr>
<tr>
<td>4.8</td>
<td>Different Cases for $T_{exe}$, $T_{com}$, and $T_{viz}$ for a Three Step Simulation</td>
</tr>
<tr>
<td>4.9</td>
<td>Visualization Time for Simulation Pre-runs</td>
</tr>
<tr>
<td>5.1</td>
<td>ViSGrEn Running as an Applet in a Browser</td>
</tr>
<tr>
<td>5.2</td>
<td>Scene Zoomed and Translated</td>
</tr>
<tr>
<td>5.3</td>
<td>Graphical User Interface for setting simulation parameters</td>
</tr>
<tr>
<td>5.4</td>
<td>Optimized Time for Getting Data from the Database</td>
</tr>
<tr>
<td>5.5</td>
<td>Optimized Time for Rendering the Geometry</td>
</tr>
<tr>
<td>5.6</td>
<td>Performance Gain Due to Progressive Rendering Technique</td>
</tr>
<tr>
<td>5.7</td>
<td>Startup Time (DB reading + Rendering)</td>
</tr>
<tr>
<td>5.8</td>
<td>Time for One Rotation or Translation</td>
</tr>
<tr>
<td>5.9</td>
<td>Time for Changing the State of One Object (Door)</td>
</tr>
<tr>
<td>5.10</td>
<td>Time for Running and Visualizing a Complex Simulation</td>
</tr>
<tr>
<td>5.11</td>
<td>Performance of Simulation Executable on Different Operating Systems</td>
</tr>
<tr>
<td>5.12</td>
<td>Optimized Visualization Time for Simulation Pre-runs</td>
</tr>
<tr>
<td>5.13</td>
<td>Effect of Skip Rate on Simulations of Varying Complexities</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Full Form</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
</tr>
<tr>
<td>API</td>
<td>Application Program Interface</td>
</tr>
<tr>
<td>CAD</td>
<td>Computer Aided Design</td>
</tr>
<tr>
<td>CoG</td>
<td>Commodity Grid Kit</td>
</tr>
<tr>
<td>COVISE</td>
<td>Collaborative Visualization and Simulation Environment</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>ECS</td>
<td>Enterprise Computational Services</td>
</tr>
<tr>
<td>EJB</td>
<td>Enterprise JavaBeans</td>
</tr>
<tr>
<td>ERC</td>
<td>Engineering Research Center</td>
</tr>
<tr>
<td>FAST</td>
<td>Flow Analysis Software Toolkit</td>
</tr>
<tr>
<td>FDS</td>
<td>Fire Dynamics Simulator</td>
</tr>
<tr>
<td>GASS</td>
<td>Global Access to Secondary Storage</td>
</tr>
<tr>
<td>GRAM</td>
<td>Grid Resource Allocation and Management</td>
</tr>
<tr>
<td>GSI</td>
<td>Grid Security Infrastructure</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HPC</td>
<td>High performance Computer</td>
</tr>
<tr>
<td>HTML</td>
<td>Hyper Text markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hyper Text Transfer Protocol</td>
</tr>
<tr>
<td>J2EE</td>
<td>Java 2 Platform Enterprise Edition</td>
</tr>
<tr>
<td>JDBC</td>
<td>Java DataBase Connectivity</td>
</tr>
<tr>
<td>JSP</td>
<td>Java Server Pages</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>NOVICE</td>
<td>Network-Oriented Visualization in the Clinical Environment</td>
</tr>
</tbody>
</table>
NSDS  NEESgrid Streaming Data Services
OGSA  Open Grid Services Architecture
RAM   Random Access Memory
ViSGrEn  Visualization System for the Grid Environment
VRML  Virtual Reality Modeling Language
WWW   World Wide Web
CHAPTER I

INTRODUCTION

1.1 Visualizations for Real-Time Systems

Situations like war, terrorist attacks, fire accidents, floods, and storms that threaten human life and property, demand immediate action to decrease the damage they cause. Non-emergency situations like stock-market trading or games also require immediate action. A system is needed that notifies the concerned personnel of these events. The situation could be better understood in less time if the data is represented as colored and moving images rather than as numbers, which requires visualizing the data in real-time or even faster than real-time. The system must be highly responsive to user actions. Though photo-realistic graphics are not necessary, the information should be presented in such a way that the situation can be easily understood in a short time without using elaborate menus or mouse clicks. Visualization should aid the concerned personnel (rescue teams), who operate in extreme conditions with threat to human life and property, in taking immediate action.

“Simulation System for Propagation of Fire and Smoke” developed by Dmitry Shulga using OpenGL and C++ is an example of such a system [23]. This system is a stand-alone system simulating fire and smoke propagation in the ex-USS Shadwell test area emulating a submarine. The main goal was to generate predictions faster than real-time fire-related
events happening onboard the ship, thus giving major emphasis to the performance of the system [23]. This system is highly interactive, allowing modifications to the model setting using a simple-to-operate Graphical User Interface (GUI). The states of different objects like doors, ventilation system, fans can be set. The GUI and simulation visualizations are shown in Figure 1.1. The system has the facility to replay previously run simulations and to compare two simulations at the same time (left figure in Figure 1.2). The user can zoom, move, and rotate the scene (right figure in Figure 1.2).
To make this application run faster, data is read from text files and stored in the main memory using a data bridge. Once the data is buffered into the main memory, the time to access the data is much less than accessing the database or text files. Also, multithreading is used to run simulation and visualization in parallel, efficiently utilizing the processing time of the CPU (Central Processing Unit). Thus, this application could achieve real-time visualization of the simulations.

1.1.1 Limitations of Stand-alone Systems

Stand-alone systems have their own limitations. For the Fire-Smoke system mentioned above, even though multithreading is used, there is still only a single CPU that executes both the visualization and simulation processes. For complex simulations requiring more processing time, this constraint becomes severe, making real-time simulation visualization almost impossible. The CPU constraint can be overcome by separating simulation and
visualization tasks and running simulation jobs on a powerful computer and visualizing the results on another computer. Thus, access to High-Performance Computers (HPCs) is needed. Also, most times, there is more than one person who needs to know about the situation or who takes action, and those persons might be at a different location than the actual scene. For example, in case of fire accidents, the response team that controls the fire suppression system might be operating from a remote location. Thus, the visualization application must be available to any number of geographically distributed users at any time.

1.1.2 Visualizations for Distributed Real-Time Systems

By making the system accessible on the Web and providing access to HPCs located on grids, the above-mentioned problems of CPU usage and remote access can be solved. For a Web-based system, there are some additional requirements to our previous ones, such as portability across different platforms and secure access to only authorized personnel. It is clear from all the above requirements that a real-time Web-based visualization system that is portable across different platforms, accessible through a simple interface, has secure access to grid resources and provides highly interactive, easy to understand, and intuitive graphics is needed. The distributed environment imposes severe performance constraints, such as latency of the network and communication delays, making highly interactive real-time Web-based visualizations very difficult. The goal of this research is to find an appropriate architecture design to satisfy the above requirements. This work also
1.2 Visualizations for Grid-Based Real-Time Systems

There are many grid-based applications that provide real-time visualization over the Web, but most of them are not very interactive. One example of such an application is a weather forecast system where 2D images are generated on the server side and displayed to remote users in real-time. The problem with this type of applications is that the 2D images provide no interactivity. Some applications provide high interactivity but are not real-time visualizations which typically involve CFD (Computational Fluid Dynamics) computations that require hours of computing time even on HPCs. Though some applications provide real-time visualization and high interactivity, they either do not have facility to run simulations on the grid or are stand-alone applications. There is no standard architecture design that satisfies all the requirements: Web-based real-time visualization, high responsiveness, accurate and easy-to-read graphics, portability across platforms, and grid access. Thus there is a need for such a system that

- Predicts the outcome of events and provides visualization in real-time
- Is available to geographically distributed users through a simple interface
- Can provide easy and secure access to HPCs located on a grid
- Is highly responsive to user actions
- Is portable across platforms
Developing such a system in a distributed environment is a difficult task, as the distributed environment imposes severe performance constraints such as communication time, and network latency. Thus, identifying the bottlenecks that affect the performance, finding ways to overcome them, and improving the design to satisfy the requirements are required.

1.3 Hypothesis

The hypothesis of this research is as follows.

1. A grid-based simulation system can provide through a Web-browser real-time visual information that is helpful for emergency response teams.

2. An architecture is feasible that supports easy modifications of geometry, replaying previous simulations, playing with different what-if scenarios, running and visualizing multiple simulations concurrently, and running different simulation models on different platforms.

The goal of this research is to develop such a system by finding an appropriate design that minimizes the limitations imposed by the Web and the grid environments and satisfies the requirements. Numerical simulations and predictions are run on the high-performance computing resources of a grid to generate results in real-time or faster than real-time. These results are displayed to the clients, who operate in extreme conditions, in an accurate and easy-to-read way. The goal is to convey the information in a short time. This visualization must be highly interactive and should respond to user actions immediately. For design and training, more accurate simulation models such as CFD (Computational Fluid Dynamics) can be used.
1.4 Research Questions

Major challenges of this research are minimizing the latency and network delays in communicating from the server to the client and vice versa, having high interactivity with the visualization at the client side in spite of having the data at the server side, providing easy and secure access to the computing resources on the grid, and overcoming the limitations, if any, imposed by Java3D, which is the language selected for the case study.

Finding answers to the following research questions will provide evidence for or against the hypothesis.

1. What are the constraints imposed by the Web and the grid environments that seriously affect the performance of the system?
2. What techniques can overcome the latency in visualizing the geometry at the start of the application and how effective is each technique?
3. What techniques allow high interactivity with the visualization at the client side?
4. What techniques can decrease the communication time between the server and the clients, and between the server and the grid resources?
5. What is the performance degradation in making the application portable across platforms? (i.e., due to Java and Java3D)
6. Should the quality of visualization be compromised in order to satisfy the requirements?

1.5 Contribution

The hypothesis, if confirmed, would provide an architecture design for developing real-time Web-based simulation visualization systems in the grid environment. This research
also contributes to understanding of the limitations imposed by the distributed environment and identifies ways to overcome them.

A system developed with this architecture would provide real-time visualizations for a remote user to take immediate action in emergency situations. This system would also support a simulation repository where the users can post their simulation results and get them analyzed by geographically distributed experts. It also acts as a remote design and training tool for designers and rescue teams.

1.6 Thesis Structure

This thesis is organized into six chapters. The introduction provides an overview of the research and motivation behind it. The second chapter surveys distributed visualization methods and provides an overview of different technologies needed. The third chapter describes the Visualization System for the Grid Environment (ViSGrEn) requirements and an architecture design that addresses the problems identified in the introduction. The fourth chapter presents implementation details, baseline system evaluation, and various optimizations to increase the performance of the ViSGrEn. The fifth chapter presents an evaluation of the proposed optimizations based on the experimental results and makes a decision about acceptance or rejection of the hypothesis. Finally, the sixth chapter summarizes the research and suggests opportunities for further research.
CHAPTER II

LITERATURE REVIEW

In this section, distributed visualization and its types are explained and some examples are presented first. Next, some technologies that are required for supporting distributed visualization are introduced. Before going into distributed visualization, let us see how visualization is done.

Visualization starts with a model simulating a process and generating data. This data is preprocessed to enhance certain features or to remove noise. This prepared data is mapped to geometry, which is then rendered as an image. Rendering is the final step of the visualization process. These steps are shown in Figure 2.1 [13].

2.1 Distributed Visualization

If some steps of the visualization pipeline are executed on one computer, and the remaining on another computer, then it is called a distributed visualization. Distributed visualization is using one or more computers to give one or more users an image of a data set [14]. Network-oriented visualization enables the user to create a visualization from data that resides on a remote server. The data and a part of the visualization pipeline are at one computer, called the server, and the image and the rest of the visualization pipeline are
at another computer, called the client. The two are connected by a network or by a number of interconnected networks, as shown in Figure 2.2 [13]. If this network is the World Wide Web, then it is called Web-based visualization. Data is sent from visualization server computer to the client computer over a global network. Network bandwidth and connection speed should be high in order to have less latency.

Distributed visualization can be used in developing a training tool where the clients can view simulations and analyze them to learn about the system and its condition under various circumstances. By having simulations done at the server side, the clients need not have any knowledge about how to run the simulations or about actual data. Also, the clients do not require computing power, software, hardware, and storage space to run and store the simulations encouraging more users to view and interpret the simulation results.
Dissemination of data is also possible through distributed visualization. The server acts as a central repository of data, allowing the clients having web access to download the data, view, and analyze the results. By delegating the simulations to high-performance computers, simulations on demand are possible. The users can submit the simulation jobs or input parameters tailored according to their needs to the server using computational grids which allow access of geographically distributed resources like supercomputers as a single resource for data-intensive computing applications. The simulation jobs are executed on high-performance computers, and the data is visualized at the client side. Distributed visualization also supports collaborative visualization, in which multiple users can work together using multiple perspectives on information [13]. This allows its users, geographically distributed, not only to access remote resources, but also to share images and cooperate across a network [19].
2.1.1 Types of Distributed Visualization

Distributed visualization varies according to whether the server or the client has the responsibility for creating visualization. It is mainly divided into two types: client-based and server-based.

Client-based visualization is also called a thick-client or fat-client solution. Here the data is downloaded from the server to the client, and the visualization and rendering are done at the client side, as shown in Figure 2.3 [27]. The client should have the visualization software, knowledge of its use, and the processing power required for visualization. The bandwidth required is large, as whole data needs to be transferred, making it unfeasible for large or sensitive data that cannot be sent to the clients. The advantage is that the client has high interactivity with the visualization, as there is no client-server communication involved once data is transferred. Technologies like plug-ins and Java applets that are downloaded and executed on the client can also be used to implement thick-client solutions.

There are some variations to this approach:

- Visualization software is present on the client. Data is downloaded from the server. The visualization system is a helper application or a plug-in fired by the browser.
- The entire visualization software is downloaded from the server. The visualization executable is provided by Java applets embedded in an HTML page.

Some examples for client-based visualization are given in section 2.1.2.

Server-based visualization, also called thin-client solution, reduces the requirements on the client in terms of processing power, software availability and training needed that are
instead required at the server side. Nearly all functionality is delivered by the server-side visualization engine, and the client performs only the display function. Here bandwidth requirement is less, and the data is not exposed to the client. The mapping of numerical data into geometry takes place at the server side. If several clients connect at once, the processing power required on the server can be quite high. The clients will have only little interaction with the visualization. If the clients want to interactively slice through the 3D volume of data, the visualization engine at the server side generates a new VRML file or image for every newly selected data slice, thus making real-time visual data manipulation very inefficient [17].

There are two variations according to [7, 27]:

- The server sends the client an image: The entire visualization is done at the server and an image is transferred to the client (Figure 2.4[27]).
The server sends the client a 3D model: The client-side viewer does the rendering process and displays a VRML 3D world in which the user can navigate. VRML world gives a little more interactivity when compared to sending an image. The client can have some control over the visualization by submitting visualization parameters to the server (Figure 2.5[27]).

Examples for server-based visualization are presented in section 2.1.3.

2.1.2 Client-based Visualization

The following is an evaluation of the existing applications using a client-based approach that are candidates for use as a case-study system.
Flow Analysis Software Toolkit (FAST), developed by the NASA Ames Research Center, is a software package for the visualization of three-dimensional data, particularly Computational Fluid Dynamics data that provides a variety of methods for examining data [5]. The FAST software is written in C and C++. The user downloads the visualization software (FAST) and then downloads data and scripts for driving the visualization. The advantage of this approach over Java applets is that the visualization program (of 10MB size when compressed) need not be downloaded each time it is used. Instead of sending scientific data over the network as pixel image files to a movie player, the raw data, sent as it is, can be analyzed by FAST. Once the data and software are downloaded to the client, there is no network traffic involved, reducing the network load and latency in user inter-
actions with the visualization. Users should have the facility to take guided expeditions through data or conduct their own expeditions independently which requires high interaction with the visualization and data. Thus the client-based approach was chosen. The main disadvantage is that FAST Expeditions run only on Silicon Graphics workstations, making it platform-dependent and unsuitable for ViSGrEn.

2.1.2.2 Vis5D

Vis5D is a system for interactive visualization of large 5-D gridded data sets, such as those produced by numerical weather models [15]. It is generally used to look at the output of models of the earth’s atmosphere and oceans in .v5d format, but it can also be applied to any 3-D data sampled on a regular grid. Vis5D is installed on the user workstation, and the web browser invokes it as an external viewer. When a link to a Vis5D file is clicked, the browser will transfer the file and possibly uncompress it. A plug-in in the browser will recognize the data file mime type and invoke Vis5D to view that file. Vis5D has many user controls to zoom, cut-away some part of the view, change the colors of graphics, etc. The disadvantage is that Vis5D is suited only for displaying data. It cannot run prediction models in real-time or report the user actions back to the server, making it unsuitable for ViSGrEn requirements. From this system it is learnt that for high interaction at the client side, it is better to send the whole data, even though it is very large, and visualize it on the client side.
2.1.2.3 VizWiz

VizWiz, developed jointly at the San Diego Supercomputer Center and UC San Diego, is a Java applet for interactive scientific visualization on the Web [20]. It supports the visualization of volumetric and elevation datasets. The main objectives were to make it platform-independent and easy-to-use. VizWiz is downloaded as a Java applet from the server. The applet then downloads data from the server and performs a local visualization. 3D graphics are implemented using Java AWT API rather than Java3D API so that the user does not require any supporting software besides a Java-enabled browser. The user can change parameters affecting the display like resolution, scaling and loading files using a control panel. Rendering in Java API imposes severe performance penalties as it cannot make use of the graphics accelerators and other hardware. This application does not have the facility to run simulations in real-time or report user actions back to the simulation program. Therefore, this system does not fit the requirements of ViSGrEn, though the approach of presenting visualization through Java applets can be used.

2.1.2.4 MolVis

Molecular visualization system is a pure client-side application that makes use of the computational power of modern desktop workstations on the client side [6]. It allows a user to download the data from the server via the Web and to visualize it interactively. The system uses protein data bank (PDB) format for storing and transferring chemical data. Its system architecture is shown in Figure 2.6 [6]. It is developed using Java and
VRML. Java classes which are downloaded by the client form the clients’ visualization system and are stored on the server. The data is rendered into a VRML scene using Java classes and displayed to the client in a VRML viewer. This system is useful for visualizing only molecular data in PDB format and gives the user a very limited interaction with the visualization. Therefore, this application does not satisfy the ViSGreEn’s requirements.

Figure 2.6 System Architecture of MolVis

2.1.2.5 Visualization at RIKZ

The National Institute for Coastal and Marine Management, called RIKZ [13] that is located in the Netherlands, supplies its clients with advice and data on the sustainable
use of estuaries, coasts and seas. Simona, an information system for computing physical phenomena in water, is used as a framework for hydrodynamic simulations. Data is collected from these simulations in the form of a Simona Data Storage (SDS) file. The SDS files, which are platform- and language-dependent, are often quite large, ranging from megabytes to gigabytes. These SDS files are converted on the server side to Matlab files, which are ASCII-based and platform-independent using a Java interface.

This system has a file server and a web server running on the server side and a web browser that can display Java applets on the client side. The user connects to the server using a web browser and chooses an SDS file. Then, the web browser downloads and launches a Java applet with an interface to convert SDS files to Matlab files. The user selects the portion of SDS file that is to be converted and stored by the server as a Matlab file. The user now connects to the file server, which is also running on the server, and visualizes the Matlab file using some graph-drawing application that is on the client-side. Here, the server is sending prepared data to the client where it is visualized. The client has control over both the remote preparation and the local visualization. Communication is done using both HTTP and the file server protocols. The main disadvantage is that this system cannot run simulations in real-time and uses data as Matlab files. Because of these two restrictions, this system is not suitable for ViSGrEn.
2.1.2.6 Particle Tracing System

Particle tracing system, a distributed visualization system that employs several powerful machines for calculations and low-end computers for visualization, is used to visualize unsteady flows by computing particle traces [28]. The server and clients register themselves on the lookup service and the client requests the server for the desired simulation. The server performs the simulation and stores the data in the database on the server side. The client then loads results from the database and visualizes. While this system visualizes particle flows and is unsuitable for other types of output, the approach where the server simulates and stores data that is visualized by the client fits well with ViSGrEn and can be adapted to it.

2.1.3 Server-based Visualization

The following is an evaluation of some of the existing systems to understand if a server-based approach is appropriate for ViSGrEn.

2.1.3.1 Vis-a-Web

Vis-a-Web offers an easy-to-access visualization service on the WWW [26]. To keep this service hardware-independent, only established WWW technologies such as Java and VRML are used. The server has a WWW-server and a visualization server. The front end of the visualization server is a Java applet that is loaded onto the client. This applet drives the user interface and the communication between the visualization server and the client.
The client connects to the server and downloads the data. After the data is collected by the client and the input parameters are set, the visualization server is contacted using a socket connection, where the visualization is done according to the controls set by the user in the front-end. The client applet is then informed and the visualization is downloaded on the client and displayed as a VRML world. Here, visualization is being performed at the server side, thus making it a server-based visualization. Even though this system gives some control over visualization before it is rendered, the VRML scenes do not allow high interaction with the visualization once they are rendered on the client side. Also, this system does not have the facility to generate data in real-time, and therefore it is unsuitable for ViSGrEn.

2.1.3.2 Remote Visualization using the World Wide Web

Bock and Peters in [7] proposed an approach for viewing visualizations of data generated by supercomputers located at remote locations as a server-side visualization, with the server sending 2D images to the client. The client software is typically a web browser making the visualizations easily accessible. The server renders the data and sends a stream of images rather than geometry to the client. JPG compression is used to compress the image. Clients can send visualization parameters to the server through a Java applet using a socket connection. This technique is used for a variety of applications like the particle system visualizer. The main problem with this approach is that user does not have any interactivity with the image, and therefore this approach is unsuitable for ViSGrEn.
Figure 2.7 Vis-a-web System
2.1.3.3 COVISE

Collaborative Visualization and Simulation Environment is developed by Wierse et al. as an extendable distributed software environment to integrate supercomputer-based simulations, post-processing, and cooperative visualization functionality in a seamless manner [3]. COVISE is used in the aerospace or automotive industry to analyze the results of complex numerical simulations of car crash tests, the flow of air around a car or plane, and the design of an engine or of satellites. The data, typically in CAD (Computer Aided Design) format, resides on the server. When the client requests visualization, VRML scenes are generated at the server, and they are viewed in a viewer at the client side. A Java applet is used to control the visualization by sending parameters to simulation runs. The protocol used in communication between client and server is HTTP. Here also, the server not only generates data but also maps it to VRML scenes, making it a server-based visualization which gives limited interactivity with the visualization. This system does not meet ViSGrEn’s requirements of having high interactivity and therefore is not useful.

2.1.3.4 NOVICE

The NOVICE project (Network-Oriented Visualization in the Clinical Environment) is developing a range of extensible Web-based visualization tools for medical applications that will work within a high-performance computing environment (HPCN) [22]. Access to computer resources is provided through a simple Web-based interface using technologies like Java and Microsoft’s COM model. The main goal is to provide advanced technology to
medical professionals from an affordable desktop by bringing high quality medical image processing without a large capital investment to the clients.

NOVICE has both fat-client and thin-client approaches. In the thin-client model, the user contacts the server through a web browser. Using a form or Java applet, visualization parameters are sent to the server. CGI scripts at the server handle the visualization, and the results are sent as JPEG images, VRML scenes or MPEG movies using HTTP. Medical data is usually huge and complex, requiring large computational power for visualizing. Thus, a server-based approach in which visualization is done on a high-performance computer instead of sending huge data across a network is appropriate for situations involving no user interactions with the visualization. ViSGrEn needs high interaction with the visualization and the data dealt with is not very large, thus this approach is unsuitable to us.

2.1.4 Real-Time Distributed Simulations

ViSGrEn should be able to visualize real-time simulations and predictions, where the server generates data step by step. The client needs to get the data and visualize it as soon as the data is available. As the client needs to render and also read data, a producer-consumer scheme can be used where the producer (server) places data steps in a queue, and the consumer (client) pulls it from the queue. The following application describes some other approaches.
Evolving Distributed Simulations: Multiresolutional visualization of Evolving Distributed Simulations [10] uses Message Passing Interface (MPI) to set up a parallel environment for running complex simulations. The simulation program sends only the evolving data to the visualization side, as some data may not change during the simulation, reducing the amount of data that needs to be sent over the network. Overlapped data communication and visualization is used to increase speed. To have good responsiveness, one process is used to respond to the user’s input and another is used to receive data from parallel simulation. Another technique is to use polling in receiving the data. The receiving process checks if new data have arrived or not. If no new data arrived, then the process responds to the user’s input. Otherwise, it visualizes the new data. The third method is to use double buffering, where one buffer is used to receive data and the other is used to visualize the data. Double buffering enables the user to operate the visualization software even when it is receiving data. Even though this application does not provide a complete solution for ViSGrEn, it uses the technique of overlapped communication and visualization that can be applied to ViSGrEn.

2.2 Technologies Supporting Distributed Visualization

From the applications presented in the previous section, it is clear that a client-based approach is more suitable for having high interactivity with the visualization. Java applet, which is a client based approach, can be used to present the visualization. Some technologies that are required for developing the ViSGrEn system are presented below.
2.2.1 J2EE, Java3D, and Web portals

A platform-independent and easy-to-access web-based visualization system is needed. The Java 2 Platform Enterprise Edition (J2EE) defines the standard for developing multi-tier enterprise applications [2]. Thin-client applications invoke business logic that executes on an application server. J2EE encapsulates Enterprise JavaBeans (EJB) components, Java Servlets API, Java Server Pages (JSP), Java Database Connectivity (JDBC), and other technologies [20]. Enterprise JavaBeans architecture encapsulates the enterprise logic, such as database access, and security and isolates it from the application itself, forming a middle tier. JDBC is an API that gives access to the database from Java programming language. Web portals provide access to resources on the web and are used for applet-to-server communication.

Java3D is an object-oriented API developed by SUN Microsystems for incorporating 3D graphics into Java applications and applets. Java3D enables developers to build, render, and control the behavior of 3D objects and visual environments [24]. Java3D is a high-level library implemented on top of OpenGL or Direct3D. OpenGL and Direct3D are both low-level graphic APIs, OpenGL being a cross platform API while Direct3D is a Windows-only.

Java Web Start is a technology for simplifying deployment of Java applications, which can be used as an alternative to applets. With Java Web Start, applications can be launched simply by clicking on a web page link. If the application is not present on the computer, Java Web Start automatically downloads all necessary files. It then caches the files on the
computer so the application is always ready to be re-launched any time the user wants, either from an icon on the desktop or from the browser link. The most current version of the application is always presented to the user no matter what approach is followed.

2.2.2 Grid Computing

Complex simulations can be run on remote high-performance computers and visualized on PCs or workstations at client side. Grids provide remote access to geographically distributed resources like supercomputers, storage systems, and data sources. Globus toolkit is an implementation of grid technology providing software tools and services to enable the development of computational grids. The Globus toolkit uses the Grid Security Infrastructure (GSI) for enabling secure authentication and communication over an open network [1]. Back-end computing resources are accessed using Java-based Globus interface Java CoG (Commodity Grid kit). Java CoG is a library providing API to access grid services through Java. Open Grid Services Architecture (OGSA) is Globus 3.0 version. The Globus toolkit has Global Access to Secondary Storage (GASS) and Grid Resource Allocation and Management (GRAM) APIs. GASS is used for remote data access. It integrates GridFTP, HTTP, and local file I/O. GridFTP is a high-performance, secure, and reliable data transfer protocol that provides secure and efficient data transfer in grid environments [1]. GRAM allows programs to be started on remote resources.

Grid portal is a user’s point of access to a Grid system. Grid portals organize and manage distributed computing resources, services, and software components that form a
grid. They provide an environment where the users can access Grid resources and services, execute and monitor grid applications, and collaborate with other users. NEESgrid Portal and Alliance Portal are examples of existing grid portals. For this work, NEESgrid Portal is used. Computational web portal is an environment that extends the user desktop by providing a seamless access to remote computational resources [12]. Figure 2.8 shows the architecture of a Web-portal [4]. The simplicity of its interface hides all implementation details from the user. It is implemented as a multi-tier system. The following is an example that uses grid-based computing.

![Three-tier Architecture of Web Portal](image)

Figure 2.8 Three-tier Architecture of Web Portal
**Web-based Visual Exploration System:** visual Exploration, a system for Web-based visual data exploration, has three major components: a Web-based user interface to grid-enabled visualization services, visualization Web application which tracks the results, and a portal application server called VisPortal that manages and coordinates the authentication for using grid resources (Figure 2.9) [16]. Web server communicates with the client via HTTP.

![Figure 2.9 Architecture of Visual Exploration System](image)

This system is used to visualize Adaptive Mesh Refinement (AMR) data using AMR-WebSheet as the user interface. The architecture of VisPortal is based on GPDK (Grid Portal Development Kit), which uses Java COG toolkit. The portal launches a parallel computing component using Globus GRAM and initiates the connection between this computing component and a high-performance back-end data source like a running simulation code [16]. After user authentication, it launches the client through a web browser.
using MIME type or plug-ins and initializes a visualization session. The client in turn connects back to the remotely located visualization component, thereby completing the distributed visualization application. When the user finishes the visualization, the session is closed. This launching procedure is hidden entirely from the user by the portal client interface. The user simply selects remotely located data and presses a button to start the visualization. Using the grid portal, easy access to resources on the grid is possible. This system provides a standard architecture for grid-based computing. A similar three-tier architecture with an application server in the middle tier that hides the complexities of accessing resources on the grid can be used for ViSGrEn.

2.3 Other Related Work

This section presents some related work that would be useful for developing the Fire-Smoke Model project that will be used as a case study.

2.3.1 Fire Dynamics Simulator

Fire Dynamics Simulator (FDS), developed by NIST, is a computational fluid dynamics (CFD) model of fire-driven fluid flow [18]. FDS is a stand-alone system simulating and visualizing the propagation of fire and smoke. FDS solves numerically a form of the Navier-Stokes equations appropriate for thermally-driven flow with an emphasis on smoke and heat transport from fires. A visualization program called Smokeview, developed in C and Fortran 90, is used to display numerical predictions of particle flow, 2D-, and 3D-
shaded contours of gas flow generated by FDS simulations [11]. Smokeview performs visualization using OpenGL and GLUT (Graphics Library Utility Toolkit) by displaying time-dependent tracer particle flow, animated contour slices of computed gas variables, and surface data. The visualization scene can be rotated and translated with mouse or scene movement dialog box. Smokeview can be used as a post-processing step to visualize FDS data or during the simulation to monitor the progress of a simulation. Figure 2.10 shows a snapshot of a simulation of a kitchen fire in a townhouse.

![Figure 2.10 Snapshot of FDS](image)

FDS, being a CFD tool, has accuracy and precision in modeling fires and associated heat and mass transfer, but CFD simulations are computationally intensive in terms of time and memory requirements. Therefore, real-time simulations are not possible without
massive supercomputers. The townhouse case shown above required 8.5 hours of CPU time on a 2GHZ Pentium IV Windows XP system [18]. Furthermore, a detailed knowledge of real-life materials exposed to heat flux is required for successful CFD simulation of fire growth. FDS, being a stand-alone application, cannot be used by remote users and therefore is not useful for ViSGrEn.

2.3.2 Prototype of Distributed Fire-Smoke Model

A prototype model visualizing the geometry of the Shadwell submarine was built as part of my Directed Individual Study [25] using Java3D and J2EE technologies. This work was mainly aimed at understanding different distributed visualization approaches and finding the feasibility of Java3D for getting high quality graphics [25]. This model acts as the starting point for the implementation of the distributed Fire-Smoke application that would be used to test the hypothesis.

2.4 Summary

The applications discussed above provide the state-of-the-art distributed visualization. It is clear from the above-mentioned examples that if high interactivity with the data set is required, a client-based approach, which allows data, visualization and interaction to occur locally, is better suited. A server-based approach is suited for expensive rendering problems like volume visualization, large data sets, novice users, and for a particular group
of users where the visualization service needs to be tailored according to the needs of the users [8].

All the mentioned systems give a solution to a narrow problem or have some limitations. FDS, being a CFD-based model, is very slow for real-time simulations, and it is a stand-alone system. FAST runs only on SGI machines, Vis5D is just a viewer for 5-D data, and VizWiz uses Java AWT and therefore cannot make use of graphics hardware or advanced visualization features. Molvis is suited for chemical data and visualizes in VRML, not giving much interaction with the visualization. RIKZ uses data only in SDS format. Server-based visualization is not appropriate for ViSGrEn, as the client needs high interaction with the visualization. Therefore, none of the systems mentioned under that section are suitable. The technique of overlapped communication and visualization presented under the real-time simulation section can be used. VisualExplorer, described in the grid computing section, provides a standard architecture for the grid environment. These works do not provide a complete solution to satisfy all requirements.

None of the presented systems satisfy the requirements of having a high responsive visualization system that predicts and visualizes events in real-time, being accessible to all through a simple interface, and having a grid-based backend to access high-performance computers. Therefore, they cannot be used as off-the-shelf components but will be used as references. ViSGrEn should be able to visualize the simulations generated on the server side in real-time or replay the simulations that are already generated with minimum latency.
CHAPTER III

VISGREN REQUIREMENTS AND ARCHITECTURE DESIGN

The goal of this research is finding an appropriate architecture design that will be useful in developing a grid-based simulation system that provides highly interactive real-time visual information to remote users. In this chapter, the requirements for real-time simulation visualizations in the grid environment are described first. These requirements are used to design the architecture of Visualization System for the Grid Environment (ViSGrEn) that is described next.

3.1 ViSGrEn Requirements

The requirements are divided into user requirements and system requirements as follows.

3.1.1 User Requirements

The target users of this system are emergency response teams who take action based on the prediction outcomes, designers who experiment with different “what-if scenarios” and modify the design accordingly, and trainees who use the system to know how the real-world system would perform under various conditions and how to respond under those
conditions. The user requirements for these three types of users are further divided as follows.

3.1.1.1 For Web-browser based

The requirements for making the system Web-browser based are

**Remote Access:** There can be more than one person who needs to know about the situation or take action, and these people might be at different locations from the actual scene. Thus a remote access to the system is needed. In order to run complex simulations or predictions in real-time, remote access to high-performance computers located on the grid is needed. The simulation job is run on a grid resource, and the results are visualized to the clients.

**Platform Independent:** As mentioned above, there are many users of the system and those users may have different computers with different operating systems. Thus the system should run on all platforms.

**No Client-side Installations:** The installations required at the client side in order to run the system should be at minimum.

**Simple to Use:** It should be noted that the users of the system would be operating under extreme conditions with a threat to both property and human lives, including their own. Therefore the system and the graphical user interfaces provided to create the simulations
should be simple and easy-to-use. The system should not require any elaborate operations to run and visualize the simulations. Accessing grid resources, submitting jobs, etc should also be simple and transparent to the user.

3.1.1.2 For Collaboration

The requirements for having collaboration among users are as follows

Sharing Geometry Data: One of the advantages of having a Web-based visualization system is that it is accessible to anyone from any place. Before actually building any structure, several designs are created and tested for vulnerability. Thus a facility to share the model designs between various users is required.

Sharing Simulation Results: The designers can take the advantage of remote access to get their simulation results analyzed by geographically distributed experts. Thus a facility to share the simulation results by storing the results and replaying them at a later time is needed.

3.1.1.3 For High Interactivity

The ViSGrEn system should allow the clients to have high interactivity with the visualization. The user should be able to rotate, translate, set states of objects, etc. The system should respond immediately to the user actions, which is very important in case of emergencies. For example, consider a case where there is a fire accident in a building. The
system predicts the events in real-time, and the fire brigade team turns on the suppression to put out the fire. If the system takes long time to actually turn on the suppression, the damage caused would be more, and there would not be any use of making real-time predictions. Thus, the system should be highly responsive to user actions. The following are some other requirements for high interactivity.

**Real-time Visualizations:** The system should be able to run predictions and visualize them to the users in real-time or faster than real-time allowing necessary actions to be taken to reduce the damage caused by the actual events.

**Set Simulation Parameters:** The user should be able to set various simulation parameters and states of different objects like doors easily and quickly.

**Modify Geometry:** The users, especially the designers, should be able to plugin different geometries or modify the existing geometry in order to test their designs. Thus the system should allow for easy modifications of the geometries.

**Run Different Simulation Models:** Usually the designers do not need real-time visualizations, but they will be interested in a more accurate and detailed output of the simulations. Therefore they need to run more complex and computationally intensive simulations like CFD simulations, which take hours of computational time. The emergency response teams, on the other hand, are more interested in having real-time visualizations than hav-
ing accurate simulations. Thus a facility to run different simulation models for the same data set is needed.

### 3.1.2 System Requirements

Apart from the user requirements, there are some requirements for the system that would indirectly affect the user requirements.

#### 3.1.2.1 Scalable

The system should be scalable to an increasing data sizes. Scalability to increasing number of users is not a major concern as the number of users accessing the system at a time would be around 20 or 30 but not hundreds or thousands.

#### 3.1.2.2 Fault Tolerant and Reliable

ViSGrEn system relies on multiple resources like the server, grid resources, and network to function. There is an increased chance of developing faults somewhere in the system. The system should be able to recover automatically from any broken network links, server or HPC crashes, lost data packets, etc.

#### 3.1.2.3 Hardware Requirements

The clients should have the following minimum requirements in order to have smooth visualization: CPU: 1GHz, RAM: 256MB, video RAM: 32MB, and network bandwidth: 10Mbps.
3.2 ViSGrEn Architecture Design

Based on the requirements listed above, an architecture for ViSGrEn is developed as described in this section. Simulations are run on high performance-computers located on the grid and the results are visualized to the client. To run jobs on the grid, an easy and secure way to access and submit jobs on the grid is required. Client-server implementation allows computing on high-performance systems while rendering, viewing, and interacting with data on a lower end workstation. Such a system should be portable across different platforms. Therefore it is made accessible through a simple interface provided by a web browser. To satisfy the requirement of having a web-browser based system, Java is used to develop the case-study system. As Java is used, Java3D is chosen instead of OpenGL for rendering visualization. Java3D applets enable platform-independent visualization of data, allowing anyone with a Java3D-enabled browser and network connection to view and interact with the visualizations [9]. A three-tier architecture as shown in Figure 3.1, with the client’s interface as a web browser-based front end, web server and application server as the middle tier, and distributed computing resources as the back end is proposed. The middle tier does the request processing and response generation for the clients. The back end provides computing resources through grid services such as GridFTP and GRAM to execute the simulations as requested by the users. The web server supports Java Server Pages (JSP) and launches the applet in the client’s browser. The application server acts as a grid portal for delegating simulation jobs onto remote high-performance computers. ECS (Enterprise Computational Services), a grid portal developed at Mississippi State
University’s Engineering Research Center, is used as the application server, providing authentication, job submission, and file transfer to distributed computing resources available on the grid. HTTPS is used for communicating between the client and web server, RMI (Remote Method Invocation) between the web server and application server, and GRAM (Grid Resource and Allocation Management) between the application server and grid services. The simulation results are brought by the server from a high-performance computer and sent to the client for visualization using GridFTP protocol.

This architecture provides an environment for experimenting with the test case system, thus acting as a test bed. It allows separation of the visualization task from the simulation task. The grid portal, i.e., ECS is already implemented. In this work, the Java client and server code is developed and communication between them is facilitated. This research helps in understanding where most of the time is spent for grid-based real-time visualizations and optimizes the architecture in order to increase the performance. It will also be found out if Java3D performs well enough to satisfy the requirements or if there is a need to compromise the functionality or graphics quality.
Figure 3.1 System Architecture for ViSGrEn
CHAPTER IV

IMPLEMENTATION, EVALUATION, AND OPTIMIZATIONS

In this chapter, the implementation details, features supported by ViSGrEn, evaluation of the baseline system, and different optimizations used to increase the performance of ViSGrEn are presented.

4.1 Testbed for ViSGrEn

The testbed is a placeholder for plugging-in the Java applet code that communicates with the database and grid resources. The framework shown in Figure 3.1 will act as the test bed. A client machine with Java3D-enabled browser acts as the front end. Browser requires installing Java3D plugin apart from the Java plugin for the browser. For the middle tier, a web server (Tomcat 4.0) that supports JSP and JDBC and an application server that acts as a grid portal for delegating the simulation jobs onto remote high-performance computers are used. Web server and application server are installed on 'Spur’ machine at ERC. ECS [12] is used as the middle tier in which all these functionalities are already implemented. Database server can be installed on the same machine as the server or on another machine. MySQL is chosen for the database, as it is available for free. MySQL is installed on SPUR machine to act as the database server. One of the main requirements
of ViSGrEn is to have high interaction with the visualization at the client side. For high interaction, client-based rendering as discussed in section 2.1.2 where the whole data is transferred to and rendered at the client side is chosen. ViSGrEn is implemented as a Java applet that runs in a browser. To request the visualization, the client connects to a JSP, which will download the applet into the browser.

4.2 ViSGrEn Features

In this section, some of the main features that are supported by ViSGrEn are described.

4.2.1 Creating Simulations

For creating simulations, many input parameters must be sent from the client to the server. ViSGrEn provides an elaborate Graphical User Interface (GUI) to set the simulation parameters and states of different geometry objects. These simulation parameters are stored in a class called `SimulationParams`. Instead of sending each parameter separately, the object of this class is sent to the server. An input file is then created at the server side using the parameters sent by the client and by reading other parameters from the database. During the replay of simulations (details are given in the next section), the input file is parsed to display the simulation parameters and to set the state of the system before the simulation. A class is created to handle the simulation input files. This class, called `Sim-
InputHandler, contacts the server to either send (during simulation creation) or receive (during simulation replays) the input parameters.

### 4.2.2 Plugging-in Different Geometries

ViSGrEn allows the users to plugin different geometries. A list of available geometries (databases) is provided in a file on the server side. This list is read and displayed by the applet to the user who can then select the geometry he/she is interested in. This feature is used by designers to experiment with different designs of the structure before actually building them. The menu shown in Figure 4.1 allows to plugin different geometries.

![Figure 4.1 Facility to Plug-in Different Geometries](image)

Figure 4.1 Facility to Plug-in Different Geometries

### 4.2.3 Replaying Simulations

One of the main advantages of having a Web-based simulation visualization system is that the simulation results are accessible to geographically distributed experts for analysis. Thus a facility where any client can replay and visualize the pre-run simulations is
provided. To have this facility, whenever a simulation is created, the simulation results are saved along with the input file at a location on the server called *Simulation Repository* and an entry is added to the list of available simulations. When the client wants to replay a particular simulation, then this list is shown, and he/she can choose one simulation from the list and it would be replayed. Figure 4.2 shows the list that is displayed to the user. A facility is provided to the clients to delete or to make private the simulations that he/she created so that those simulations are not available for other clients.

Each step has a step number indicating the sequential count of the steps generated, and step time indicating how many seconds out of total simulation duration have passed. In replay mode, the client can choose to sample in time or space [23] by providing time or step interval respectively. For example, if sampling in space with a step interval of 10 is given, then every 10th step is visualized by skipping the other steps. For sampling in time with a time interval of 10 seconds, after visualizing the first step, every step whose step time differs from the previous visualized step by 10 seconds or more is visualized. For having a replay in real-time, sampling in space cannot be used as the density of steps generated per unit time is not constant and visualization gets slower for higher densities. Sampling in time is useful for such situations. Figure 4.3 shows this feature.

### 4.2.4 Fault-tolerance for Lost Data Packets

Simulation results are transferred from the Grid resource to the client in packets. There is a high chance of losing some data packets in the transfer. If whole or some part of a
Figure 4.2 List of Pre-run Simulations Available for Replay

Figure 4.3 Sampling in Time or Space
simulation step is lost, ViSGrEn handles the situation by substituting the previous step’s data in place of the lost data and continues to visualize. Thus the visualization may appear to be ‘frozen’ at times when the data is lost, but never hangs-up.

4.3 Implementation of the Baseline System

For identifying the constraints of ViSGrEn, Fire-Smoke simulation visualization system [23] is re-implemented on the proposed architecture. This system, originally implemented by Dmitry Shulga as a stand-alone system using C++ and OpenGL, simulates the propagation of fire and smoke in a ship. The geometry data of Ship was generated from AutoCAD files provided by Havlovic Engineering Associates, Inc., and the network model that runs simulations was provided by Hughes Associates, Inc. This system is re-implemented using Java and Java3D as a stand-alone system. Then it is ported onto the architecture to make it a Web-based system without using any of the proposed optimizations. This Web-based system is called the baseline system. Having the same application in three stages (stand-alone C++ & OpenGL, stand-alone Java & Java3D, and Web-based Java & Java3D) helps in understanding the limitations of Web and Grid environments, separate from the limitations of Java and Java3D. The proposed optimizations are implemented on the baseline system to get the optimized Web-based system.

The Java3D scene graph structure for the Fire-Smoke system is shown in Figure 4.4. The scene graph is a data structure that contains all information to display the Java3D universe. There are three main types of nodes in the scene graph: group nodes represented
with circle, leaf nodes represented with triangle, and node components represented with rectangle.

The performance is measured using factors such as latency in initial viewing of geometry, also referred to as startup time, interactivity and response time, and simulation visualization time. These figures are compared for three systems (stand-alone C++ and OpenGL, stand-alone Java and Java3D, and baseline Web-based system). The stand-alone C++ and OpenGL system was modified in order to have one view and no extra menus so that it is consistent with the Java3D system.

The datasets used are geometries of different ships or different parts of the same ship. Six different data sets: ShadwellSample1, ShadwellSample2, Shadwell1, Shadwell2, Peterson1, and Peterson2 are used. All these datasets have vertices, sides, walls, and compartments. Peterson data has a suppression system, whereas Shadwell data has a network system. In order to make the datasets consistent, these elements are not read or rendered. The data to generate simulation input files is available only for Peterson1. Therefore, only Peterson1 data is used to test the performance of simulation visualizations. The database size is taken as the number of vertices of the polygons that are drawn plus the number of junctions. The above-mentioned six databases have the sizes as follows: ShadwellSample1 = 1029, ShadwellSample2 = 2058, Shadwell1 = 4275, Shadwell2 = 8550, Peterson1 = 11350, Peterson2 = 22700. The symbols used for timings are Tgl (Time for stand-alone OpenGL and C++ system), Tsj (Time for stand-alone Java and Java3D system), Tbj (Time for Web-based baseline system), and Topt (Time for the optimized Web-based system).
Figure 4.4 Java3D Scene Graph Structure for Fire-Smoke System
The timers used are `System.currentTimeMillis()/1000.0` in Java, which gives time in seconds, and `clock()/CLOCKS_PER_SEC` in C++, which again gives time in seconds.

The Fire-Smoke system that is used for the case-study is both graphically and computationally intensive. Therefore, its performance greatly depends upon the graphics hardware and the CPU speed. The computer (Fire-PC) that is used for the experiments has a 2.6 GHz Pentium4 processor, 128 MB RAM, and ATI Fire GL 8800 Video Accelerator with 128 MB of video memory and 400 MHz of internal DAC (Digital Analog Converter). The network used is Ethernet with 100Mbps bandwidth. In taking the time measurements, it was noticed that the display area size and the scene size as displayed on the screen have a major impact on the performance. If the display area is small, the interactions and rendering is much faster. Also if the scene is zoomed out to a very small size, visualization is faster even though the display area size is large. Thus, both the display area size and the scene size had to be the same in all four systems to get the correct results.

### 4.4 Evaluation of Baseline System and Optimizations

In this section, evaluation of the baseline system to find the major constraints, and various optimizations used to overcome the identified constraints are discussed. Measurements used for evaluation are startup time, interactivity and response time, and simulation visualization time.
4.4.1 Startup Time

Startup time, also called latency in initial viewing of geometry, is the time to get the application up and running by reading the data and rendering the geometry. This time is subdivided into time for getting data from the database and time for rendering the data.

\[
\text{Startup Time} = \text{Time to get the data from the database} + \text{Time to preprocess} + \text{Time to render the data}
\]

4.4.1.1 Getting Data from the Database

The graph in Figure 4.5 shows time for getting data from the database plotted against the database size. From the Tgl and Tsj curves it is clear that Java is slower than C++ in reading from the database and filling the defined data structures. The baseline Web-based Java system (Tbj curve) performs very poorly for large data, which is not due to communication time over the network, but due to different Java Virtual Machine (JVM) on the server. For Peterson2 data, it is observed that it takes 11.0 secs out of 11.52 secs to read data from the database on the server and only 0.52 secs in communication. Different JVMs are optimized for different purposes and their behavior is not very predictable. Increasing the heap size for the JVM did not have any effect in this case. Optimizing the JVMs is out of the scope of this work and so this issue will not be addressing in this work.
Figure 4.5 Time for Getting Data from the Database
4.4.1.2 Optimization: Preprocessing and Caching the Database

Offline preprocessing and database caching are used to reduce the time in reading the data from the database. To understand the advantages of caching, consider an example where an application takes ten seconds to generate a big set of data using complex query with multiple table joins. If this data is cached in memory, subsequent retrievals can directly access the cached data instead of running the query again. Caching is possible only if the data does not change during all the retrievals. Typically caching is done on the client-side making it useful when one client frequently needs the same data from the database. In case of ViSGrEn, whenever the client requests visualization, the data should be read from the database and sent. The database is normalized to have flexibility and to reduce data redundancy. When tables are normalized, querying gets complicated requiring linking up of several tables to get meaningful data. Converting the raw data in normalized tables into geometry data that is required in rendering constitutes preprocessing. It is to be noted that the data in the database is never altered by the clients. Even though the clients are allowed to modify the geometry for simulation purposes, these modifications are stored as a state of the simulation and are not updated in the original database. Thus it is sufficient if preprocessing the raw data in the database to the geometry data is done once instead of every time the clients request visualization. The concept of database caching is extended to ViSGrEN by caching the preprocessed data on the server side. Server side caching is useful since all clients need the same geometry data. For ViSGrEn, database is modified very infrequently (like once in a few months). Therefore the preprocessed cache
must be saved to a persistent storage like file or database instead of to the main memory. As reading from a file is much faster than reading from a database, the preprocessed data is cached in a file.

A package is created that reads the database, preprocess and stores the data in a class acting as a data-holder whose object is used in rendering routines. Now this object of the data-holder class must be stored as a persistent object on the server. Java provides ObjectOutputStream and ObjectInputStream in java.io to write and read objects respectively. For writing an object, its state is represented in a serialized form so that the object can be reconstructed when it is read. Thus object serialization which allows read or write a whole object to and from a raw byte stream is needed. Serialization guarantees all ACID (Atomicity, Consistency, Integrity, and Durability) properties for the data but does not provide typical database features like indexed access, caching, and a query language. Long-term storage needs should continue to be catered by conventional relational or object-oriented databases. In order to serialize a class, an interface called Serializable available in java.io package must be implemented. Serialization is a recursive process, and therefore all classes in the package must also be serialized. The serialized object of data-holder class can be written to a file on the server, typically saved with an extension .ser to represent that the file has serialized data. Whenever the user requests data, the file is accessed instead of the database and is sent as an ObjectOutputStream to the client. The client reads the stream using ObjectInputStream and deserializes it to the data-holder object. For deserializing, the package should also be present at the client side. After deseri-
alizing, the client has access to all the functions and variables defined in the package. Thus in this method whole data is sent in one step as one big chunk. The Figure 4.6 shows the preprocessing and caching phase. The limitation of this method is that the provider must generate the preprocessed database caches (i.e., serialized files) whenever the database is updated. The database changes are not very frequent, and so creating a serialized file every time the database is changed is a minor overhead when the time saved in reading the database and preprocessing the data is taken into consideration.

![Figure 4.6 Preprocessing and Caching the Database](image)

By using the above technique, time is saved in:

1. Accessing the database: Instead of accessing the database, file is now accessed which is faster.

2. Communication: Instead of sending each value separately, all the data is stored in one class and its object is sent. Thus there is just one communication call between server and the client.

3. Preprocessing: Preprocessing is totally eliminated saving all its time.
4.4.1.3 Rendering Time

The graph in Figure 4.7 shows rendering time vs database size. The Java3D systems (curves Tsj and Tbj) perform very poorly when compared to the OpenGL system (curve Tgl). The initialization time (1 sec) itself is very significant for Java3D when compared to the initialization time for OpenGL system (0.5 sec). Rendering time for Java3D increases steeply as the data size increases confirming that Java3D is not only slow but also scales very poorly when compared to OpenGL. It is also seen that the baseline Web-based system performs better than the stand-alone system, which is again due to different JVMs used for the browser and for stand-alone execution. The percentage loss in performance due to Java3D when compared to OpenGL is between 200% to 340% depending upon the complexity of the geometry. Even though Java3D performance is poor, a platform-independent system can be achieved by developing the system in Java. Therefore, to cope up with Java and Java3D performance loss, the following optimization is proposed.

4.4.1.4 Optimization: Progressive Rendering

Progressive rendering and interlacing are often used in Web-designing to mask the time delay in downloading large JPEG and GIF files. In these techniques a low-resolution version of an image is displayed while the file is still downloading. By using these techniques, it looks as if the images are downloaded faster because the rough previews of images come up faster. Same technique can be used for ViSGrEn. It is not possible to reduce the rendering time, but some of the rendering time can be masked by running the
Figure 4.7 Time for Rendering the Geometry
rendering routine as a background process. The main motivation for progressive rendering comes from the fact that not all the objects are needed at the very first instance. Initially, only the basic structure of the model can be shown and other details can be rendered when the user is trying to figure what operation to do next. To understand better, consider an example where a building structure is being rendered. Here, the main objects are floors and rooms, and these are rendered first. Objects like doors, windows, and other details are rendered in a background process.

The main thread, after rendering the basic structure, creates and starts another thread for rendering the details. This second thread called **progressiveRenderer** thread is given the lowest priority by using the Java command `thread.setPriority(Thread.MIN_PRIORITY)`. While the user is clicking buttons or rotating or translating the scene, this thread will finish creating and rendering the details that are then added to the scene. For replaying or running simulations, detailed objects are needed in order to set their states. Thus, an exception will be thrown if creating and rendering the detailed objects is not complete by then. Therefore, the priority of the **progressiveRenderer** is increased when the simulation button is pressed. This will force the creation and rendering of the details by the time the user is ready to start the simulation.
4.4.2 Interactivity and Response Time

Interactivity and response time is how fast and easily the client can interact with the visualization by rotating, translating, zooming, changing object states, etc., and how fast the system responds to the user actions.

4.4.2.1 Optimization: Client-based Rendering

From the literature review, it is clear that only client-based rendering can provide high interactivity with the visualization. Thus client-based rendering is used for the baseline system without conducting any experiments with the server-based rendering.

4.4.3 Simulation Visualization Time

Simulation visualization in the grid environment involves producing the simulation output in steps, sending the data to the client, reading each step, and visualizing the data step as soon as it is available at the client side.

i.e.,

\[ T_{wcreate} = T_{exe} + T_{com} + T_{viz} \]

where

\[ T_{wcreate} = \text{Time taken for Web-based system to create and visualize simulation} \]

\[ T_{exe} = \text{Time for executing the simulation run by the simulation model on the grid} \]

\[ T_{com} = \text{Time for communicating to the client over the network} \]

\[ T_{viz} = \text{Time for visualizing the simulation data} \]
\( = \text{Time to read the data} + \text{Time to display the data} \)

There arise three scenarios as shown in Figure 4.8

Figure 4.8 Different Cases for \( T_{\text{exe}}, T_{\text{com}}, \) and \( T_{\text{viz}} \) for a Three Step Simulation

where \( \Delta_{\text{exe}} = \text{Time for producing a single simulation step} \)

\( \Delta_{\text{com}} = \text{Time for sending a single simulation step} \)

\( \Delta_{\text{viz}} = \text{Time for visualizing a single simulation step} \)

Case 1:

\( T_{\text{viz}} \geq T_{\text{exe}} \) and \( T_{\text{com}} \)

\( T_{\text{total}} = T_{\text{viz}} + \Delta_{\text{exe}} + \Delta_{\text{com}} \)

Case 2:
\[ T_{com} \geq T_{exe} \text{ and } T_{viz} \]

\[ T_{total} = T_{com} + \Delta_{exe} + \Delta_{viz} \]

Case 3:

\[ T_{exe} \geq T_{com} \text{ and } T_{viz} \]

\[ T_{total} = T_{exe} + \Delta_{com} + \Delta_{viz} \]

\( T_{exe} \) depends on how fast the simulation model runs and how fast the computer running the model is. \( T_{com} \) depends on the network bandwidth. This research concentrates only on decreasing the \( T_{viz} \). The proposed techniques are effective only for case 1 where the simulation output is produced and sent to the client at a faster rate than the client can handle.

In order to ignore \( T_{exe} \) from calculations, time taken for the Web-based system to visualize the pre-run simulations i.e., time for replay (\( T_{wreplay} \)) is used instead of \( T_{wcreate} \).

\[ T_{wreplay} = T_{com} + T_{viz} \]

Time for replaying pre-run simulation in the stand-alone system is \( T_{sreplay} = T_{viz} \)

Figure 4.9 shows the graph plotted for simulation duration time versus visualization time for replaying the simulations. Simulation duration time is only an index of the size or length of simulation and is not the simulation execution time. ECS grid portal uses NEESgrid and therefore NEESgrid Streaming Data Services (NSDS) protocol is used to stream the data from the grid resource to the client. There is an initial latency of around 4.5 seconds due to NSDS streaming to get the first byte of data from the grid resource to the client. Also, the NSDS streaming is very slow when compared to HTTP transfer. The
problem with HTTP is that the file should be located in the Web directory of the server and cannot be used for transferring the data from a local directory of the grid resources. Therefore NSDS streaming is used, which is still immature as it is in the development stages. Hopefully future releases of NSDS would improve the streaming. Though the initial latency affects the total visualization time significantly for smaller simulations, it can be ignored for larger simulations.

To keep $T_{wreplay}$ close to $T_{sreplay}$, quality of visualization should be compromised to compensate for the extra time taken in communication. Therefore optimization techniques are applied to reduce visualization time.

4.4.3.1 Optimizations: Reducing Simulation Visualization Time

As mentioned above, visualization involves reading the data and displaying it. Optimizations proposed are described as follows.

1. Overlapping communication with visualization, a well-known technique used in client-server visualization
2. Using byte arrays in a circular buffer instead of Java piped streams, a technique taken from [21]
3. Reducing frame rate, a well-known technique used to decrease the animation time
4. Skipping simulation steps, a method adapted from reducing frame rate technique

First two techniques concentrate on reducing the data reading time, while the later two concentrate on reducing the display time.

To overlap communication and visualization, which reduces the time to wait for the data, two threads are created for reading the data and visualizing the data that is read. The
Figure 4.9 Visualization Time for Simulation Pre-runs
data is sent to the client from the grid-resource as discrete byte packets. The visualization thread can consumes this data at a rate faster or slower than the rate of receiving, which is a typical producer-consumer problem. Java piped I/O streams manage synchronization of the data buffer across producer and consumer threads. In order to send continuous stream of bytes to the visualization thread, the data is written to a Java `PipedOutputStream` which is connected to a `PipedInputStream`. The visualization thread can then read the data from `PipedInputStream` continuously. When no data is available, the thread waits until the end-of-file or until more bytes are written to the stream. When the consumer (visualization task) falls behind the producer (simulation task), the data is stored temporarily in the stream. Using piped streams is not an efficient solution because the Java implementation of piped streams is very inefficient. Thus, an alternative approach [21] where the consumer directly uses byte arrays is applied. In this method, the byte arrays are stored as Java Objects in a circular buffer. A circular buffer has two marks: `writePosition` and `readPosition` and a count of number of unread objects in the buffer. The circular buffer is implemented with blocking read and write operations meaning when the buffer is empty (number of unread objects equals to zero), the consumer waits till an object is written and when the buffer is full (number of unread objects equals to buffer length), the producer waits till an object is read. Java methods of `Thread` class, `wait` and `notify`, are used to make the threads wait on each other. The object read is converted into byte array and used. End-of-file is detected when the byte array received is null or when one of the byte values in the array is -1.
Third optimization is reducing the frame rate. Frame rate is the number of frames of an animation that are drawn on a computer monitor in one second. It is measured in frames per second (fps). A higher frame rate allows smoother animation but requires more processing power and system bandwidth between the graphical card and the main memory. Usually, the frame rate is 60 to 80 fps. The human eye can see the new frames coming if the frame rate falls to less than 30 fps. If the frame rate is high, most of the time would go into the rendering process because the rendering threads are given the highest priority as in the case of Java3D. Thus, if frame rate is decreased a little, the animation time would be less as some of the simulation steps are skipped without rendering. If the frame rate is too small, the visualization gets jerky and the interactivity would also decrease. As one of the requirements of ViSGrEn is high interactivity, the technique of decreasing frame rate is unsuitable.

A better approach is to skip some of the simulation steps so that interactivity is not reduced. Motivation for skipping the steps comes from the fact that complex simulations generate simulation steps for every fraction of a second. A reasonable quality of visualization can be maintained even if the output generated at every one second of simulation duration is visualized. Thus, the steps in between can be skipped.

Skipping can be done at three levels:

1. Simulation model skips generating the data step, saving $\Delta_{exe} + \Delta_{com} + \Delta_{viz}$ times.
2. Server skips sending the data step to the client, saving both $\Delta_{com}$ and $\Delta_{viz}$ times.
3. Client skips visualizing the data step after reading it, saving only $\Delta_{viz}$ time.
First option is the best one as it saves more time but, it may not be feasible because the simulation models do not often provide control over the output rate during simulation runs.

Other points to consider are at what rate the data steps are skipped and who decides the value of skip rate. If the steps are skipped at a frequent rate, the visualization becomes jerky and important details might get lost. If the rate of skipping is very less, it might not result in the desired performance. One method is to compare the current step’s time with the previous step’s time and if the difference is within the agreed range, then skip the visualization of the current step.

4.4.4 Summary of Evaluation

From the evaluation of the baseline system, research question 1 from section 1.4 can be answered as follows.

The constraints imposed by the Web and the Grid environments that affect the performance of the system are

1. Latency in initial viewing of geometry (startup time)
2. Poor performance of Java and Java3D
3. Initial latency and poor performance of NSDS streaming

In contrary to what was expected, communication time is not a major constraint. From section 4.4.1.1, it is clear that more than communication, it is Java that consumes time in reading large data from the database. From section 4.4.1.3, it is observed that Java3D is
much slower in rendering when compared to OpenGL. Developing a Web-based system requires using Java and Java3D, thus decreasing the performance of the system. Because of the NSDS streaming, there is an initial delay of around five seconds to receive the first byte of data from the grid resource. Thus, techniques are required to decrease the startup time, to hide or overcome the performance loss due to Java and Java3D, and to overcome the latency in communication.
CHAPTER V

EXPERIMENTAL RESULTS

ViSGrEn resulted in a proposed and implemented architecture for real-time Web-based visualizations in the grid environment. ViSGrEn is remotely accessible for demonstration purpose. It runs in a simple interface provided by a Java3D-enabled web browser, as shown in Figure 5.1. It has the facility to plugin different geometry models.

![ViSGrEn Running as an Applet in a Browser](image)

Figure 5.1 ViSGrEn Running as an Applet in a Browser

The user can interact with the visualization by rotating, translating, and zooming into the scene. Figure 5.2 shows snapshots of the scene after various interactions. The user can also change the states of different objects. It has the facility to replay pre-run simulations. The user can select from the list of available simulations on the server.
It also has the facility for creating and running simulations on the grid. The server has access to remote high-performance computers that run the predictions and simulations. Accessing the grid resources is transparent to the user. A GUI is provided to the user for setting various parameters of the simulation and for changing the states of objects, like opening or closing the doors. Simulations are visualized in real-time by running the simulation model on grid resources and displaying the output to the client.

The architecture also allows to run different simulation models. It is also fault tolerant to lossy data transfers. When some part of the simulation output data is lost in the transfer, the system copes up by getting the lost data from the previous simulation step. With all the above features, this system acts as a remote design and training tool for the designers and trainees.
Figure 5.3 Graphical User Interface for setting simulation parameters
5.1 Issues Encountered

The following are some of the issues encountered in implementing and testing ViS-GrEn.

5.1.1 Java3D with Swing Bug

There seems to be a bug in using swing with Java3D. The swing components like menus and dialogs keep flickering when used under some graphical cards and operating system. The flickering is observed when run on a machine with ATI Fire GL 8800 graphical card with windows XP operating system. This was not observed on any other machine. Hopefully, the future releases of Java3D will fix this problem.

5.1.2 Timers

Another problem is with the accuracy of the timers. Built-in Java timer’s method System.currentTimeMillis() for measuring the times is used. This timer does not return timing information in milliseconds, as its name might imply, but rather in discrete numbers separated by 16 milliseconds. Thus, its precision is around 16ms on Windows NT/2k/XP operating system. The loss of precision could have introduced a significant error in the results of interactivity times as they differ in only a few milliseconds. The error percentage is more prominent for smaller loops (less than 50). Therefore to decrease the error percentage, loops of 100 and 1000 are used to take the measurements.
5.2 Performance of ViSGrEn

The optimization techniques described in section 4.4 are implemented for ViSGrEn. To find out the effectiveness of these techniques, performance measurements such as latency in initial viewing of geometry, also referred to as startup time, interactivity and response time, and simulation visualization time are taken for the optimized Web-based system (Topt curve in the graphs) and compared with Web-based baseline system (Tbj), stand-alone Java and Java3D system (Tsj), and stand-alone C++ and OpenGL system (Tgl).

5.2.1 Startup Time

The optimizations used to decrease the startup time are:

1. Preprocessing and caching the data at the server side
2. Progressive rendering

For details, refer section 4.4.1.

5.2.1.1 Getting Data from the Database

In the evaluation of baseline system, time to get the data from the database was measured for the three systems namely Tgl, Tsj, and Tbj. The proposed optimization preprocessing and caching optimization is implemented for ViSGrEn and the time to get data from the database is measured again for the optimized Web-based system (Topt). The graph in Figure 5.4 shows this time plotted against the database size. This preprocessing and caching optimization eliminates the time to read the database and fill data structures
from the startup time. Therefore this method is also effective in hiding the performance loss due to JVM on the server. The optimized version (Topt) performs very well for any data size. The increase in time with the increase in data size is negligible for the optimized Web-based system. Preprocessing and caching and sending whole data in one step as an object produced an increase of 95% in performance for Peterson1 data, which is a tremendous increase.

5.2.1.2 Rendering Time

The graph in Figure 5.5 shows rendering time versus database size. For Topt, the rendering of junctions and smoke compartments was delayed, and so this time did not count into Topt’s rendering time. The data sets used for the graph plots do not have any complex detailed objects. This optimization of progressive rendering is effective for datasets having more detailed objects and/or more complex detailed objects. To demonstrate the effectiveness of progressive rendering, Shadwell and Peterson1 datasets are used. Shadwell has a network system consisting of ventilation pipes, water mist pipes, etc., which are represented using cylinders and spheres, which are graphically complex primitives. Peterson1 has a suppression system represented by cones that are again complex graphics primitives. The bar graph in Figure 5.6 shows the rendering times for these datasets. Progressive rendering increased the performance of the optimized Web-based system by 43.3% for Shadwell and by 17% for Peterson1 when compared to the baseline Web-based system.
Figure 5.4 Optimized Time for Getting Data from the Database
Figure 5.5 Optimized Time for Rendering the Geometry
Figure 5.6 Performance Gain Due to Progressive Rendering Technique
5.2.1.3 Total Startup Time

Total startup time is the time to get the data from the database plus the time to render
the data. Figure 5.7 shows the graph plotted for total startup time versus database sizes.
For the startup time, it is observed that the optimized Web-based system performs very
well for all data sizes and even better than stand-alone OpenGL system for very large
data sets. For Peterson1 data, the percentage increase in performance for the optimized
system is 68%. There is a total decrease of 4 seconds in the startup time due to the two
optimizations mentioned above.

The Table 5.1 shows the startup time for Peterson1 with the suppression system.

<table>
<thead>
<tr>
<th>Task</th>
<th>Tgl (sec)</th>
<th>Tsj (sec)</th>
<th>Tbj (sec)</th>
<th>Topt (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB reading</td>
<td>1.134</td>
<td>2.155</td>
<td>3.91</td>
<td>0.244</td>
</tr>
<tr>
<td>Rendering</td>
<td>.756</td>
<td>2.334</td>
<td>1.91</td>
<td>1.619</td>
</tr>
<tr>
<td>Total time</td>
<td>1.89</td>
<td>4.489</td>
<td>5.81</td>
<td>1.863</td>
</tr>
</tbody>
</table>

From this section, research question 2 mentioned in section 1.4 can be answered as
follows.

Q) What techniques can overcome the latency in visualizing the geometry at the start
of the application and how effective is each technique?

A) Techniques to overcome the latency in initial viewing of geometry are preprocessing
and caching the database and progressive rendering. Preprocessing and caching is found to
Figure 5.7 Startup Time (DB reading + Rendering)
be very effective (94% increase in performance for a database of size 11,000) for all types of geometries, whereas progressive rendering is effective (43% increase in performance for a database of size 4,000) for data having graphically intensive geometry objects.

5.2.2 Interactivity and Response Time

For calculating the time for interactivity, changing the states of objects, rotations, etc. are done programmatically to eliminate the variations in the user inputs.

Client-based rendering is used as an optimization to decrease the interactivity time. Since the baseline implementation also used client-based rendering, there are no Tbj values. Thus, only Tgl, Tsj, and Topt curves are obtained.

5.2.2.1 Rotations and Translations

The time for one rotation or translation is measured as (time for 500 rotations + 500 zooms for the entire scene)/1000. These values are in milliseconds. Tsj, Tgl curves in Figure 5.8 show that Java3D is faster for very small data sets and as the data size increases, OpenGL performs better to Java3D. From Topt, Tsj curves, it is observed that the optimized Web-based system using client-based rendering performs almost equivalent to the stand-alone Java3D system. The slight difference (Topt > Tsj) can be due to a different JVM in the browser. Also note that the values are in milliseconds. The optimized Web-based system is 10 milliseconds slower to OpenGL system for every rotation or translation.
It is assumed that the user typically makes around 20 to 50 rotations or translations. Thus the total loss would be around 0.5 seconds and therefore is ignored.

5.2.2.2 Changing Object States

The time for changing the state of one door from open to close is measured as (time for changing the state of 1 door for 100 times)/100. Here also OpenGL is slower to Java3D for small geometries but is faster for large geometries. Again these values are in milliseconds, and therefore the performance loss for Java3D can be neglected. The difference between Tsj and Topt in Figure 5.9 is due to a different JVM used by the browser.

The Table 5.2 shows interactivity and response time for Peterson1 with suppression system.

<table>
<thead>
<tr>
<th>Task</th>
<th>Tgl (ms)</th>
<th>Tsj (ms)</th>
<th>Topt (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rotation or Translation</td>
<td>6.57</td>
<td>11.22</td>
<td>12.27</td>
</tr>
<tr>
<td>State Change</td>
<td>6.56</td>
<td>9.84</td>
<td>8.69</td>
</tr>
</tbody>
</table>

The problem with the Java’s timer, as mentioned before, might have introduced some amount of error in the values. It is not clear if the observed difference of 5 to 10 microseconds is due to Java3D or due to the timer inaccuracy.

From this section, research question 3 mentioned in section 1.4 can be answered as follows.
Figure 5.8 Time for One Rotation or Translation
Figure 5.9 Time for Changing the State of One Object (Door)
Q) What techniques allow high interactivity with the visualization at the client side?

A) Client-based rendering is very effective in achieving high client side interactivity with the visualization.

5.2.3 Simulation Visualization Time

Simulation visualization time is the time taken for visualizing a simulation run. The times for replay mode and create mode are measured separately to understand the behavior better in both the modes. The data used was Peterson1 along with the fire suppression system.

Optimizations used to increase the performance are:

1. Skipping the simulation steps
2. Using byte arrays stored in a circular buffer instead of Java piped streams

5.2.3.1 Creating Simulations

For creating simulations, the input file is generated on the grid resource by the server. The simulation model is then executed on the grid resource and the output data is sent to the client as well as stored as a file on the grid resource for replaying at a later time. Spur machine with Intel Xeon 2.4 GHz dual processor and cache size of 512KB with Redhat Linux 8.0 operating system, located on the ERC network is used as the grid resource for the experiments. The results of running and visualizing a complex simulation are shown in Figure 5.10.
Figure 5.10 Time for Running and Visualizing a Complex Simulation
It is observed that the simulation execution time (Texe) on the grid resource (Spur) is larger than the Texe on the client machine (Fire-PC). Running simulation on the grid resource is slower due to two reasons. First, Spur machine is slower when compared to the Fire-PC machine because of different hardware. Second, the operating systems for the two machines are different. Spur has Linux while Fire-PC has Windows XP. Different operating systems give different performances because the simulation model used is a windows compiled executable (.exe). To run this executable on Linux, a Windows emulator is used which makes it slower. To find out the performance loss due to the emulator, a simulation was run on a laptop that is bootable in both Windows and Linux operating systems. The results as shown in Figure 5.11 indicate that there is a performance loss of 14.43% due to running the executable in a Windows emulator. Another issue encountered with using emulator is that it cannot be executed using Globus. Globus cannot get the home environment needed for the emulator to work. Therefore the server was used as a grid-resource to execute the simulations. The simulation network model is a third-party deliverable provided by Hughes Associates, Inc. Linux version of the model could not be obtained by the time the experiments were conducted. Windows machines cannot be used as a grid resource as OGSA (Open Grid Services Architecture) that provides grid technology software tools is implemented only on Linux or Unix.

Another point to note is that the simulation model is not parallelizable and therefore runs only on one grid node. Typically, the high-performance cluster, as a whole, has large
Figure 5.11 Performance of Simulation Executable on Different Operating Systems
computing power but any single node on the cluster has similar or even lesser configuration than a modern PC.

In the conducted experiments, the client machine (Fire-PC) was found to be faster than the grid node (Spur). No optimizations in the visualization would help as this falls under the case 3 described in the section 4.4.3. If a Linux version of simulation model is available which can also be run in parallel, then the ViSGrEn should perform much better than the stand-alone system. Though performance gain was not obtained for visualizing real-time simulations, there are other advantages of having a grid-based system. A user can simultaneously run different simulation jobs on different grid-resources and visualize the results at the same time. Running many simulations simultaneously is not possible on a stand-alone system. Another advantage is to be able to access and run a secure executable that cannot be distributed to run locally on the user machine.

In order to understand better the performance of the ViSGrEn, replay times of pre-run simulations are compared. In replaying simulations, there is no Texe involved and thus the above mentioned problems of nonparallel Windows executable and poor grid resources do not impose any performance loss.

5.2.3.2 Replaying Pre-run Simulations

By using the byte arrays from the circular buffer and skipping the visualization of simulation steps, the visualization time could be decreased. Skipping the simulation steps is effective for complex simulations but not very effective for simpler simulations where
the output steps are generated at an already lesser frequency. Thus, skipping steps for a simpler simulation results in significant loss of detail. The values Topt1 and Topt2 indicate the optimized Web-based system’s performance using one second and two second skip rate respectively. Figure 5.12 shows that the visualization time of the ViSGrEn is close to that of the OpenGL system with some loss of quality.

The Figure 5.13 shows the visualization time for three different simulations that have same duration (250 seconds) but varying complexities, complex-201 being the most complex of three, complex-202 being moderately complex, and simple-109 being the least complex. As the complexity increases, the simulation model produces steps at a frequent rate. Thus if skipping rate is chosen as one second, the number of steps skipped would be more for a complex simulation resulting in less visualization time. If the rate of skipping is increased to two seconds (Topt2) where steps occurring at every two seconds are shown, then the visualization time for the optimized Web-based system is even less than the stand-alone OpenGL system (Tgl). Different users have different requirements for the quality of visualization and the speed of visualization. A design engineer would be interested in a quality visualization to analyze the design of the structure even if it takes more time. Whereas a rescuer would be interested to have faster results than quality graphics. For this purpose, in ViSGrEn, an option to chose the visualization skip rate is provided to the user. Thus by foregoing the quality of visualization, performance can be improved to the desired level.

By analyzing the results, research question 6 is answered as follows.
Figure 5.12 Optimized Visualization Time for Simulation Pre-runs
Figure 5.13 Effect of Skip Rate on Simulations of Varying Complexities
Q6) Should the quality of visualization be compromised in order to satisfy the requirements?

A) Even though all visualization functionalities that are implemented in OpenGL could be implemented in Java3D, it is clear from sections Create Simulations and Replay Pre-run Simulations that the quality of visualization should be compromised in order to have a real-time Web-based simulation visualization system in the Grid environment. In order to compensate for the performance loss due to Java3D rendering, communication time, and slow execution on the grid resource, some simulation steps are skipped from visualization, resulting in lesser quality visualization.

From all the above experiments, research questions 4 and 5 can be answered as follows.

Q4) What techniques can decrease the communication time between the server and the clients, and between the server and the grid resources?

A) Communication did not pose a significant loss in performance. The clients are expected to have a reasonably fast network (bandwidth > 10Mbps) because the distributed systems are not efficient on slow networks. Sending data between the server and the client as one big chunk improved the performance a little. Due to NSDS streaming, there is a small initial latency to get the first bit of data from the grid resource to the server.

Q5) What is the performance degradation in making the application portable across platforms? (i.e., due to Java and Java3D)?

A) There is a performance loss for making the system platform independent, i.e., due to Java and Java3D. Performance loss due to Java was prominent while reading from a large
database and filling the data structures. This was observed due to the poor performance of the JVM on the server. Optimizing JVMs is outside the scope of this work. Though the performance loss due to Java3D is observed throughout, i.e., in geometry rendering, interactivity, and simulation visualization, it is more prominent in initial rendering of geometry. By using the proposed optimizations, this performance loss was overcome to a large extent. Interactivity is not affected by the poor performance of Java3D because the users cannot enter the data faster than the system can handle.

### 5.3 Validation of Hypothesis

The first hypothesis states that a grid-based simulation system can provide through a Web-browser a real-time visual information that is helpful for emergency response teams. VisGrEn is a Web-based system that visualizes the simulations run on grids in real-time. It is also very interactive and responds immediately to user actions. Even though the performance during simulation runs is poor, it is because of the simulation model and not because of the VisGrEn. The performance of the system can be better understood in the replay mode where there is no time involved for the generation of simulation output. In replay mode, results comparable to that of the stand-alone OpenGL system are achieved, proving the first hypothesis. Facilities for easy modifications of geometry are also provided. Facility to change the entire geometry is currently provided but changing part of geometry like removing some walls or compartments is not yet provided. The architecture can easily be extended to incorporate the modifications of geometries also. Facility for
replaying pre-run simulations is provided. The user can also play with different what-if scenarios by visualizing simulations for different cases. Facility for running multiple simulations concurrently could not be provided due to lack of time but it should be rather easy to implement it on this architecture. This architecture also allows running different simulation models. Thus most of the second hypothesis is also proved.

5.4 Summary of Results

Overall the results are very encouraging. Using the proposed architecture, a performance gain of 68% in the startup time, between 71% to 12% in simulation replays, and very high interactivity at the client side could be achieved. The system is scalable for all data set sizes. As of now, the middle tier does not have a facility to choose the select the grid resources and it uses only one machine as the grid resource making it very inefficient for running simulataneous simulations. Therefore, the system is not scalable for increasing numer of simulation executions.

The technique of storing the database cache as a serialized object at server side and sending it across the network required less communication time and bandwidth. Progressive rendering technique was able to hide the poor performance of Java3D. Skipping simulation steps generated at less than one second frequency and using byte arrays instead of Java piped streams helped in decreasing the visualization time without much quality loss. For real-time simulations, the simulation execution time on the grid-resource was very large due to the non-availability of a suitable simulation model.
Like most stand-alone and client-based visualizations, one of the ViSGrEn’s weaknesses is its high client side requirements, limiting the display size when inadequate RAM and video memory are available. Java3D also limits the performance, but it is tolerable when the advantages of Java are taken into account. The architecture could not be tested for more graphics-intensive application, and therefore the performance for those cases is not known.
6.1 Conclusions

Emergency situations like wars, fire accidents, terrorist attacks etc require immediate action by the user. A real-time system that predicts and visualizes the events to the users and that responds to user actions is useful for reducing the damage caused by the emergency situations. Both simulation and visualization tasks demand high computing power. Even the fastest desktop computer’s power is not sufficient to do both the tasks in real time. Thus high performance computers are needed to run complex simulations and high end graphics systems to visualize those simulations. To run simulations on the grid, an easy way to access and submit jobs on the grid is needed. Also, the system should be available on the Web to geographically distributed users. But the Web and the Grid environments impose several constraints such as network latency, bandwidth, and security etc. making real-time simulation visualizations difficult.

A new architecture is proposed and implemented to overcome the limitations of Web and Grid for real-time visualizations. ViSGrEn (Visualization System for the Grid Environment) is implemented using the proposed architecture. This architecture has three tiers with the client having a Web browser as the front end, web server and application server
as the middle tier, and grid resources as back end. The client is provided with a GUI to set the simulation parameters that are sent to the server. Simulations are run on the grid resources and results are sent back to the client. Accessing grid resources is transparent to the user and is done by the server. The results are visualized at the client side. For having high interactivity at the client side, client based rendering is used.

To test the architecture, Fire-Smoke system was re-implemented using the proposed architecture. This system simulates and visualizes the spreading of fire and smoke in ships. Java and Java3D are used to develop the system. In the evaluation of this system, the constraints imposed for real-time visualizations in the Web and the Grid environments are identified as startup time, poor performance of Java and Java3D, and latency in getting the results from the Grid. To decrease the startup time, offline preprocessing and caching the database is used. Progressive rendering of detailed objects is used to further decrease the startup time. To increase the visualization speed for simulations, simulation steps occurring at frequencies less than a second are skipped from visualization. Also, overlapped communication and visualization and byte arrays stored in a circular buffer instead of Java piped streams are used. There was some performance loss due to Java3D but was overcome to a great extent by using the proposed optimizations. The results showed that a performance gain of 43% in rendering time, 68% in total startup time, and between 71% to 12% in replaying simulations could be achieved when compared to a system that was implemented without using the proposed optimizations. The system also scales linearly with increase in the data size.
The architecture of ViSGrEn satisfies all the requirements. In this research the limitations of the Web and the Grid are identified and overcome using various optimizations. The results are generalized to say that the architecture provides solution to all applications with similar requirements. The results validate the stated hypothesis that it is possible to have a grid-based simulation system providing real-time visual information through a web browser.

6.2 Future Work

Even though the current system is a fully functional prototype, there are some features that are not implemented due to lack of time. One of them is having facility for modifying the geometry like deleting a wall or compartment, opening a door, etc. during a simulation run. Another one is to be able to run multiple simulations concurrently.

Security which is an important issue for the Web and grid systems needs to be addressed. The user requesting the visualization service needs to be authorized. Based on the clearance level of the user, the information he/she accesses can be controlled. For example in the case of Navy Research Laboratories for whom the Fire-Smoke system was developed, the structure of the ship is accessible to every one but simulation visualizations showing how the ship would respond to certain attacks is very classified. Therefore in this case, general users should be allowed to visualize only the geometry, whereas users with appropriate security clearance should be allowed to run and visualize the simulations. Also, even though the access to the grids is secure, the data that is sent over the network is
not secure. It can easily be corrupted or trapped by an unauthorized person. Thus security features need to be implemented.

The middle tier should have the facility to choose the grid resource that has less load. This feature will allow the system to scale for increasing number of simulation executions. Another very attractive feature is to have server side rendering for graphic intensive applications. Right now only client side rendering is used which demands high-end graphics and reasonably high processing power at the client side. Client side rendering might not work very well for applications involving complex graphics algorithms like ray-tracing, medical data that requires very high precision, etc. Though sending large data was made possible by reducing the transfer time over the network through optimizations, client based rendering is still not feasible for sensitive data that cannot be sent to the clients. In such cases, the visualization could be rendered on the server itself and images could be sent to the clients. By having a facility for server side rendering, the visualization task could also be distributed to HPCs on the grids.

Another nice feature to have is to be able to find the bandwidth and available client resources and serve accordingly. For example if the clients do not have Java3D enabled browser, the rendering could be switched to server side and images could be sent to the clients. Also, depending upon the bandwidth, the rate of simulation steps to be transferred could be determined.

Collaborative visualization where multiple users can view, interact and steer the running simulations at the same time could also be implemented. Having virtual reality would
be a good feature for using the system as a training tool. The system could also have sensors attached to it to detect the events happening onboard. It could then run several simulations based on the existing situation, notify the concerned personnel and take necessary measures like turning on the fire suppression system to actually put off the fire.
REFERENCES


[25] B. Sura, *Distributed Visualization using Java3D and J2EE Technologies*, directed individual study report, Department of Computer Science and Engineering, Mississippi State University, Starkville, MS, May 2003.

