DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

SIMULATION OF WAVE PROPAGATION IN ANISOTROPIC MEDIA

by
Mustafa KASAP

October, 2008
İZMİR

SIMULATION OF WAVE PROPAGATION IN ANISOTROPIC MEDIA

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Computer Engineering

by
Mustafa KASAP

October, 2008
İZMİR

Ph.D. THESIS EXAMINATION RESULT FORM

We have read the thesis entitled "**SIMULATION OF WAVE PROPAGATION IN ANISOTROPIC MEDIA**" completed by **MUSTAFA KASAP** under supervision of **PROF. DR. TATYANA YAKHNO** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Doctor of Philosophy.

Prof. Dr. Tatyana Yakhno
Supervisor

Prof. Dr. Valery Yakhno	Prof. Dr. Alp Kut
Thesis Committee Member	Thesis Committee Member

Examining Committee Member	Examining Committee Member

Prof. Dr. Cahit HELVACI
Director
Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENTS

First of all, my sincere thanks go to the jury members for their valuable reviews and comments.

In addition, I would like to thank my wife Zerrin, for her love and support throughout the ups and downs of this research period.

Most of all, I would like to thank my family for their support during this long journey. I am indebted for your endless love and support.

Mustafa KASAP

SIMULATION OF WAVE PROPAGATION IN ANISOTROPIC MEDIA

ABSTRACT

Analyzing complex structures such as electromagnetic waves requires interactive visualization techniques for better interpretation. Based on deep mathematical knowledge, model of the structure and the medium is defined with computationally expensive explicit formulas. Without using computer resources, it is difficult to make robust analyses over the model. In such cases using computers are necessary for rapid and reliable data computation and visualization. Moreover, computers are necessary for disseminating the resulting information to other researchers. To fulfill these requirements, interdisciplinary research between mathematics and computer science is needed.

Considering the above requirements, in this thesis we studied the simulation of wave propagation in anisotropic media. Firstly the explicit formulas are constructed as a solution of the problem. Secondly appropriate parallel computation and visualization technique is implemented. For this purpose we used graphic card processing unit that is capable of executing hundreds of instructions in milliseconds. Using this approach makes it possible to access the computation result directly in the graphic card. By this way we eliminate the data transmission between main memory and the graphic card memory. Immediately after computation, resulting data in the graphic card's memory is directly visualized. Finally we developed a web based prototype platform that makes it possible to share the experiment results with other scientists.

Keywords: Electromagnetic wave, complex structures, simulation, parallel computation, visualization.

EŞYÖNSÜZ ORTAMDA DALGA YAYILIMININ SİMÜLASYONU

ÖZ

Elektromanyetik dalgalar gibi karmaşık yapıların analizi daha iyi yorumlanmak için etkileşimli görüntüleme teknikleri gerektirir. Derin matematik bilgisi temel alınarak, yapı modeli ve ortamı hesaplanması pahalı açık formüller ile tanımalanır. Bilgisayar kaynakları kullanılmadan model üzerinde güvenilir analizler yapmak zordur. Benzeri durumlarda güvenilir ve hızlı veri hesaplaması ve görüntülenmesi için bilgisayar kullanımı gereklidir. Dahası, bilgisayarlar sonuçlanan bilginin diğer araştırmacılara yayımı içinde gereklidir. Bu gereksinimleri karşılamak için matematik ve bilgisayar bilimleri arasında disiplinler arası araştırma gerekir.

Yukarıdaki gereksinimler düşünülerek, bu tezde eşyönsüz ortamda dalga yayılımının simülasyonunu çalıştık. İlk olarak problemin çözümünü sağlayan açık formüller oluşturuldu. İkinci olarak uygun paralel hesaplama ve görüntüleme teknikleri geçekleştirildi. Bu amaçla milisaniyeler içinde yüzlerce işlem yapabilen grafik kartı işlemci birimi kullanıldı. Bu yaklaşımla hesaplama sonucuna grafik kartı üzerinden direkt erişim mümkün oldu. Bu yolla ana hafıza ile grafik kartı hafızası arasında veri taşınmasını ortadan kaldırdık. Hesaplamanın hemen ardından grafik kartı hafızasında yer alan sonuç veri direkt olarak görüntülendi. Son olarak deney sonuçlarının diğer bilimadamları ile paylaşımına olanak kılan web tabanlı prototip platform geliştirdik.

Anahtar Kelimeler: Elektromanyetik dalga, karmaşık yapılar, simülasyon, paralel hesaplama, görüntüleme.

CONTENTS

Page

THESIS EXAMINATION RESULT FORM ii
ACKNOWLEDGEMENTS iii
ABSTRACT iv
ÖZ v

CHAPTER ONE - INTRODUCTION 1

1.1 Introduction 1
1.2 Overview of Parallel Computation Architecture 2
1.3 The Goal of the Thesis 4
1.4 Methods and Tools 5
1.5 The Structure of This Thesis 7

CHAPTER TWO - ELECTROMAGNETIC WAVE PROPAGATION 8

2.1 Introduction 8
2.2 Maxwell's System for Electromagnetic Waves 9
    2.2.1 Finding the Solution 10
2.3 Exact Solution of the Electric Wave Propagation Equation 11

CHAPTER THREE - VOLUME VISUALIZATION 14

3.1 Introduction 14
3.2 Volume Data Structure for Visualization 14
3.3 Raw Volume Data Structure After Calculation 16
3.4 3D Visualization Basics 22
    3.4.1 Slice Plane Technique 22
    3.4.2 Iso-surface Technique 24
    3.4.3 Data Stream Line, Cone Plot and Arrow Plot Techniques 24
3.5 Texture Based Visualization 24

3.6 Image Filtering for Visualization 27
3.7 Volume Rendering Theory 29
  3.7.1 Visualization af a Particle, a Pixel in a Volume 30
  3.7.2 Ray-Casting 31
  3.7.3 Blending the Color 32
  3.7.4 Shear-Warp Problem 33
3.8 Graphics Hardware for Visualization 34
  3.8.1 Graphic Pipeline 34
  3.8.2 Geometry Processing 35
  3.8.3 Rasterization 35
  3.8.4 Fragment Processing 36
  3.8.5 Programmable Graphic Processing Units 36

CHAPTER FOUR - REAL-TIME WAVE PROPAGATION SIMULATION .. 39

4.1 Introduction and Related Work 39
4.2 Memory Allocation 40
4.3 Electromagnetic Wave Propagation Basics 44
  4.3.1 Simulation of Wave Propagation 45
4.4 Fast Fourier Transformations 48
4.5 GPU Based Computation 50
  4.5.1 CUDA Computing Architecture 51
  4.5.2 FFT, BLAS and Built-in Function Libraries 53
4.6 Volume Visualization 53
  4.6.1 Pixel Buffer Object (PBO) 54
  4.6.2 CUDA PBO Interaction 55
4.7 Accelerated Simulation of Electromagnetic Wave 56
4.8 CUDA Limitations and Possible Solutions 58
4.9 Implementation 58

CHAPTER FIVE - WEB BASED EXPERIMENT LIBRARY 60

5.1 Introduction 60
5.2 Resource Sharing 60
5.3 Shared Content 63
5.4 System Functionality 65
5.5 Mathematical Symbol Representation 70
5.6 File Compression and Extraction 71
5.7 Image Thumbnails 71
5.8 System Security 71
5.9 Video Streaming 72
5.10 ActiveX Control for Interactive Visualization 73

CHAPTER SIX - CONCLUSION 75

6.1 Summary and Contributions 75
6.2 Future Works 76

APPENDIX A - MATLAB CODE TO SOLVE SYMBOLIC EQUATIONS..... 82

A.1 Positive Definite Function 82
A.2 Test Function for Inverse Square Root of Epsilon 82
A.3 MATLAB Code to Find Explicit Equation for E 83

APPENDIX B - 3D VISUALIZATION OF PROPAGATION 87

APPENDIX C - SAMPLE CUDA COMPUTATION 92

C.1 Matrix Multiplication by Sub-blocks 92
C.2 CUDA Based Fast Matrix Multiplication 94

CHAPTER ONE
INTRODUCTION

1.1 Introduction

Analyzing the structure of electromagnetic waves during it is propagating inside the anisotropic crystals is active research field. Outputs of this research field are subject to many application areas from engineering to medicine (Cohen, Heikkola, Joly, & Neittaanmaki, 2003). Main idea about analyzing the propagation of the electromagnetic wave is to understand the internal structure of the materials where the wave is emitted. Another reason is to simulate new materials that reflect the desired behavior under several predefined conditions. Also it is known that the electromagnetic waves coming from different sources propagate in the same environment and their influence on materials especially the living ones are unpredictable. Because of its importance, electromagnetic wave theory is developed to simulate any desired system for further analyzes.

Mathematical model of wave propagation consists of several constants that specify material and environment properties. Since the complexity of the model is extremely increasing for most of the material types, it is impossible to solve their equations expressed in the mathematical model. But the solutions of the model for simple materials are represented with explicit formulas to apply numerical computation. Desire for computer aided simulation to visualize the wave propagation also requires explicit equations. It is almost impossible to find an explicit equation with the traditional mathematical approaches. Symbolic computation method is limited in some cases because of its complexity and mostly it is impossible to apply “by hand” computation which produces potential errors. On the other hand numeric computation method is simple but requires huge amount of computation steps that is impossible to compute by hand.

Main solution for simulating the wave propagation according to a mathematical model is to use computer programs which are fast and reliable but limited to its hardware and software capacity. Popularity of computers in scientific computation and limited computation power of desktop computers let several new approaches developed for better performance. Mainly parallel execution algorithms and parallel computation frameworks are developed for resource sharing. To be able to use those algorithms, it is necessary have more then a computer interconnected by an appropriate network infrastructure. Such approaches provide hundreds and thousands time better performance but accessibility is limited because of the hardware cost. Recently developments in computer industry make it possible to use parallel computation feature with an ordinary desktop computer. By this way, previously impossible or expensive computation methods become applicable and new research fields are opened.

1.2 Overview of Parallel Computation Architecture

Efficient processing capabilities of desktop computers are not good enough for scientific computation. Executions of generic applications are aimed during the architectural design of the desktop computer’s processing unit. Modern desktop computers have multi-core processors ranging from 2 to 4 with the capability of processing multiple data at a time. Those processors are general purpose and the amount of the processed data is limited with the number of processors. Processing massive data even with 4 processors doesn’t have an acceptable performance. So generally either distributed computation techniques or powerful computers with specific architectures are used. Main disadvantages of such systems are the cost and the accessibility.

Exponentially growing game industry brings several new technologies that open new eras in scientific computation fields. Desire for visual realism in computer games has increasing the performance of graphic hardware. To succeed visual realism, fast computation technology is developed to process every pixel of each frame in an animation. Aim is to apply complex lighting equations over the model surface to produce real-like representation. The idea to process thousands of pixels at a time is similar to processing massive data at a time. This also requires an improvement in the processor number and directly accessible fast memory modules. Since the price of generic processing unit and general purpose memory modules are quite expensive for a desktop computer, graphics industry produced a new technology to overcome those problems. Many processor but cheaper, so they are not generic and can execute limited number of instructions. Limited size memory modules but have very fast response time for direct accessibility by many processors at a time. This specifications result in the birth of graphical processing units with special graphic card architecture. New generation graphic cards have dedicated fast memory modules and number of simple processing units that are many times faster than generic CPUs (Central Processing Unit).

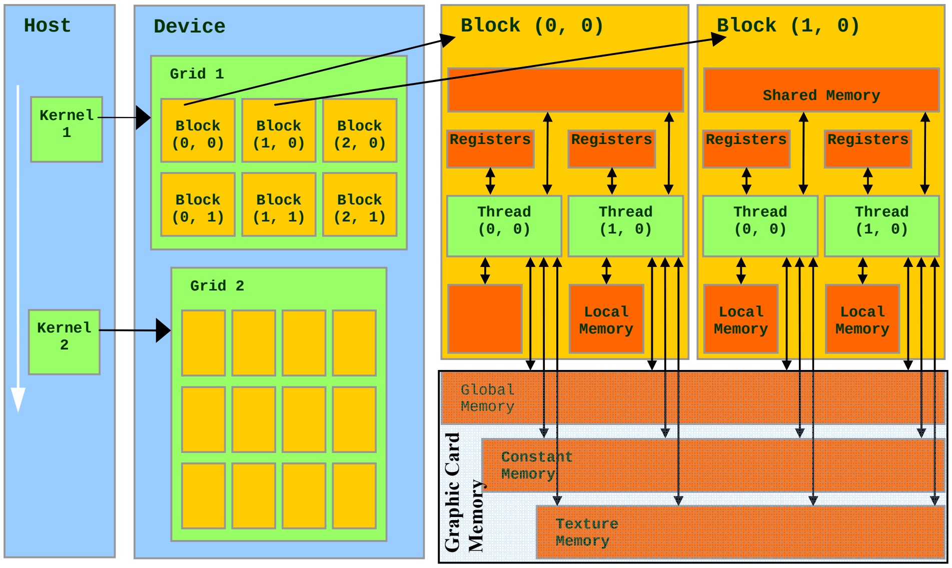


Figure 1.1 CUDA processing architecture.

Figure 1.1 represents the general computation architecture of the CUDA (Cuda, 2008), a scalable parallel programming model. In this schema, computer's mainboard is named as host, and the graphic card is named as device. Kernels are the CUDA programs to be executed. Massive data is divided into blocks and each block is divided into threads. Maximum number of blocks and threads are dependent on the graphic cards' model and so the capability. Each instruction within the kernel (CUDA program or C++ like code) is executed at the same time over blocks of threads. During the execution, each thread has different type of accessibility on specific regions of the graphic card memory. In the appendix of the thesis there is a matrix multiplication example demonstrated by using the CUDA features. Also in chapter 4, details of the CUDA architecture that are used in wave propagation simulation is also explained.

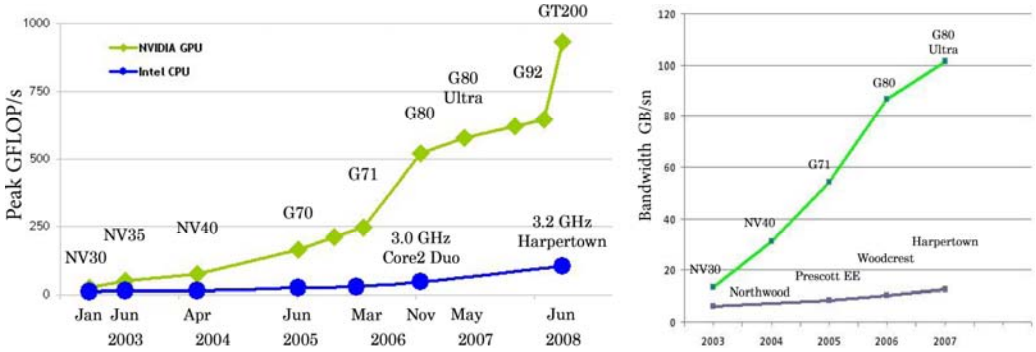


Figure 1.2 CPU and GPU comparision (Nvidia, 2008).

In Figure 1.2, two main performance issues are illustrated for comparison reasons. Figure on the left shows the GFlops (Giga floating point operations per second) comparision between an Intel CPU and Nvidia GPU (Graphics processing unit) over time. Figure on the right shows the memory bandwidth comparision which is important for efficient data transfer. Acronyms such as NV, G and GT are representing the generation of the GPU. Names Harpertown etc. are the names of CPU models. Alone this figure is enough to show the performance improvements of an ordinary desktop graphic card.

1.3 The Goal of the Thesis

Currently there is no single application on the market that is capable of solving complex mathematical equation and performs real-time visualization of massive data at the same time. One of the well known applications Matlab (Matlab, 2008) has such capability but the computation performance is not satisfactory for real-time simulation. In our previous work (Kasap, 2003; Yakhno, 2003), we succeed simulating the wave propagation in an isotropic homogeneous medium by using Matlab application. We solve the Cauchy problem and found the explicit formulas for the solutions of electromagnetic wave propagation. One of the shortcomings of the previously developed system was the computation time for single material simulation. It took hours or days to compute a numerical solution of wave propagation at a specific time slice. Another shortcoming is the performance of the visualization stage. It is almost impossible to visualize 3D representation of the propagation with a user interaction like rotating the model during the animation. First picture in Figure 1.3 (left) is a sample output of Matlab based simulation which is not possible to animate or transform for further analyses. Second picture (right) is a snapshot of 3D real-time animation by using the recent method.

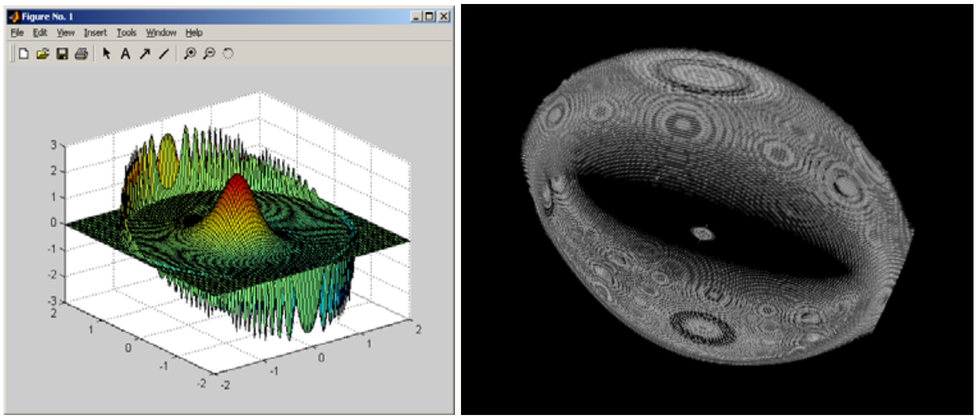


Figure 1.3 Simulating the propagation of an electric field component at time t.

The goal of the thesis is to overcome such problems and simulate the propagation of electromagnetic wave within different material types. Solution of the equation for a specific environment, numerical computation and animated visualization are performed in real-time which make it novel according to our knowledge.

1.4 Methods and Tools

In this thesis we have used mainly three types of programming libraries. First set of libraries are provided by Matlab for extendibility. Matlab provides a development environment where the user can write mathematical equations that will be solved by the computation engine. This independent environment has an input and output window where the result of the computation is streamed or the user defined functions are entered. With this functionality it is not possible to integrate Matlab into another environment where you can programmatically input equations and get results as an input to another program. To overcome this problem, we used Matlab compiler where you can write Matlab functions and compile them as C++ libraries that can be linked to another C++ application. Matlab compiler generates a wrapper file and corresponding header file which makes it possible to call the user defined functions written in Matlab specific language. For parallel computation and fast graphical drawing purposes, we used CUDA graphic processing unit programming libraries. And finally for the visualization purposes, we prefer OpenGL graphic library which provides user input handlers for model transformations and high level graphic programming library interface. General overview of this framework is represented in Figure 1.4.

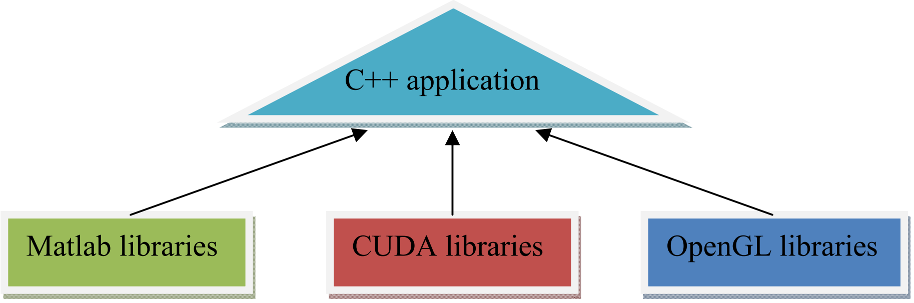


Figure 1.4 Main components of the simulation framework.

Main difference between this and our previous work is the CUDA library that we used for real-time simulation. Recent developments in computer graphics hardware now makes it possible to process thousands of data at the same time. Mainly this feature is aimed to be used for rendering graphic primitives pixel by pixel. Since graphic applications like games have animated scenes with around 25 frames per second, applications must process every pixel in every frame of the animation. Though recently developed graphic cards have the programmability property for fast visualization performance. There exists a support of native low and high level mathematical functions in CUDA enabled new graphic cards. Mostly the low level trigonometric functions and high level Fourier transformation functions of these features are used in our case. Another benefit of those graphic cards is in-place computation support. Since the computations are handled on the graphic card, results are directly written on the graphics card memory. With this feature there is no need to make data transfer between main board and the graphic card. Also native support for Fourier computations makes the best use of memory locations. On the other hand OpenGL library provides the necessary high level graphical programming interface for the visualization stage. CUDA has the interoperability capability with the

OpenGL libraries. It is possible to apply transformation operators on the CUDA generated data values by means of OpenGL library commands.

1.5 The Structure of This Thesis

Thesis consists of six main chapters. First chapter is dedicated for the general description of the problem that is solved. In the second chapter brief description of Maxwell theories are given and explicit formulas for electromagnetic wave propagation introduced. In the third chapter visualization techniques are presented. Recent techniques that are used for volume data visualization is mentioned. In the forth chapter, the real-time system that we developed to simulate the wave propagation is described. The fifth chapter is dedicated for web based simulation library that we developed for publishing the experiment results. In the last chapter analyses of the developed system and future works mentioned. Appendix contains several code snippets used in the developed simulation system. Also from the developed application, several screenshots of the wave propagation simulation is illustrated.

CHAPTER TWO
ELECTROMAGNETIC WAVE PROPAGATION

2.1 Introduction

In this chapter, base of electromagnetic wave propagation in anisotropic dielectrics are studied. Direction of the wave propagation shows different behaviors in anisotropic dielectrics. On the other hand the behavior is linear in isotropic dielectrics. Generally the explicit formulas for fundamental solutions of the Cauchy problem for electromagnetodynamic system are represented. Specifically the theory developed by Maxwell and Hertz is used to solve the problem and represent the electromagnetic waves by means of two vector fields E and H, the electric and magnetic fields. Formulation of the explicit formula differs according to the type of crystalline material. It is known from crystallography that in crystalline materials the constituent atoms are arranged in a regular repeating configuration. By analyzing the smallest unit of material it is possible to create the three dimensional replication. Again according to crystallography science the smallest unit of a material have one of seven basic shapes, such as cubic, hexagonal, tetragonal, trigonal, orthorhombic, monoclinic, triclinic. These symmetry properties tell how the cell can be reflected, rotated and inverted to produce the same special arrangements of atoms. For linear homogeneous non-dispersive anisotropic dielectrics the electric flux density D and the electric field E have the following connection $D = \varepsilon E$. The relation of the crystallographic structures of anisotropic dielectrics with the structure of their dielectric permittivity tensor E is presented by Table 1 (Dienlesaint, 1980).

Details of the research results that we mentioned in this section are also published in a journal (V.G. Yakhno, T.M. Yakhno, Kasap, 2006).

Table 2.1 Crystal systems and dielectric permittivity tensor structures. (diag = diagonal matrix)

Crystallographic structure of dielectrics	*Dielectric tensor ε structures*
Cubic	$\varepsilon = diag(\varepsilon_{11}, \varepsilon_{11}, \varepsilon_{11}), \quad \varepsilon_{11} > 0$
Hexagonal, tetragonal and trigonal	$\varepsilon = diag(\varepsilon_{11}, \varepsilon_{11}, \varepsilon_{33}), \quad \varepsilon_{11} > 0,\ \varepsilon_{33} > 0$
Orthorhombic	$\varepsilon = diag(\varepsilon_{11}, \varepsilon_{22}, \varepsilon_{33}), \quad \varepsilon_{11} > 0, \varepsilon_{22} > 0, \varepsilon_{33} > 0$
Monoclinic	$\varepsilon = \begin{pmatrix} \varepsilon_{11} & \varepsilon_{12} & 0 \\ \varepsilon_{21} & \varepsilon_{22} & 0 \\ 0 & 0 & \varepsilon_{33} \end{pmatrix}, \quad \varepsilon_{11} > 0, \varepsilon_{22} > 0, \varepsilon_{11}\varepsilon_{22} - \varepsilon_{12}{}^{2} > 0$
Triclinic	$\varepsilon = \begin{pmatrix} \varepsilon_{11} & \varepsilon_{12} & \varepsilon_{13} \\ \varepsilon_{21} & \varepsilon_{22} & \varepsilon_{23} \\ \varepsilon_{31} & \varepsilon_{32} & \varepsilon_{33} \end{pmatrix}$, ε is symmetric positive definite.

2.2 Maxwell's System for Electromagnetic Waves

Propagation of electromagnetic wave within an anisotropic homogeneous medium is described by Maxwell's system. Two vector fields, namely the electric (E) and magnetic (H) fields are used to build the system. The properties of the anisotropic homogeneous medium within this system are given by symmetrical matrices ε and μ, where ε and μ are positive definite and they are used to define magnetic permeability and dielectric permittivity. Maxwell's system is defined by the following equations:

$$curl_x H = \varepsilon \frac{\partial E}{\partial t} + j, \tag{2.1}$$

$$curl_x E = -\mu \frac{\partial H}{\partial t}, \tag{2.2}$$

$$div_x(\mu H) = 0, \tag{2.3}$$

$$div_x(\varepsilon E) = \rho, \tag{2.4}$$

Where t is time variable from $\Re$, ρ is the density of electric charges, $E = (E_1, E_2, E_3)$, $H = (H_1, H_2, H_3)$ are electric and magnetic fields, $E_k = E_k(x,t)$, $H_k = H_k(x,t)$, $k = 1,2,3; j = (j_1, j_2, j_3)$ is the density of the electric current, $j_k = j_k(x,t), k = 1,2,3$. The operator $curl_x$ and div_x are defined by

$$curl_x E = \left(\frac{\partial E_3}{\partial x_2} - \frac{\partial E_2}{\partial x_3},\ \frac{\partial E_1}{\partial x_3} - \frac{\partial E_3}{\partial x_1},\ \frac{\partial E_2}{\partial x_1} - \frac{\partial E_1}{\partial x_2} \right),$$

$$div_x E = \frac{\partial E_1}{\partial x_1} + \frac{\partial E_2}{\partial x_2} + \frac{\partial E_3}{\partial x_3}.$$

The conservation law of charges is given by (Ramo, 1994)

$$\frac{\partial \rho}{\partial \mathrm{t}} + div_x j = 0 \tag{2.5}$$

2.2.1 Finding the Solution

To find a solution for the above equations we assume that $\mu = 1$ and $\varepsilon = (\varepsilon_{ij})_{3\times 3}$ is a symmetric positive definite matrix with constant elements. Moreover the following initial conditions are assumed to be true:

$$E = 0,\ H = 0,\ \rho = 0,\ j = 0 \quad \text{for} \quad t \le 0 \tag{2.6}$$

this condition means that there is no electric charges and currents at the time $t \le 0$; so the electric and the magnetic fields vanish until this time.

Since ε and j are given, the main problem is to find E and H that satisfy the equation 2.1, 2.2 and the condition 2.6. We note that ρ can be defined as a solution of the initial value problem for the ordinary differential equation 2.5 with respect to t , subject to $\rho|_{t \le 0} = 0$. Here $div_x j$ is given.

Differentiating equation 2.1 with respect to t and using 2.2 and 2.6 we find

$$-curl_x curl_x E = \varepsilon \frac{\partial^2 E}{\partial t^2} + \frac{\partial j}{\partial t}, \quad x \in \Re^3, \quad t \in \Re, \tag{2.7}$$

$$E\big|_{t\le 0} = 0 \tag{2.8}$$

Therefore for the solution of the main problem first of all we have to find $E(x,t)$ satisfying 2.7 and 2.8 and after that we can find $H(x,t)$ satisfying 2.2 and the condition $H\big|_{t\le 0} = 0$ is $E(x,t)$ is given.

2.3 Exact Solution of the Electric Wave Propagation Equation

Let $\widetilde{E}(v,t)$, $\widetilde{j}(v,t)$ be the Fourier transform images of the electric field $E(x,t)$ and current $j(x,t)$ with respect to $x = (x_1, x_2, x_3) \in \Re^3$, i.e.

$$\widetilde{E}(v,t) = F_x[E](v,t), \quad \widetilde{j}(v,t) = F_x[j](v,t),$$

where

$$F_x[E](v,t) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} E(x,t) e^{ivx} dx_1 dx_2 dx_3,$$
$$v = (v_1, v_2, v_3), \quad xv = x_1 v_1 + x_2 v_2 + x_3 v_3, \quad i^2 = -1.$$

The problem 2.7 and 2.8 can be written in terms of the Fourier image $\widetilde{E}(v,t)$ as follows:

$$\varepsilon \frac{\partial^2 \widetilde{E}}{\partial t^2} + S(v)\widetilde{E} = -\frac{\partial \widetilde{j}}{\partial t}, \quad t \in \Re^3, \tag{2.9}$$

$$\widetilde{\mathrm{E}}\big|_{\mathrm{t}\le 0} = 0, \quad v \in \Re^3, \tag{2.10}$$

where

$$S(v) = \begin{pmatrix} v_2^2 + v_3^2 & -v_1v_2 & -v_1v_3 \\ -v_1v_2 & v_1^2 + v_3^2 & -v_2v_3 \\ -v_1v_3 & -v_2v_3 & v_1^2 + v_2^2 \end{pmatrix} \tag{2.11}$$

The method of the exact solution consists of several steps. In the first step, using the transfer matrix formalism for the given $S(v)$ and symmetric positive definite matrix ε we construct a non-singular matrix F and a diagonal matrix $D(v)$ with non-negative elements such that

$$T^T(v)\varepsilon T(v) = I, \tag{2.12}$$

$$T^T(v)S(v)T(v) = D(V), \tag{2.13}$$

where I is the identity matrix, $T^T(v)$ is the transposed matrix to $T(v)$.

In the second step we are looking for the solution of 2.7 and 2.8 in the form

$$\widetilde{E}(v,t) = T(v)Y(v,t), \tag{2.14}$$

where the matrix $T(v)$ is constructed in the first step and a vector function $Y(v,t)$ is unknown. Substituting 2.14 into 2.9 and 2.10 we find

$$\varepsilon T \frac{d^2Y}{dt^2} + S(v)T\,Y = -\frac{\partial \widetilde{j}}{\partial t}, \quad t \in \Re, \ v \in \Re^3, \tag{2.15}$$

$$Y(v,t)\Big|_{t\le 0} = 0, \quad v \in \Re^3. \tag{2.16}$$

Multiplying 2.15 by $T^T(v)$ and using 2.12 and 2.13 we have

$$\frac{d^2Y}{dt^2} + D(v)Y = -T^T(v)\frac{\partial \widetilde{j}}{\partial t}, \quad t \in \Re, \quad v \in \Re^3. \tag{2.17}$$

In the third step of the method, using the ordinary differential equations technique, a solution of the initial value problem 2.16 and 2.17 is given by

$$Y(v,t) = \begin{matrix} Y_1(v,t) \\ Y_2(v,t) \\ Y_3(v,t) \end{matrix}, \tag{2.18}$$

where for $t < 0$ the function $Y_n(v,t)$ vanishes and for $t \geq 0$ is defined by

$$Y_n(v,t) = -\int_0^t \cos(\sqrt{d_n(v)}(t-\tau))\left[T^T(v)\tilde{j}(v,\tau)\right]_n d\tau, \text{ if } d_n(v) > 0, \tag{2.19}$$

$$Y_n(v,t) = -\int_0^t \left[T^T(v)\tilde{j}(v,\tau)\right]_n d\tau, \quad \text{if } d_n(v) = 0; \quad n = 1,2,3, \tag{2.20}$$

where $d_n(v),\ n = 1,2,3$ are elements of the matrix $D(v)$; $\left[T^T(v)\tilde{j}(v,\tau)\right]_n$ is the nth component of the vector $T^T(v)\tilde{j}(v,\tau)$.

Using formula 2.14, 2.18-2.20 and given matrices $T(v), T^T(v), D(v)$ found in the first step we find the Fourier image of the electric field $\tilde{E}(v,t)$. In the last step the electric field $E(x,t)$ is determined by the inverse Fourier transform F_v^{-1} of $\tilde{E}(v,t)$, i.e.

$$E(x,t) = F_v^{-1}\left[\tilde{E}\right](x,t) = \frac{1}{(2\pi)^3}\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}\tilde{E}(v,t)\,e^{ivx}dv_1dv_2dv_3.$$

The procedure for finding the magnetic field is the same with the above formulations so it is not repeated in the text. Appendix A contains MATLAB code for finding the Electric field vector. In all steps of the code we execute extra instructions to prove the correctness of the intermediate results (i.e. singularity of resulting matrix).

CHAPTER THREE
VOLUME VISUALIZATION

3.1 Introduction

In this chapter we will discuss about the general volume rendering methods and then focus on the specialized techniques that are used in the thesis. Also these specialized techniques are explored under two different categories. First we will explain the third party tools used for visualization and then we will explain our own tools for optimized visualization techniques.

Traditionally volume data represented with polygonal meshes. Visualization of such surfaces handled with special lightening algorithms. Earlier in computer graphics Phong shading algorithms are used to illuminate a model (Phong, 1973). Currently there exist much more complex BRDF (Bidirectional reflectance distribution function) lightening algorithms for realistic visualization of the models. All those lighting algorithms interact with the surface of the model to generate corresponding illumination effects. In volume rendering, without doubt we need to visualize the interior part of the models with special illumination algorithms.

Generally volume rendering is used for scientific visualization (Levoy, 1988) of multidimensional data with variety of techniques such as magnetic resonance imaging, computed tomography, wave propagation. In recent decades, there are specialized research groups working on the volume rendering subject by both improving the algorithmic techniques and hardware implementations. One example is Semcad (Semcad, 2008) company which is developing special hardware and software for processing and visualizing real-time medical data. They use special hardware devices which can process millions of model vertices per second.

3.2 Volume Data Structure for Visualization

Volume data for visualization is mainly represented with a three dimensional array. Data type of this array depends on the resolution of the volume data. In computer graphics, each element of the volume is called voxel (Kaufman, 1994). Some example data sizes for a voxel are 4 bit, 1 byte, 3 byte, 4 byte etc.

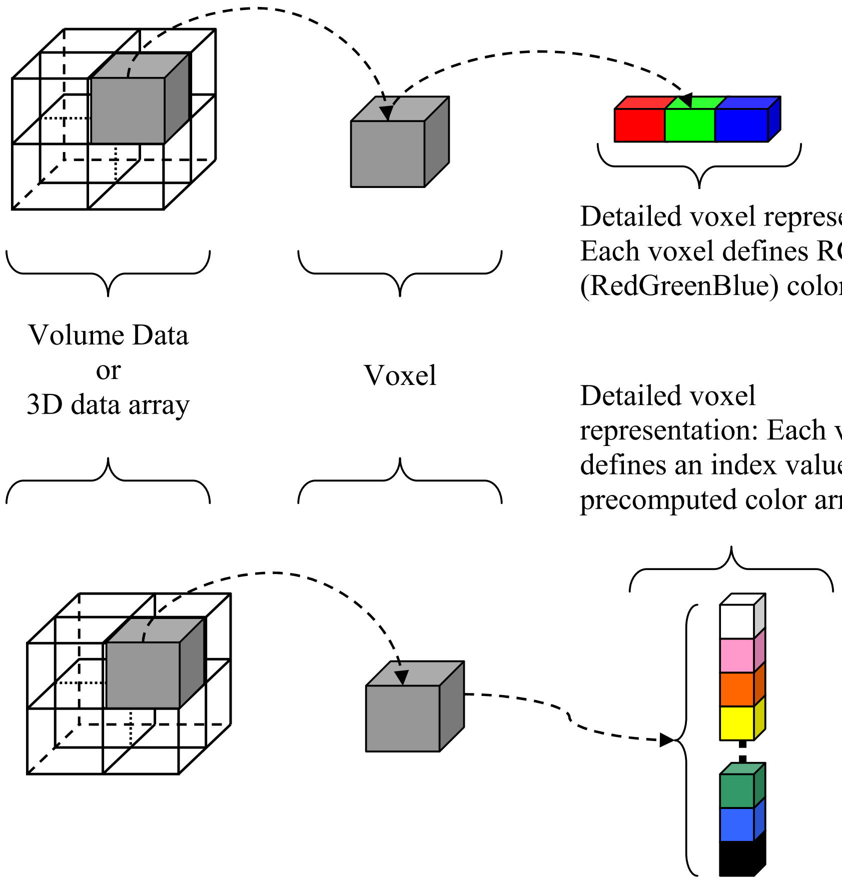


Figure 3.1 Voxel representation.

In Figure 3.1, two different representation type of the voxel are illustrated. In the first example (first row) each voxel defines the RGB components of its color. There exist different color space representations like CMYK, HSV etc. In comparison to others, RGB is the most popular representation type used in graphic cards at hardware level color representation. To define a color in RGB color space, one need to have three data components that corresponds to Red, Green and Blue values. Size of a voxel depends on the sum of sizes of those components. As an example assuming the size of each component is 8 bit (1 byte) in RGB space, and then we can represent $2^8*2^8*2^8$ different color within this space. By using just a single component, it is possible to represent 2^8 different gray colors. In the second example (second row) of the Figure 3.1, each voxel is an integral value referencing to the index of the color of that voxel. In such cases, along with the volume data, indexed color map must be provided. If only 16 different colors are used in data volume then each voxel size would be 4 bit length. With 4 bit data, $2^4 = 16$ different color index can be defined. Number of bits used to represent the color is called color depth. To compare data storage size of these two different approach lets assume that we have the size 10x10x10 of volume data. So this volume data consists of 1000 voxels. In this volume data assume that just 16 different colors are used. For the first approach, assuming that each voxel is used to define 24bit (3 byte) color information then final data storage size will be 3000 byte (~3 Kilobyte). For the second approach, we generate an indexed color map for 16 different colors. Size of this map will be 16 * 24 bit = 48 byte. Apart from the color map size, volume data size will be 1000 * 4 bit = 500 byte. Here 4 bit represents the index of one of the 16 color in the map so total data size will be 500 byte + 48 byte = 548 byte (1/2 Kilobyte). Depending on the number of distinct colors in the volume, one of these approaches can be preferred. There exist lots of other optimized methods like using data compression for volume data storage but the above two method represented just to depict the general idea. In case of the data compression techniques, high level computer graphic card properties are used. Two well known high level graphic programming interfaces are OpenGL (OpenGL, 2008) and DirectX (Directx, 2008). While using these interfaces, compressed data is send to a graphic card for visualization. In this way, limited size graphic card memory can be efficiently used by sacrificing the performance.

3.3 Raw Volume Data Structure After Calculation

In our system, calculation with specific electromagnetic wave propagation parameters results in 3D data array. Each component of this array represents the intensity, pick level of the propagation in the corresponding point of the 3D environment. We used 8 byte double precision floating point data type (Goldberg, 1991) for declaration of each array element. Transformation between intensity values to a color value is handled with special algorithms. This process demonstrated with the following sample MATLAB example. 3D volume data cube consists of 2D frames so to make it simplified, here only 2D frame used to demonstrate the transformation. First we generate a random 2D double array as represented in Figure 3.2.

```
>> Frame2D = magic(4)

Frame2D =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

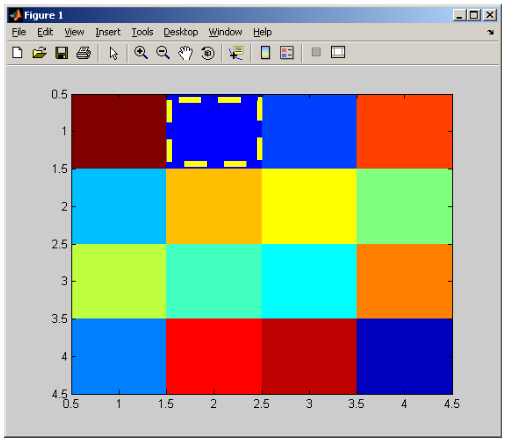
Figure 3.2 Random numbers generated by inline Matlab function.

Resulting value is a 2D array where data type of each array component is double. This array is stored in variable called `Frame2D`. Values of the components of the array vary from 1 to 16. According to this range we generate an indexed color map with 16 different RGB color values.

```
>> map = jet(16)
map =
         0         0    0.7500
         0         0    1.0000
         0    0.2500    1.0000
         0    0.5000    1.0000
         0    0.7500    1.0000
         0    1.0000    1.0000
    0.2500    1.0000    0.7500
    0.5000    1.0000    0.5000
    0.7500    1.0000    0.2500
    1.0000    1.0000         0
    1.0000    0.7500         0
    1.0000    0.5000         0
    1.0000    0.2500         0
    1.0000         0         0
    0.7500         0         0
    0.5000         0         0
```

Figure 3.3 Color map values.

Resulting value stored in a variable called map which is a 16*3, 2 dimensional array. This variable holds 16 different color values with three RGB color components. Visualizing 2D array data stored in the `Frame2D` variable with a specific color map stored in the variable `map` is shown in Figure 3.4.



```
>> image(Frame2D)
>> colormap(map)
```

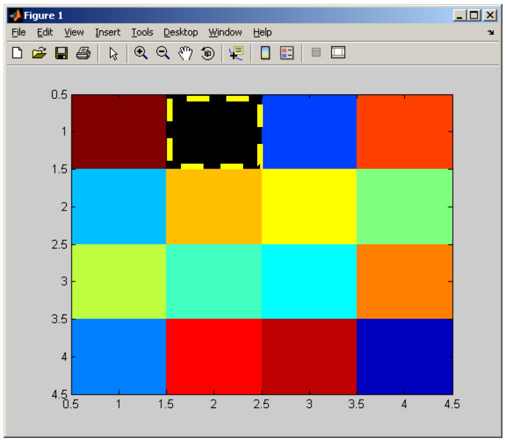
Figure 3.4 2D data visualization.

Value of the second element of the color map is {0, 0, 1.0} where first and second components are 0, and this means red and green contributions are zero but the third component, 1, means full blue contribution. Final color representation of this second element is pure blue. Changing this array element to {0, 0, 0} means that it is black.

```
>> map(2,:) = [0; 0; 0]
map =
         0         0    0.7500
         0         0         0
         0    0.2500    1.0000
         0    0.5000    1.0000
               . . .
    1.0000    0.7500         0
    1.0000    0.5000         0
    1.0000    0.2500         0
    1.0000         0         0
    0.7500         0         0
    0.5000         0         0
```

Figure 3.5 Modified value of color map.

Resulting visualization of `Frame2D` data with the modified color map is illustrated in Figure 3.6.



```
   >> image(Frame2D)
   >> colormap(map)
```

Figure 3.6 Data visualization with modified color map

First row, second column of the `Frame2D` data array is equal to 2. This means that the color value on that position will be black as it is represented in the resulting Figure 3.6.

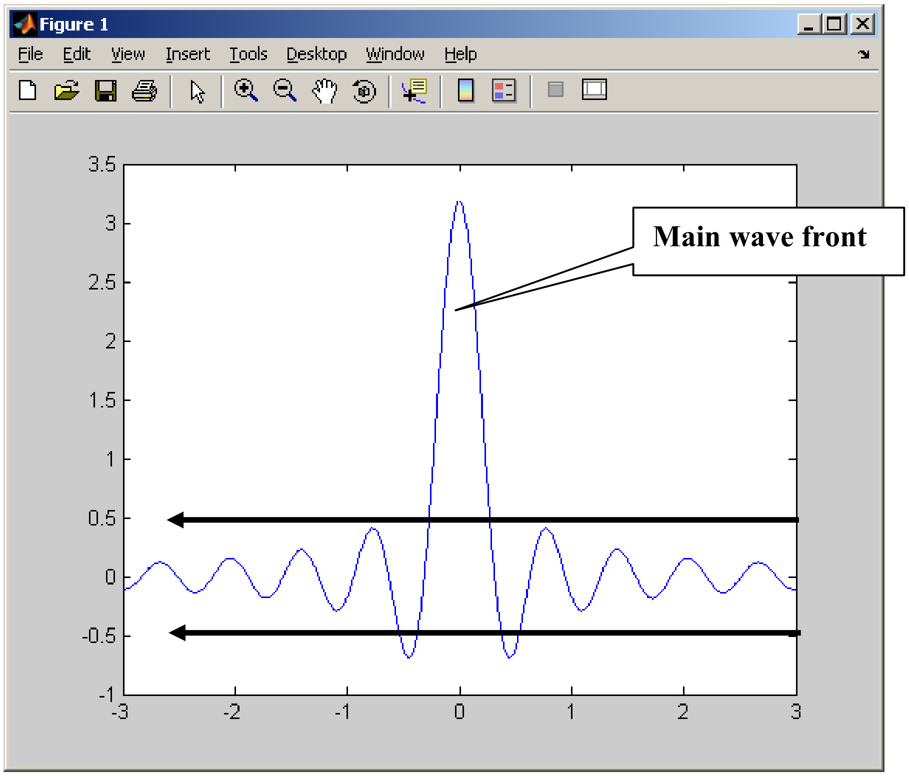
The above example data array is visualized with the following steps:

- Create a color map. Size of the color map may vary according to the number of distinct data value in the data.

- Find minimum and maximum values in data array.

- Linearly map the minimum value to the first indices of the color map and maximum value to the last indices. Other values between minimum and maximum values are linearly mapped to the closest indices.

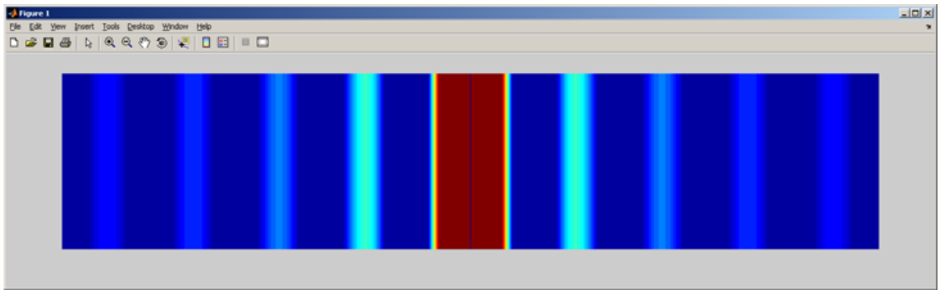
Scaling the indices map is also possible. i.e. instead of linear mapping it is possible to use non-linear mapping to emphasize greater values with more color intensity. This technique is used in our case to emphasize the wave front.



```
>> eps = 0.1;
>> x = -3:0.005:3;
>> y = sin(x / eps) ./ (pi * x);
>> plot(x, y);
```

Figure 3.7 Filtering wave front with color map.

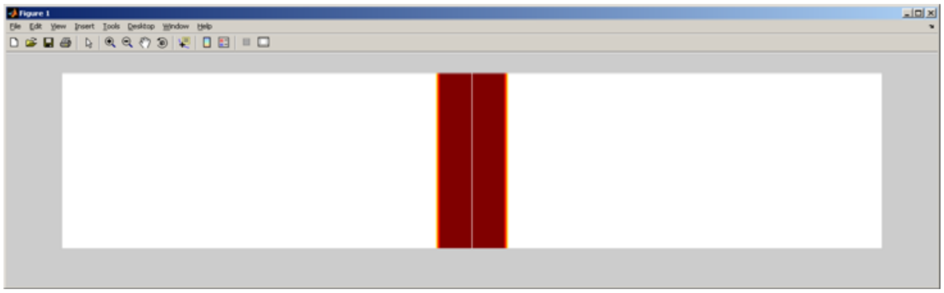
In Figure 3.7, sample wave front represented with some fluctuation. To remove the noisy data ranging between -0.5 to 0.5 we can define a color map where the values between this ranges represented with white color. Other values, here the pick or wave front, will be represented with some other color.



```
eps = 0.1;
x = -3:0.005:3;
y = sin(x / eps) ./ (pi * x);
y2D = repmat(y, [256 1]);
imshow(y2D);

map = jet(32);
colormap(map);
```

Figure 3.8 None filtered data visualization.

Figure 3.8 is the visualization result of the sample wave that is represented in Figure 3.7. Since there is no filtering operator applied, the fluctuations are represented with the color blue and its variations. On the other hand same data is filtered to remove the data between the interval [-0.5, 0.5].



```
eps = 0.1;
x = -3:0.005:3;
y = sin(x / eps) ./ (pi * x);
y2D = repmat(y, [256 1]);
imshow(y2D);

map = jet(32);
map(1:20,:) = ones(20,3);
colormap(map);
```

Figure 3.9 Data filtering through color map.

3.4 3D Visualization Basics

There exist many third party tools (Matlab, 2008; Mathematica, 2008) which support generalized visualization techniques for volume data. We will focus on Matlab application that is used in our examples. We can categorize volume data under two main heading:

- Scalar volume data, where each cell of the volume data contains single value.

- Vector volume data, where each cell of the volume data contains more than one value.

In section 3.2, volume data structure is presented with scalar volume data where each cell holds the amplitude of the wave on the corresponding position. In our case, we have both electric and magnetic waves propagating in space where each wave has three components in a specific point of the space. This in mind, finally we will be dealing with vector volume data visualization.

For simplicity, we will describe application of some common visualization methods by starting from Scalar to Vector data. For scalar data we will demonstrate iso-surfaces and slice planes, for vector data the stream lines, cone plots and arrow plots.

3.4.1 Slice Plane Technique

Slice plane is based on defining three orthogonal planes. Data values on a plane and volume intersection area are projected on the corresponding planes.

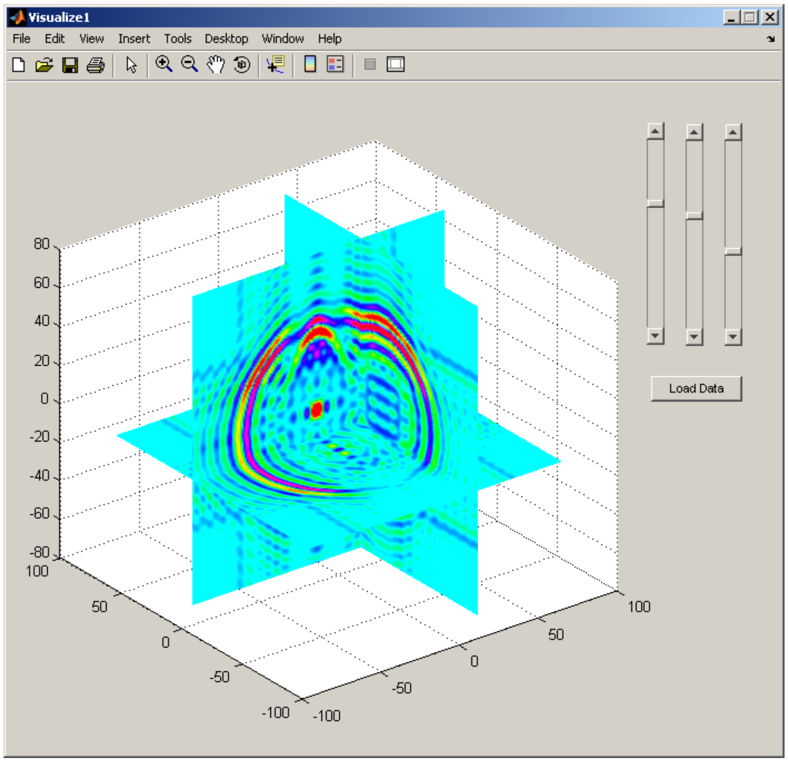


Figure 3.10 Slice plane representation of volume data.

Figure 3.10 represents the visualization of the three dimensional data cube where each cell of the cube holds a floating point value. Each floating point value corresponds to the intensity of the wave propagating from the center of the cube at a specific time value. Taking three orthogonal planes from this cube means extracting specific indices ranges from each dimension of the cube. Corresponding array values are visualized as mentioned in section 1. So the result is a kind of image where the specific orthogonal planes of the volume are projected.

3.4.2 Iso-surface Technique

Iso-surface generation from a data cube requires some preprocessing stages like filtering the data. According to a user specified iso-value, data is filtered and closer values around iso-value are kept. Only the resulting contour values rendered as a shell. Below figure represents sample wave propagation at a specific time value.

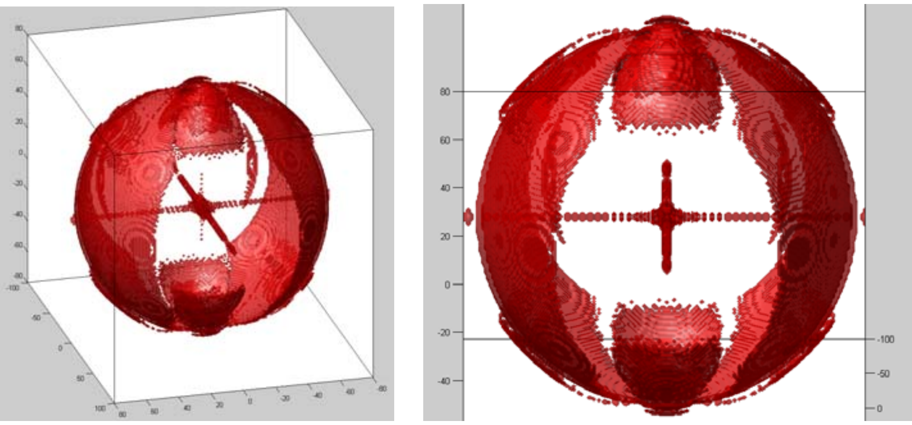


Figure 3.11 Iso-surface representation of volume data.

3.4.3 Data Stream Line, Cone Plot and Arrow Plot Techniques

Data stream lines require special volume data. In contrast with the previous examples, stream line data has vector data with three or more components. Each vector on the corresponding volume position visualized as a continuous ribbon. In cone or arrow plot technique, each vector value is represented with directional cone or arrow in the corresponding volume position.

3.5 Texture Based Visualization

Rendering large amount of data requires special techniques even a special hardware. One of the recent approaches for visualization is to use cluster of computers for parallelized volume rendering (Humphreys et al., 2000). In this approach, it is necessary to combine networked computers and use special parallel programming algorithms for both calculation and visualization. Such kind of implementation requires more than one powerful computer with cost effective constraints. An alternative approach is to use the capability of graphic cards' computation resources. By this way, load balancing between the computers central processing unit (CPU) and the graphic processing unit (GPU) is achieved. Graphic card based computation and visualization techniques will be explained in the following chapter. In this section we will demonstrate texture based volume rendering methods by using CPU power.

Texture volume rendering is based on a chunk of image files. With optimal delta value, image slices are extracted from a volume data. These images are then combined together by adjusting a translucency value to visualize them as a whole volume cube. One of the main researches based on this method is visualization of human body project called Visible Human (Visible Human, 2003). Within this project, acquisition of transverse CT, MR and cryosection images of representative male and female cadavers are generated. Female cadaver sectioned at one third of a millimeter intervals. Each section is a 2D image file where special visualization tools are used to combine these images and visualize them as a single volume. One of the main features of these tools is to let the end-user to explore a specific part of the volume details.

As in Visible Human project, we make calculations to compute the final volume data. Before starting the visualization step, we generated a set of 2D images from the resulting volume data. Those images are combined together with a special application for visualization. In Figure 3.12, some sections from the volume data are represented. Before using these images, preliminary filtering process takes place to have clear visual results in the final volume image. Details of filtering process are described in the next section. In this example, the simple filtering method mentioned in section 3.3 is used.

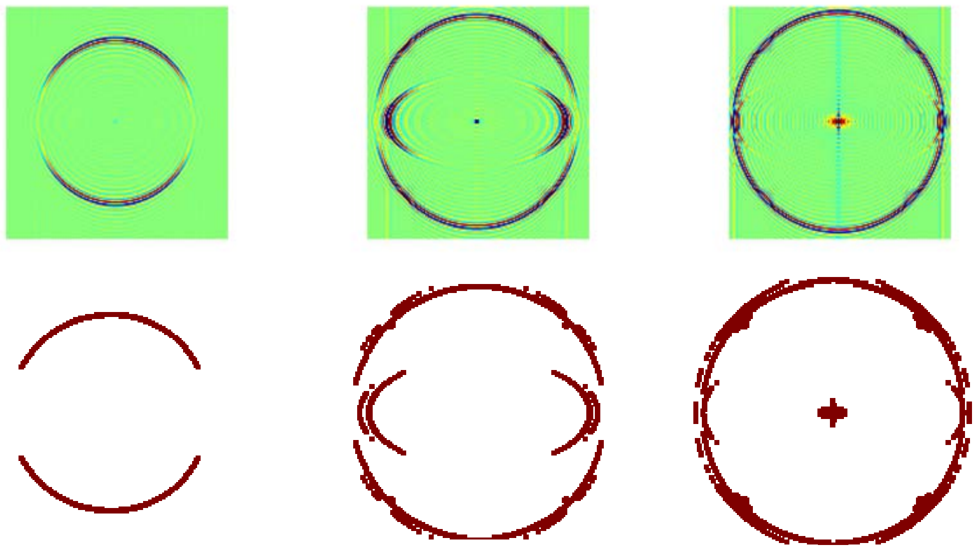


Figure 3.12 Row and filtered images of sections from volume data.

Above images are combined together and mapped on proxy geometry which is simply a box. Final geometry rendered as a 3D object (cube) with volume texture mapping methods. Schematic representation of this process is shown in Figure 3.13.

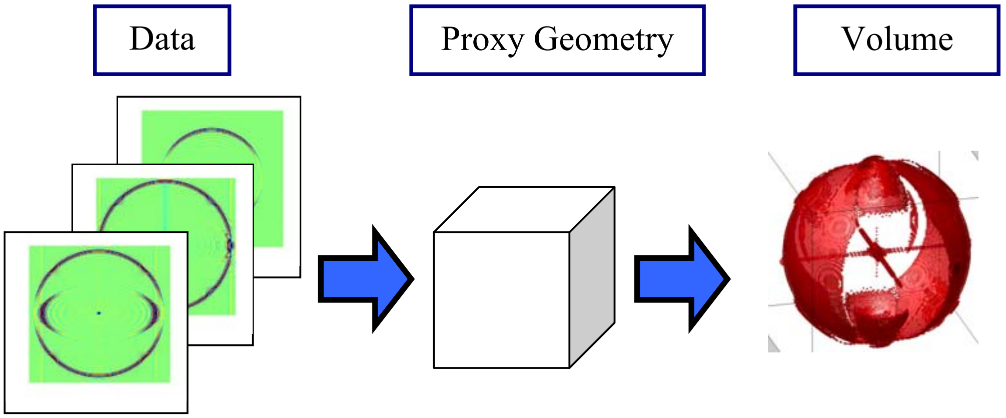


Figure 3.13 Volume rendering pipeline.

Sampling rate and delta value between each image is directly affecting the quality of the visualization. If this value is not small enough, then smooth interpolation between each image is not possible. Also this will lead to an incorrect emission and absorption on the final result.

3.6 Image Filtering for Visualization

Numerical simulation of the magnetic wave data contains fluctuations in the raw texture files. Apart from these fluctuations, nature of the inverse fast Fourier transformation that is used in the calculations also produces some noise. Main wave front overrides these side-effects by its strong intensity and visually it is easy to recognize the wave front. We tried two different filtering types for eliminating this noisy data from the image. First we found the main intensity component according to the image histogram. Except from this intensity value, we assigned zero as a transparency value to the rest of the image pixels. Figure 3.14 represents the elimination of different intensity values.

Another filtering method is canny edge detection algorithm (Canny, 1986). In this method, image is smoothed with Gaussian convolution then first derivative operator is applied to highlight the image. By this way, edges became much more bumped in the recent gradient magnitude image. These bumped pixels are tracked to generate a line on the final output. Second row of the Figure 3.12 represents the filtered versions of images with canny edge detection algorithm.

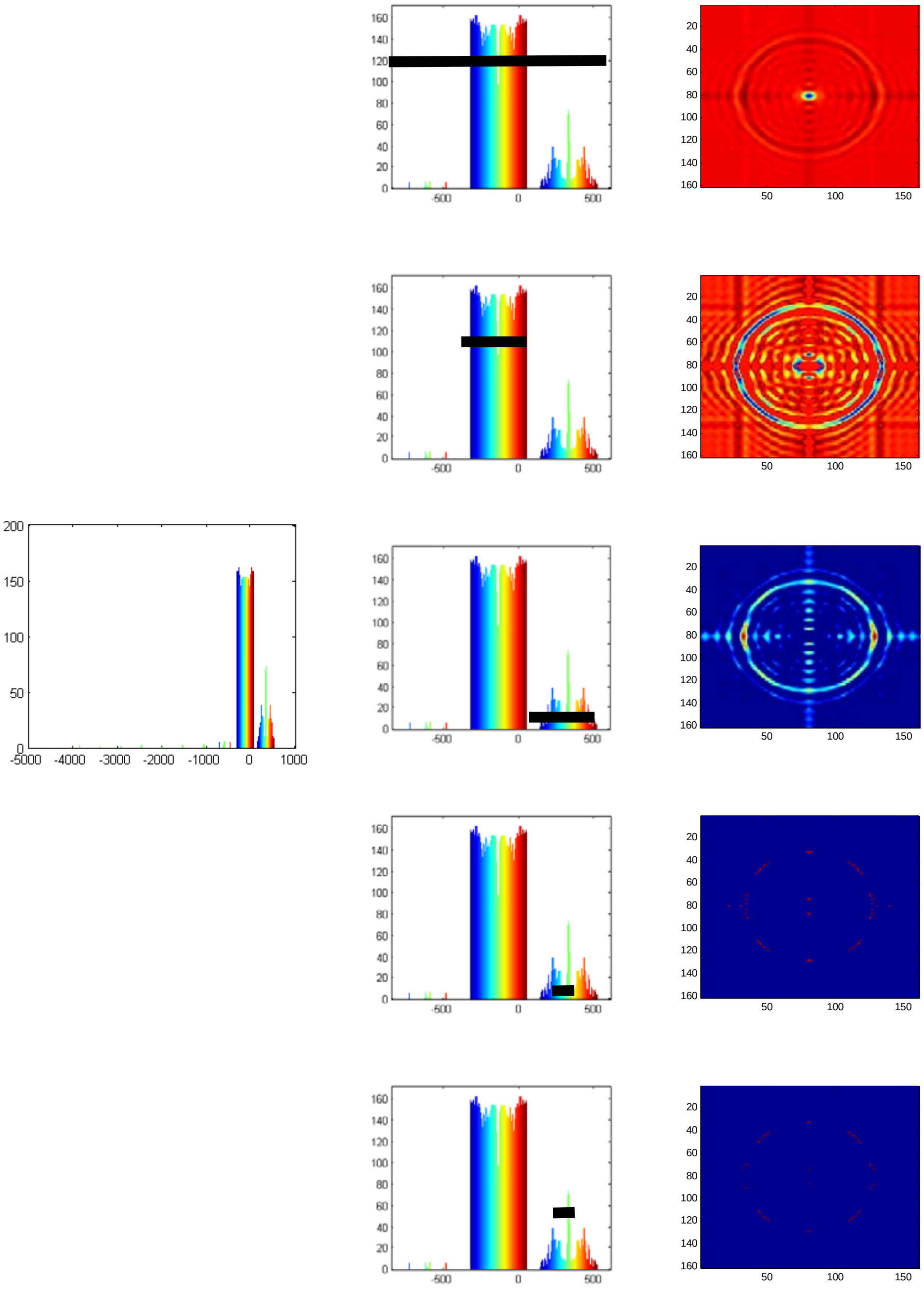


Figure 3.14 Image filtering by histogram.

3.7 Volume Rendering Theory

There are many application areas such as medicine, geology, archeology, material science, biology, computational science, computer science that are using volume rendering. Most of these application areas deal with Computed Tomography (CT) scan data. CT data supposed to be a set of 2D images. Depending on the resolution of the scanned data, 2D image size and color depth are varying. Also number of images in a set is directly related with the number of samples taken from the subject. There exist different sample CT datasets on Internet for research purposes like Visible Human Dataset. In Figure 3.15 set of the 2D images gathered by CT scan and combining these images to generate the 3D leg volume generated by the combination of the images are presented.

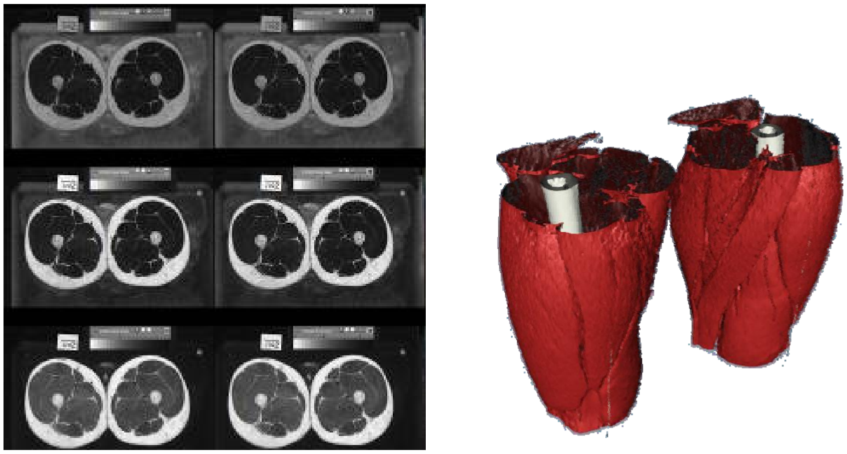


Figure 3.15 Volume representation of human leg from a set of 2D CT images.

One of the volume visualization methods is to simulate data with an optical method approach. Each value in the volume assumed to be particles behaving like optical material that can absorb, reflect, emit, and scatter the light. In his paper, Max (1995) summarizes all these methods for volume rendering.

Absorption only method: Volume is consisting of optical particles which absorb the coming light.

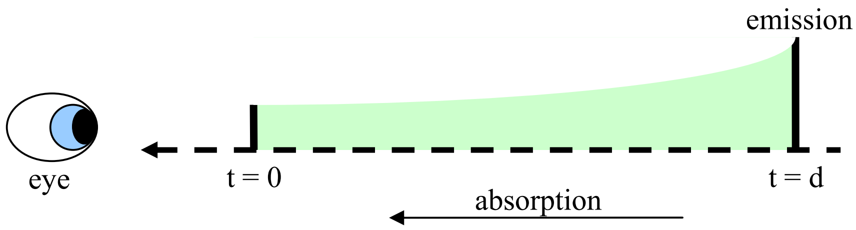
Emission only method: Volume is consisting of optical particles which emit the light but not absorb.

Absorb and Emission method: No scattering and shadowing. It is a combination of the previous two methods.

Apart from the above methods, there exist many other complex methods for volume rendering like self shadowing, scattering, etc.

3.7.1 Visualization af a Particle, a Pixel in a Volume

Volume rendering by means of particle visualization is the basic approach in this field. There exist different approaches for visualizing the optical particle. Common approach is ray-casting method which is supposed to be the basics of numerical method. Main idea is to cast a ray from the eye to the center of the particle in the volume and than integrate the optical properties along the cast to the particle. The result of this integration will be the particle's cumulative intensity. During the integration, some assumptions considered because of the numerical approach to the continuous data. Riemann sum is used as an integration method for such numerical approximation. It is called radiant energy and the model is represented in Figure 3.16.



t : distance
$\vec{x}(t)$: ray cast into the volume
$s(\vec{x}(t))$: scalar position along the ray
$\kappa(s)$: Absorption coefficient
$c(s)$: emissive colors

Figure 3.16 Radiant energy emittion model.

Energy emitted at $t = d$ is absorbed during it comes to the eye where $t = 0$. If the absorption is constant during the distance t, then only the energy, c', emitted from the source will reach the eye.

$$c' = c * e^{-\kappa d}$$

If the absorption coefficient is not constant during the distance from eye to the particle then the formula will be:

$$c' = c * e^{-\int_{0}^{d} \kappa(t)dt} \quad , \tau(0,d) = \int_{t=0}^{d} \kappa(t)dt$$

Here the integral is called the absorption depth τ . Total amount of the energy coming to the eye is not only the energy emitted from the particle but also the other particles apart from the subject. So we will take the integral from 0 to infinity.

$$C = \int_{0}^{\infty} c(t) * e^{-\int_{0}^{t} \kappa(t)dt}$$

3.7.2 Ray-Casting

Ray-casting (Levoy, 1988) is a method of image generation of 3D volume from a specific perspective. For each pixel of the image, one ray is casted into the scene to find the corresponding cumulative energy that is emitted through the ray. Along the equal spaced intervals, generally tri-linear filter used to resample the data where in 3D neighboring 8 pixels is used to sample the center pixel. So the optical depth τ is approximated by Riemann integral:

$$\tau(0,t) \approx \tilde{\tau}(0,t) = \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i * \Delta t)\Delta t$$

Here Δt is the distance between to sampled pixel. By substitution we can rewrite the above formula in multiplication form

$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} e^{-\kappa(i*\Delta t)\Delta t} \qquad (3.1)$$

If we denote opacity with A by means of alpha blending

$$A_i = 1 - e^{-\kappa(i^*\Delta t)\Delta t}$$

Finally the equation becomes

$$e^{-\tilde{\tau}(0,t)} = \prod_{j=0}^{\lfloor t/d \rfloor}(1 - A_j)$$

Similarly emitted energy of the i th ray segment will be

$$C_i = c(i * \Delta t)\Delta t$$

Now we both approximated the emission and the absorption of a single ray that pass through the particle. So the final integral will be

$$\tilde{C} = \sum_{i=0}^{n} C_i \prod_{j=0}^{i-1}(1 - A_i)$$

where n is the number of samples. This final equation can be evaluated from back to front or vice versa. Generally in computer graphics, because of computational complexity, back to front evaluation is preferred.

3.7.3 Blending the Color

Because of its computational simplicity here we will demonstrate back to front evaluation of the color blending approach so index variable i changes from $n-1$ to 0.

$$C_i^{'} = C_i + (1 - A_i)C_{i+1}^{'}$$

If we suppose that the starting condition $C_n^{'} = 0$, iteratively we can calculate the desired value by back to front evaluation.

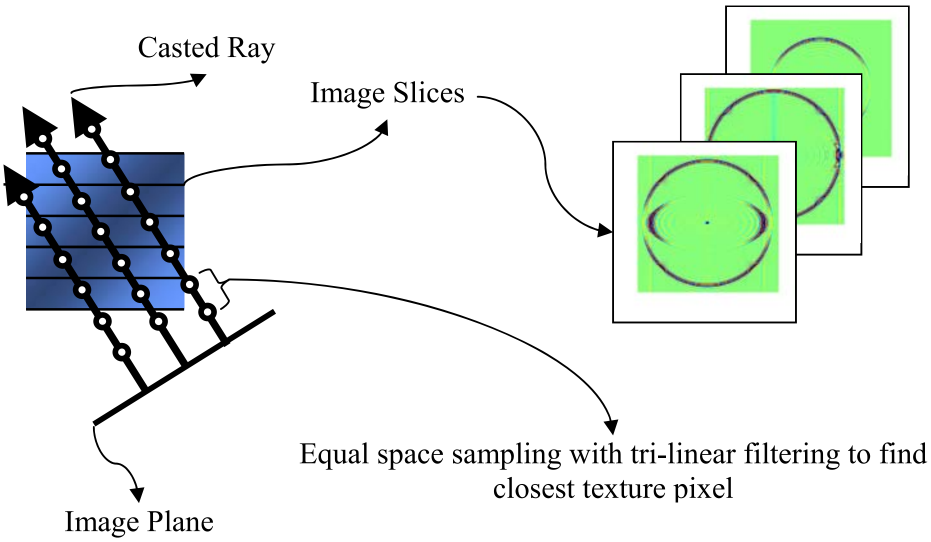

Casted Ray
Image Slices
Equal space sampling with tri-linear filtering to find closest texture pixel
Image Plane

Figure 3.17 Application of ray-casting method for volume rendering.

3.7.4 Shear-Warp Problem

Ray-casting method for volume rendering requires so much computation. In real-time rendering case, for huge data, it is almost impossible to use ray casting with ordinary graphic hardware. One alternative approach which mimics ray-casting method is called Shear-Warp algorithm.

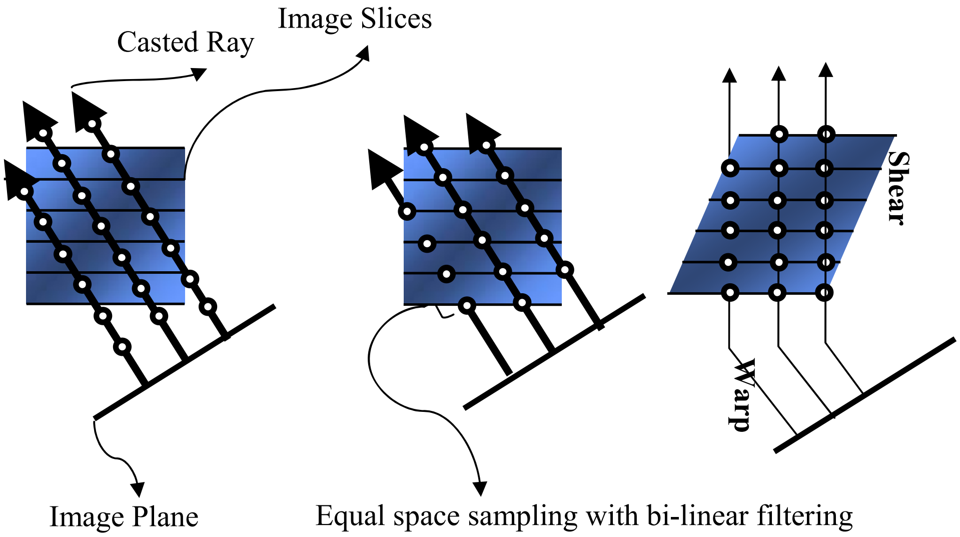


Figure 3.18 Application of shear-warp algorithm with orthogonal projection.

Figure 3.18 illustrates the application of shear-warp algorithm. Sampling distance along the rate is equal to the distance between each image slice. Looking to the volume from different angles, not perpendicularly, requires shearing the volume. After shearing integrated color value is warped and projected on the image plane. These approaches are much faster than ray-casting method. Also the shear-warp method is improved with run-length compression algorithms which are suitable for back to front approximation for color blending.

3.8 Graphics Hardware for Visualization

There have been significant improvements in graphics hardware during the recent years. Initially the trend is focused on fixed graphical pipeline for rendering until 1998s. Then programmable pipelines take stage and finally fully programmable floating point graphic processing units became popular. Recent developments show that there is challenge in achieving higher programmability and parallelism in graphic hardware. Success of such improvements based on producing graphics cards with billions of transistors on a single chip. Because recent graphic cards have such a powerful performance, they became popular for implementing general purpose computation and parallel processing algorithms.

3.8.1 Graphic Pipeline

We can numerate the standard graphic pipeline under three subgroups:

Geometry processing: Transformation operations such as translation, rotation, scaling are applied on the vertices of the primitives. New transformed vertex values are then combined together to form the final point, line, polygons. This stage is known as the geometry processing stage of the standard graphics pipeline.

Rasterization: Fragment values of the primitives are determined in the rasterization stage. Each fragment value corresponds to a pixel value on the final display device. If there is an association with the primitive and a texture, then rasterization operation takes place.

Fragment operations: Various fragment operations are performed such as lightening, culling, depth test, etc. Final color value of the corresponding pixel is determined in this stage.

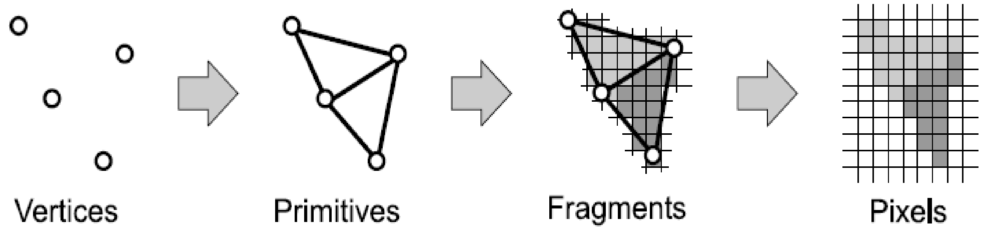


Figure 3.19 Standard graphic pipeline.

3.8.2 Geometry Processing

Per-vertex operations are performed in this stage. Transformations and the per-vertex lighting operations are performed. More precisely, model transformations that are positioning each primitive in the virtual world takes place. Transformations are performed by 4x4 matrices in homogeneous space. One another transformation called viewing transformation also takes place in this stage again with 4x4 matrices. Viewing transformation is used to position the camera that projects the virtual world or scene. Combination of the model and view matrix generates single matrix which makes it efficient in numerical calculations. After the recent calculations Phong (Phong, 1973) shading algorithm is used to make the illumination calculations on the model. Next step in this stage is the clipping operation where the invisible parts of the model and parts out of the camera scope are clipped. Finally perspective or orthogonal projection matrices are performed on the recent vertex values to project them on the scene with their real positions.

3.8.3 Rasterization

Rasterization is the process of converting geometric data into its corresponding fragment value or pixel value on the output device. Polygons are rasterized by calculating the inner pixels and linearly interpolating the properties of the polygon such as color, within the interior pixels. If there is a texture associated with the polygons, then corresponding texture value is interpolated on the polygon.

3.8.4 Fragment Processing

In this final stage of the standard pipeline, different test operations are performed on the resulting pixel values. First of this test operation is the alpha test that is used to discard the pixel values according to their alpha component. Then stencil test is applied to discard pixel values according to predefined conditions. Next one is the depth test, applied for discarding models or their sub-regions which are hidden by the frontier models. And finally alpha blending operations are performed to generate transparency effects on the models.

3.8.5 Programmable Graphic Processing Units

Main innovation of the recent graphics cards is that they allow the developers to define their own graphics pipeline stages. User can define his/her custom vertex (geometry) and fragment (rasterization) stages by low level programming. This hardware specific low level language consists of similar assembler commands that are used to program the CPU. Recent developments also introduce new high level languages for programmability where some of the major ones are GLSL, HLSL, CG, etc.

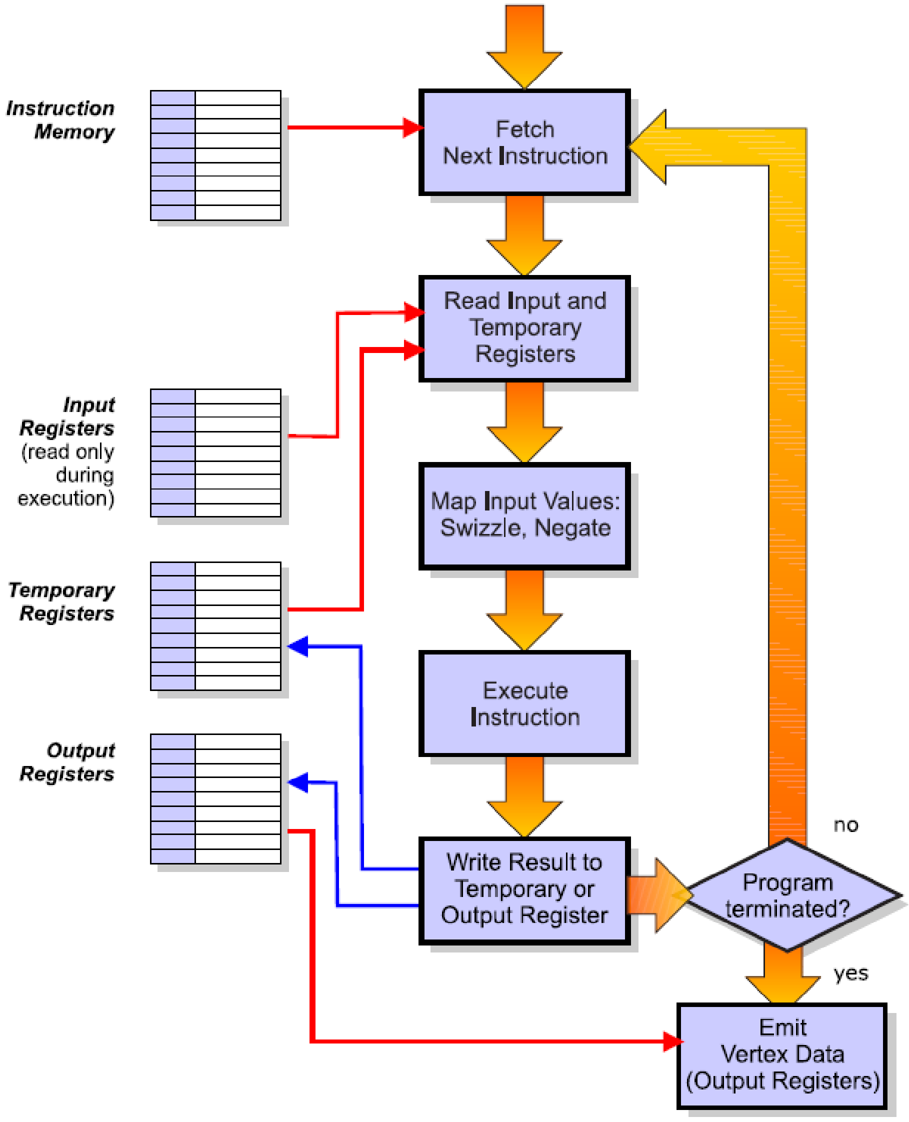

Instruction Memory
Fetch Next Instruction
Read Input and Temporary Registers
Input Registers (read only during execution)
Map Input Values: Swizzle, Negate
Temporary Registers
Execute Instruction
Output Registers
Write Result to Temporary or Output Register
Program terminated?
no
yes
Emit Vertex Data (Output Registers)

Figure 3.20 Pixel processing unit architecture (Real-Time Volume Graphics, 2004).

We can summarize the processing schema of the graphic hardware under two discrete unit; pixel and fragment units. Each unit can be programmed by its own program called pixel and fragment programs. Architecture of the pixel processing unit is represented in Figure 3.20. Also Figure 3.21 represents the architecture of the fragment processing unit.

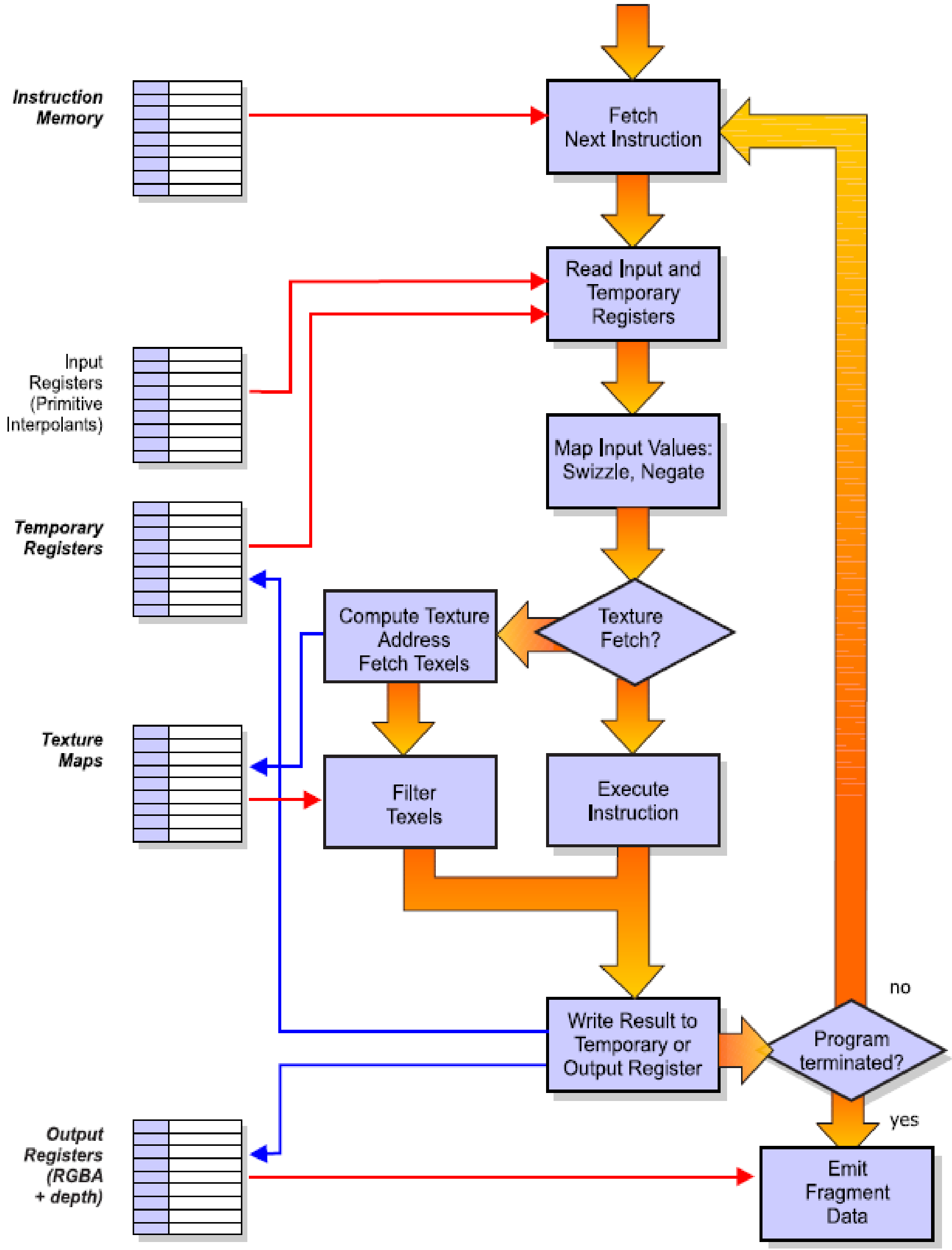

Instruction Memory
Fetch Next Instruction
Read Input and Temporary Registers
Input Registers (Primitive Interpolants)
Map Input Values: Swizzle, Negate
Temporary Registers
Compute Texture Address Fetch Texels
Texture Fetch?
Texture Maps
Filter Texels
Execute Instruction
no
Write Result to Temporary or Output Register
Program terminated?
yes
Output Registers (RGBA + depth)
Emit Fragment Data

Figure 3.21 Fragment processing unit architecture (Real-Time Volume Graphics, 2004).

CHAPTER FOUR
REAL-TIME WAVE PROPAGATION SIMULATION

4.1 Introduction and Related Work

Using high performance computation (HPC) methods in scientific research fields is the most popular approach for simulation of real events. Since many years, because of hardware constraints, scientists could not make enough simulations of experiments with different parameters on massive data. Even there exists HPC devices; they are available only in few specific research laboratories because of their cost. Nowadays, second phase of the new high performance scientific computation trend is getting popular in many fields. This trend is basically using existing graphic card hardware for parallel execution of single instruction on massive data. Because of huge demand on graphically realistic computer games, new type of graphic cards becomes a standard hardware in recent desktop computers. Main feature of this hardware is that it is capable of performing many complex illumination algorithms on every screen pixel at a time. Since this feature is noticed by the people in parallel computation field, it is under the subject of the field with its limitations. Hardware manufacturers are now producing much more capable graphic cards that can be used for general purpose computation. Second phase of this trend is using these commodity graphic cards for complex scientific computations.

Scientists from many different fields need to behave in multidisciplinary way to benefit from these developments. Since scientific computation with commodity graphic hardware is getting popular in many fields, in this section we will explain the details of the topic with its limitations. As a case study, we will demonstrate the results of real-time electromagnetic wave propagation simulation with commodity graphic cards, their performance comparisons with the ordinary computation methods. Since scientific computations are generally performed over massive data, we also mention about memory allocation constraints and real-time visualization techniques for such size of data.

4.2 Memory Allocation

Most of the scientific researchers use third party computational tools for modeling, simulation and analyzing their hypothesis. Main demand on using third party tools like Matlab, Mathematica, and Mapple (Mapple, 2008) is their simplicity and high level interfaces and languages. These features make it possible to easily start developing your own application with very complex functionality. Using these tools, one doesn't need to know mainly the low level programming languages, code optimization, and visualization techniques. Moreover to get the benefits of such tools one must sacrifice the optimal usability of the hardware and software resources. It is not fair to expect every scientist from different discipline than computer science to understand and optimally use the hardware resource. In this case the only way is to use third party tools with their high level interfaces. Aside from the simplicity of these tools, most of the researchers prefer to use specific type of operating systems (OS) since they are easy to manage and use. But one must know that by having simplicity you sacrifice the flexibility and capability of resources. In this section we will mention about the memory constraints and how it is managed by one of the popular operating system Microsoft Windows (Microsoft, 2008).

Most of the computers' processing units are designed for 32 bit computation, so we will focus on 32 bit operating systems running on these computers. Since you can represent 4,294,967,296 = 4GB different numbers with 32bit numeric data type, main disadvantage of these types of computers are that their address space is also limited to 4GB. We know that with specific type of computer mainboards, you can install more than 4 GB physical memory on the computer. Since we are using 32 bit operating system, it means that we cannot benefit from memory space larger than 4 GB.

Reasons for having larger memory then 4GB is out of the scope of these section so we will just focus on this 4GB memory space and how we are efficiently using it. As it is same in our case suppose that we want to make computation on 512 * 512 * 512 size data cube (Figure 4.1). Also assume that we need to use high precision numbers in our computation. Today's computers have built-in support for IEEE 754-1985 double precision numbers, but one can use third party tools for representing the numbers with much more precision. In that case, the computation speed decreases radically so it is not the preferred way. Since double precision number requires 8 byte memory space, in our example, for using double precision numbers you need 1 GB of memory space for single data cube. Moreover the requested space must be a single memory block. This means that the slots of memory block must be consecutive and there should not be any gap between cells. This single block allocation is mandatory for several optimized computations like matrix multiplication, Fast Fourier transformations, etc.

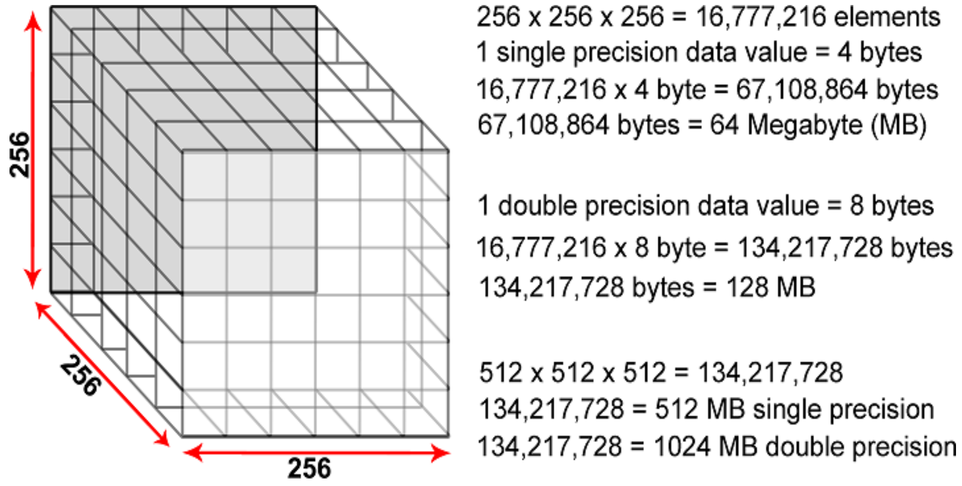

256
256
256
256 x 256 x 256 = 16,777,216 elements
1 single precision data value = 4 bytes
16,777,216 x 4 byte = 67,108,864 bytes
67,108,864 bytes = 64 Megabyte (MB)
1 double precision data value = 8 bytes
16,777,216 x 8 byte = 134,217,728 bytes
134,217,728 bytes = 128 MB
512 x 512 x 512 = 134,217,728
134,217,728 = 512 MB single precision
134,217,728 = 1024 MB double precision

Figure 4.1 Data cube representation, a single memory block.

With 32 bit operating system, having 4 GB physical memory doesn’t mean that you can benefit from this space as you want. Since the operating system itself runs different applications like firewall, internet connection management software, virus checkers, etc. some part of this space is already consumed. Since these memory allocations are spread through the memory with several gaps between them it becomes impossible to benefit from a large memory slot. Because it is physical memory and rapidly changes its size and lifetime, it is not possible to de-fragment these spread blocks.

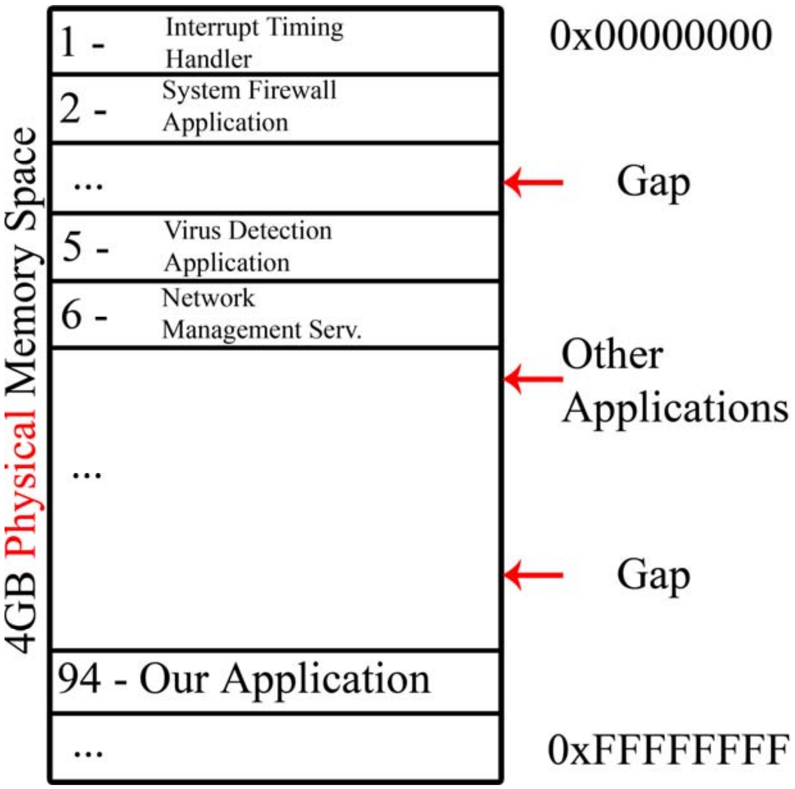

1 - Interrupt Timing Handler
2 - System Firewall Application
...
5 - Virus Detection Application
6 - Network Management Serv.
...
94 - Our Application
...
4GB Physical Memory Space
0x00000000
Gap
Other Applications
Gap
0xFFFFFFFF

Figure 4.2 Sample memory allocation within 4GB space (4GB in hexadecimal equals to 0xFFFFFFFF).

In Figure 4.2, sample memory allocations of the applications are represented. Because the user or program itself, lunches and terminates lots of applications or threads during its runtime, this sample memory allocation schema dynamically and rapidly changes and as in the example in Figure 4.2, there exists gaps because of the terminated applications. In this example we just show the allocation of the physical memory. In reality each application has its own memory space and this space is restricted to 2GB by default, with some low level operating system settings you can benefit maximum 3GB memory space per application. Question is how it is possible for more than one application to have their own 3GB memory space with 4GB of physical memory? In this point virtual memory concept gets into account.

Because of limited and expensive physical memory resources, operating system uses virtual memory concept where larger physical memory is simulated on a portion of hard disk. If memory space is required for an application and if there is not enough space on the physical memory space, then virtual memory is used for allocation. If an application wants to access (read/write) data on a corresponding area of its memory space then specific memory part is loaded into the physical memory space and operations are handled. This scenario is slower because of accessing mechanical device, hard-disk, for reading and writing data.

By means of virtual memory and 32 bit OS, each application has maximum 3GB memory space dedicated for itself. Even we can address 4GB space with 32 bit, the reason for having maximum 3GB is related with OS itself. Because OS wants to communicate with each application to manage its executions, requests, etc., one of the 4GB memory space for each application is reserved by OS. That is why we can allocate maximum 3 GB space per application.

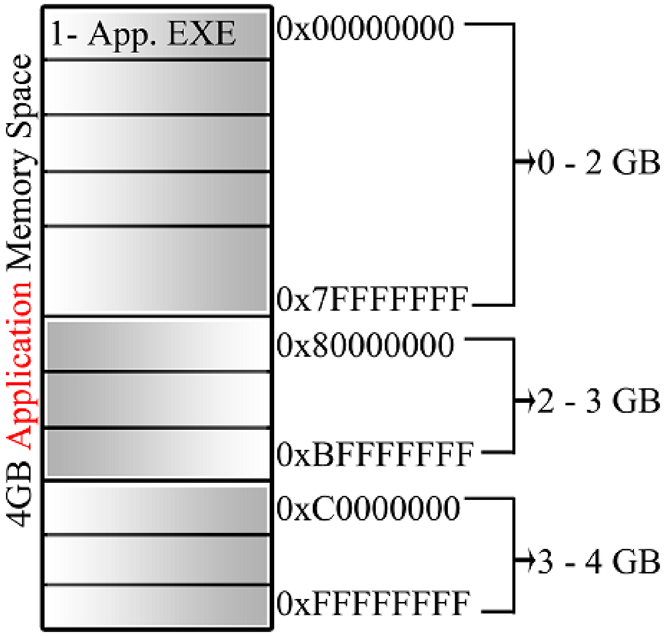


Figure 4.3 Sample application memory space allocation.

In Figure 4.3 sample application memory space allocation is illustrated. Last memory block is reserved by OS for communication with application. 2-3 GB memory block is enabled by specific OS settings. By default, application is using the 0-2 GB memory block. Normally if an application is using an extra library module (i.e. DLL file), it is loaded into the same memory space of the application. In case of simple application, by default it loads system specific DLL files like C Runtime libraries. Generally these system specific libraries are loaded into the memory space range 1-2 GB. So it is clear that first one gigabyte memory block is fragmented by application's executable, second one gigabyte block is fragmented by system dlls and normally we don't have single 1 GB memory block left. In case of 512 * 512 * 512 dimensioned double precision memory block is required, it is not possible to allocate this space. This is why third party computation tools give "out of memory" warnings when we try to work on big data sizes.

To overcome this problem there exist many solutions which sacrifice the computation efficiency. One solution is to divide the cube into small memory blocks and make computation on each piece separately. This is not impossible but very inefficient in case of specific computations like Fast Fourier Transformation (FFT) since the computation of a block is dependent on the cumulating of the other blocks. Other solution is to create memory mapped files where you can create any size virtual memory on harddisk and use it by loading the corresponding part of the file for computation. In this case there will be many load/store operations between the physical memory and the harddisk which is very inefficient for large size data computation.

In our case we prefer to use 512*512*512 single precision data cube which requires only 512MB free single memory block and which is almost always available if 3GB memory support is enabled by the operating system. For larger data size that can fit into the physical memory, one can prefer 64 bit operating system to overcome all those problems that are addressed.

4.3 Electromagnetic Wave Propagation Basics

As it is detailed in section 1, propagation of an electromagnetic field requires special computation techniques. In our experiments we consider electromagnetic wave propagation and elastic wave propagation in different anisotropic media generated by different types of sources.

To understand how electromagnetic waves propagate in anisotropic media is a very important question when we study the properties of new materials, such as crystals with cubic, hexagonal, tetragonal, trigonal, orthorhombic, monoclinic and triclinic structures (Cohen, Heikkola, Joly, Maki, 2003; Graff, 1975).

Mathematical model of electromagnetic wave propagation in an anisotropic 3-D domain is described by Maxwell's system with arbitrary positive definite symmetric matrices of dielectric and the magnetic permeability. The electromagnetic waves are excited by a pulse, external current concentrated at a fixed point (Humphries, 1997).

The explicit formulas for generalized solutions of initial value problems for this Maxwell system with different types of anisotropy are obtained by Yakhno et al. (2003). These formulas have the form of a linear combination of the Dirac delta functions with the support on the fronts of three waves and its derivatives, Heaviside step function, with the support inside the wave fronts. Explicit formulas were described in details in (Yakhno et al., 2003) and chapter 2.

From one hand these formulas look rather cumbersome and complicated but on the other hand they have quite regular structure and symmetry. This allows us to make a decomposition of the formulas and calculate in parallel the values of points for different fragments of the images. This parallelization essentially reduces the runtime of the simulation program.

A dynamic mathematical model of elastic wave propagations in anisotropic elastic media is described by the system of partial differential equations. This system can be reduced to Symmetric hyperbolic system of the first order (Fedorov, 1968; Dienlesaint, 1980).

The initial value problem for the system of anisotropic elasticity is considered from two points of view which allow us to find a solution of the initial value problem with polynomial data and create a procedure of the polynomial coefficients recovery (Yakhno et al., 1998; Yakhno et al., 2000).

4.3.1 Simulation of Wave Propagation

In our framework, all types of dielectrics are considered for wave propagation simulation. Developing a generic formulation for all types is not possible with existing hardware and software. So depending on the structure of the dielectric, generic formulation for specific dielectric type is generated. In all cases, symbolic calculations are handled until the inverse Fast Fourier Transformation (IFFT) stage. Figure 4.4 summarizes the overall wave propagation simulation process.

Calculation starts with the determination of the specified epsilon value. If it is diagonal case, then we use pre-calculated symbolic equation of the specified crystal type. If it is one of the complex cases (orthorhombic, monoclinic or triclinic) then all calculations until the IFFT stage are performed from scratch. First reason for this distinction is that the pure symbolic calculation is not possible for complex cases with the computation platform that we use. This limitation does not have direct relation with the physical resources but the computational capability of the Mapple kernel integrated in Matlab. Second reason is that the precision of the standard numeric data type is not enough for the computation of complex symbolic epsilon value and even for some of numeric epsilon values.

These symbolic and in some cases numeric calculations are performed in a different computation space. After finding the solution, intermediate result is then transformed into the original space by means of IFFT operation. To do so, first substitution operation is applied. Because of the complexity of the symbolic intermediate equation, it is not possible to apply symbolic IFFT operator. So in this stage we turn back to numerical computation approach which is performed on massive data.

For simple cases intermediate result also contains symbolic variables for μ_1, μ_2, μ_3 on the other hand in all cases intermediate result contains symbolic time variable. Main benefits of having the resulting equation in symbolic notation are; first there is no loss of accuracy or round-off errors until that computation stage, second it is very efficient to make substitution for dynamic parameters like time parameter in our case.

Final stage of the simulation is to substitute the intermediate equation with specific time and numeric epsilon values. Resulting equation is used to compute the value of each cell of the data cube that is mentioned in subsection 4.2. Finally, data cube is visualized as an intensity valued cube. To achieve dynamic propagation simulation, in each data cube rendering stage, we substitute the intermediate solution with different time value, and this is one another reason to use symbolic notation.

Symbolic formulations intended to be generated for the following crystallographic structure of dielectrics are:

$$\mathscr{E} = \operatorname{diag}(\varepsilon_{11}, \varepsilon_{11}, \varepsilon_{11}), \quad \varepsilon_{11} > 0$$

$$\mathscr{E} = \operatorname{diag}(\varepsilon_{11}, \varepsilon_{11}, \varepsilon_{33}), \quad \varepsilon_{11} > 0, \varepsilon_{33} > 0$$

$$\mathscr{E} = \operatorname{diag}(\varepsilon_{11}, \varepsilon_{22}, \varepsilon_{33}), \quad \varepsilon_{11} > 0, \varepsilon_{33} > 0, \varepsilon_{22} > 0$$

$$\mathscr{E} = \begin{pmatrix} \varepsilon_{11} & \varepsilon_{12} & 0 \\ \varepsilon_{12} & \varepsilon_{22} & 0 \\ 0 & 0 & \varepsilon_{33} \end{pmatrix}, \ \varepsilon_{11} > 0, \ \varepsilon_{33} > 0, \ \varepsilon_{11}\varepsilon_{22} - \varepsilon_{12}^2 > 0$$

$$\mathscr{E} = \begin{pmatrix} \varepsilon_{11} & \varepsilon_{12} & \varepsilon_{13} \\ \varepsilon_{12} & \varepsilon_{22} & \varepsilon_{23} \\ \varepsilon_{13} & \varepsilon_{23} & \varepsilon_{33} \end{pmatrix}, \ \mathscr{E} \text{ is symmetric positive definite}$$

For the first three crystal types there is no problem to generate symbolic equation for computation. Because of graphic cards' memory limitations (which is different from physical computer memory) fourth crystal is partially computed. For three distinct electromagnetic field components, generated symbolic equations have very long intermediate computation steps and even the simplest component equation requires many variables in graphic cards memory. But the numbers of registers for variables are limited with the graphic cards features. For the last crystal, it is currently not possible to generate symbolic equations for simulation.

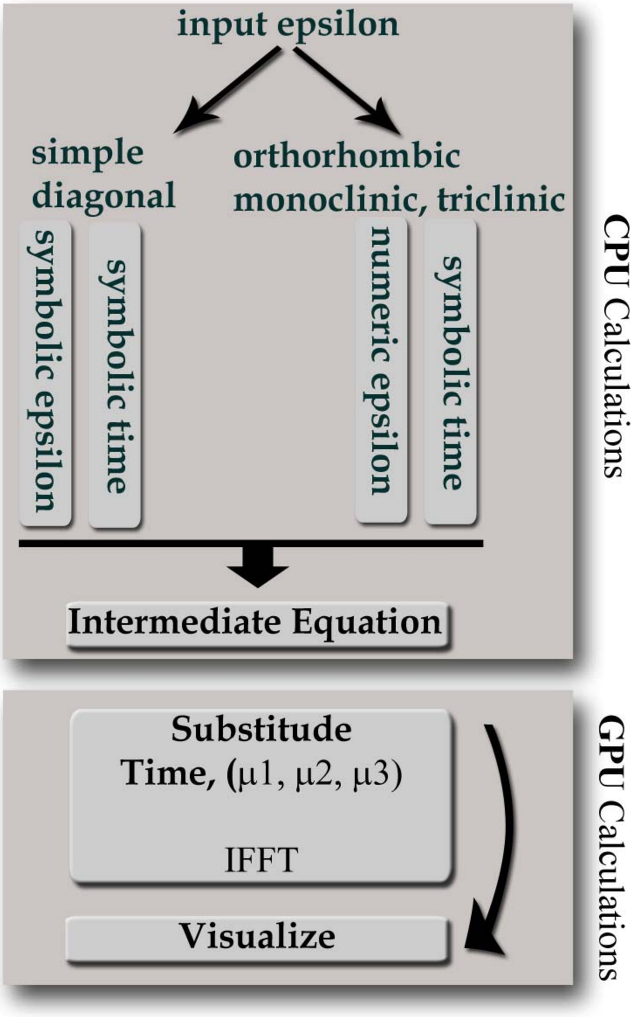


Figure 4.4 Overview of wave simulation framework.

As it is represented in Figure 4.4, it is not possible to perform pure symbolic calculations with monoclinic and triclinic cases even with enough computation resources. For some epsilon values, also it is not possible to perform numeric computation because of the computational capability of Mapple kernel used by Matlab. In our framework we use the MEX programming interface provided by Matlab to integrate the symbolic calculation feature into our C++ code.

4.4 Fast Fourier Transformations

One of the numerical computation steps in modeling approach is IFFT. Reader may find introductory knowledge about FFT and its application from Bracewell

(1999). Since $\widetilde{E}(\nu,t)$ and $E(\nu,t)$ have real values, we use the following formulation for raw computation of the IFFT.

$$F_\nu^{-1}\left[\widetilde{E}\right](x,t)=\frac{1}{(2\pi)^3}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\widetilde{E}(\nu,t)\cos(\nu,x)d\nu_1 d\nu_2 d\nu_3$$

Visual representation of this formulation means that during the transformation of one space to another, all elements of the first space are involved in calculation of each element of the second space. Implementation of this formula in programming environment requires six nested loops, where the inner loop makes trigonometric calculations. For a data cube with each dimension equals to 256, number of computations will be 256^6 = 281,474,976,710,656. Raw computation of such formulation with an ordinary desktop will take months to complete. So for CPU based computation where the instructions are executed one by one, performance of the loop computation is extremely inefficient. For these reason there exists optimized Fourier Transformation algorithms that are much more efficient than the loop based approach.

There exists many efficient well know FFT libraries mostly used in scientific computation field. One of these libraries is called FFTW (FFTW, 2008), which is a C subroutine library for computing the discrete Fourier transform in one or more dimensions of arbitrary input size, and of both real and complex data (Frigo & Johnson , 1998). Others are part of computational libraries called Netlib (2008) and Numerical Recipes (2008).

Some of these libraries require extra memory space as same size as the input arguments for FFT calculation. It is called “out of place” calculation in case of extra memory space required and “in place” otherwise. For 512 MB size data, it will try to allocate one other copy of the data for computation. As it is explained in section 4.2, allocating big size memory blocks is generally impossible in 32 bit operating systems. So such algorithms generally fail to run on large size data. One other alternative that we used in our implementation is to prefer the graphical processors built-in computation functionality. Details of this functionality are explained in section 4.5.2.

4.5 GPU Based Computation

Multimillion dollar computer game industry has a very important impact on the development of the graphic cards- hardware. Desire for a better graphics and realism in the games encourage the hardware manufacturers to develop specialized graphic cards that can be programmable. Also recent technological developments make it possible to use large number of transistors in the processors which results massive parallel computation. Parallelism makes it possible to process thousands of pixels and vertices at a time, which is one of the bottlenecks in the graphics hardware. Currently almost all commodity graphic cards have its own Graphics processing unit (GPU) for parallel processing of the pixels and vertices. Aside from the computational power, memory bandwidth of the new graphics hardware is multiple times faster than the main system. Not surprisingly, these developments for graphical realism are also become a de-facto as general purpose computation with graphics hardware.

Because of its complexity it was not possible to efficiently use features of a new generation graphics card with low level programming languages. Recently major GPU manufacturers developed high level programming languages along with their new hardware. Early in 2007, Nvidia announced “Compute Unified Device Architecture” (CUDA) where massive general purpose computations could be done with high level language interface.

In the thesis, we refer to graphics cards with GPU and dedicated high speed memory as “device”, hosting platform with CPU and system memory as “host”. The architecture of this system is represented in Figure 4.5. Instead of using CPU for computation and transferring thousands of resulting data between the device and the host memory, it is now possible to directly process huge amounts of data in a clock cycle over the device memory. These features enable the user to process a huge amount of data within a millisecond since the GPU architecture is designed to execute a single instruction on multiple data.

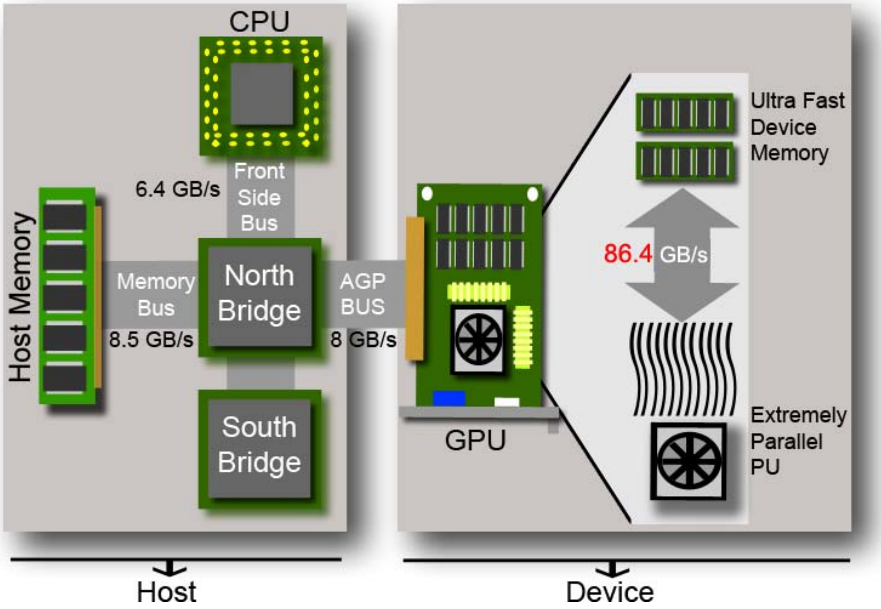


Figure 4.5 Single core computer system architecture.

4.5.1 CUDA Computing Architecture

CUDA is designed to let data and computation intensive problems to be solved in GPU with revolutionary programming interface. CUDA simplifies the development of applications with standard C programming language. CUDA software stack is an interface between the GPU and the CPU and lets it possible to access GPU functionality with CUDA libraries and runtime. These libraries are set of optimized functions for FFT, BLAS (Basic Linear Algebra Subprograms), trigonometric and logarithmic calculations. Over these libraries, runtime layer provides high performance computation functionality with specific device driver interaction.

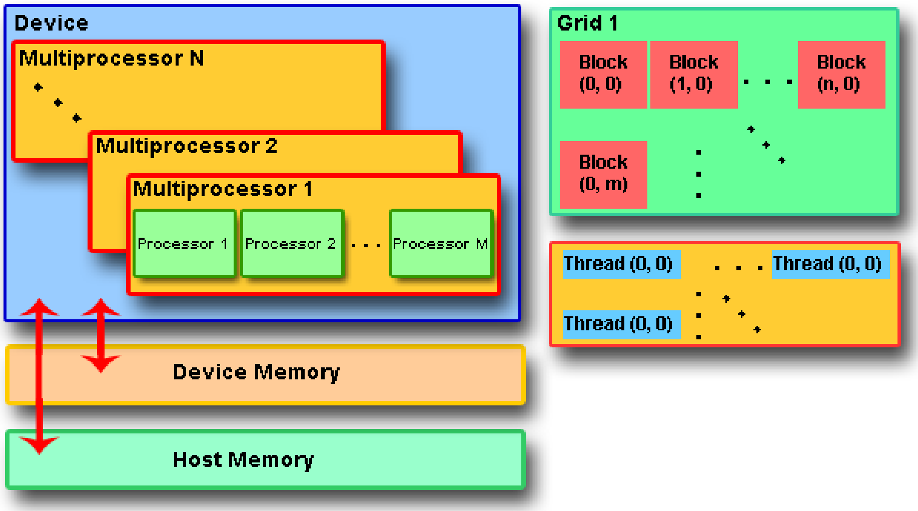


Figure 4.6 CUDA computing architecture summary.

Figure 4.6 represents the simplified architectural overview of CUDA execution concept. Depending on the graphic cards model, number of multi processor stack is available. Each stack has multiple processors that are used to execute the same instruction on a set of data, at a time. For parallelism, data is divided into thread blocks that are encapsulated by a grid. Depending on the parallel computation capability of a device, grid of block can be executed at a time or sequentially. User defines C coded kernel function that is executed on each thread.

Even the programmer is flexible to use general programming approach, to get full performance of the CUDA architecture, one need to pay attention about the hardware architecture. Depending on the hardware capability, number of declared local variables and the way of access to local and global variables in the kernel program would make big impact on the performance. If the number of local variables doesn't fit into the multiprocessors' registers, then it will require more clock cycles to execute a single instruction. Since the architecture of the system is executing many blocks at the same time, if more than one block tries to access the same memory location, then parallel execution is blocked to prevent memory conflicts.

Taking into account such performance issues, performance of CUDA programs are strictly specific to the CUDA compatible hardware architecture and the way of using the resources.

4.5.2 FFT, BLAS and Built-in Function Libraries

CUDA has a set of built in libraries that are specially designed to benefit from the parallel processing capability of the GPU. Aside from the efficiency, main disadvantage of this architecture is that it is based on a single precision computation. One of these libraries consists of several FFT algorithms each of which has different performance and accuracy depending on the input data size. These algorithms operate on real and complex data with in-place and out of place computation manner. Second main library package is called BLAS which consists of basic linear algebra subprograms that is based on Netlib. This second library is designed to perform fast matrix algebraic operations and to solve linear algebra equations. Aside from these libraries, there exist built-in function calls that are subset of standard C library. This built in functions are computationally much more efficient than their C equivalents but they just work in single precision mode.

4.6 Volume Visualization

Most of the 3D data capture devices such as MRI, human body scanners, etc. can generate large amount of data. Analyzing such data through its visual representation is one of the main targets of volume visualization field. As it is represented in Figure 4.5, processing massive data in the host according to real-time user interaction and then translating it to the device for visualization has a performance bottleneck. Volume visualization field generally focuses on the optimization techniques to overcome these problems.

Volume rendering techniques are classified under two broad categories; direct and indirect (surface fitting) rendering. Indirect methods use an abstraction like iso-surfaces to visualize the data. On the other hand, direct methods blends each data value along the eye space and projects the final data on a plane. Direct method is generally preferred because of its efficiency but requires post-processing stage for better visual result. Furthermore, direct methods are categorized under forward, backward and hybrid rendering headings (Elvins, 1992).

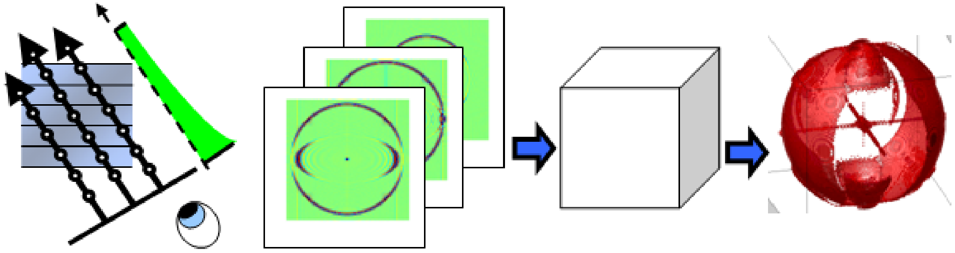


Figure 4.7 Direct volume rendering summary.

Using OpenGL (OpenGL, 2008) build in texture blending functions, calculated cube data is visualized with direct rendering method. From the viewer's eye position, rays are casted into each particle of the cube. Taking into account all other particles that lay along the cast, intensity value of each particle is accumulated. Particles that are close to eye position have high degree of contribution in the accumulation and the ones far away from the eye position have less contribution. Resulting accumulated values are projected on to a plane for visualization.

According to the user interactions like rotation and scaling, data accumulation for visualization is recalculated. Without special hardware and computation techniques, calculations are performed on the host and the resulting value translated in to the device. Considering the data bus bottleneck between the host and device, real-time performance achievement is impossible with massive data. In our case simulation of wave propagation, where data values are re-calculated in each frame, is much more difficult. It will not be wrong to say that our aim is not to show a novel visualization approach but to show a novel computation approach that is efficiently integrated with standard visualization technique.

4.6.1 Pixel Buffer Object (PBO)

To apply direct volume visualization technique with one of the standard 3D graphic rendering library like OpenGL, two main inputs need to be provided. First input is a set of images that construct the 3D Volume, second one is a proxy geometry that will be used to map this textures on it. In our case the size and shape of data volume is not changing during the simulation time. Only the intensity values of the particles in the cube are changing with respect to the value of time variable. In the case of visualizing large amount of data that is changing with respect to time variable, achieving real-time efficiency is almost impossible without optimization.

As it is represented in Figure 4.5, capabilities of hardware architecture are also limited. One of the main limitations is the data transfer bandwidth between the main system memory and the graphic card. Processing large amount of data with CPU then transferring the result back into the main memory and finally transferring the result again from main memory to the graphic card memory is very inefficient. To overcome this problem OpenGL library extended to allow direct access to the graphic cards memory. New concept called Pixel Buffer Object (PBO) is introduced to access the graphics card memory directly and modify the content. By this way, user has the capability of in-place modification of the texture values of the geometries and there is no need for data transmission. Because all computations are handled in the device and the result is directly written on the graphic card for visualization, big performance gain is achieved.

4.6.2 CUDA PBO Interaction

PBO allows CPU to access graphic cards memory directly and modify the content. As it is represented in Figure 4.5, with recent technological developments there is still a memory bandwidth problem between the CPU and the graphic card. Even PBO is a big improvement to accelerate the data transmission, large set of data visualization with real-time efficiency is still impossible.

CUDA architecture allows interacting with OpenGL primitives like textures and PBOs. Since we make numeric wave propagation calculations on GPU by means of CUDA, we use the in-place calculation method to directly work on the graphic cards memory. Recent graphic cards have at least 10 times faster memory bandwidth between the GPU and graphic card memory. Since we make parallel computation over GPU, we directly store the resulting value in PBO as a 3D intensity texture which is used for final visualization. Aside from powerful computation capability of GPU, in-place modification over graphic cards’ memory provides one another improvement in volume rendering field.

4.7 Accelerated Simulation of Electromagnetic Wave

Electromagnetic wave propagation simulation involves complex computational steps which is error prone because of computer resource limitations. Even using double precision data type, under and overflow conditions occurs while performing hundreds of arithmetic operations. For this reason in our calculations we benefit from symbolic and numeric computation methodologies to overcome the loss of data precision while computing the cumbersome formulas.

Our electromagnetic wave simulation formulation is based on symbolic calculations. Since the calculation of this formulation is not possible “by hand”, we use Matlab software package to find robust result for our problem. Even using computer systems to have robust solution, at some point we are out of system resources to have the final computation result. In this point, we switch to numerical computation method.

Numerical computation of massive data with traditional computer systems is time consuming operation. In most cases, computation time is unacceptable. As it is discussed in section 4.4, without any system failure and loss of data precision, it would take months to finish a simple numerical calculation. For this reason there exist different computation phenomena like grid, parallel, super, distributed computing. All these solutions are very expensive and not easy to implement with standard hardware and software packages. In our method we show that numerical computation on massive data is possible with commodity graphic cards. Our sample implementation consists of several computation steps that are handled in different software/hardware environments but managed from a single application. During these computations, we just use commodity desktop computer system for a complex scientific data computation and visualization.

Figure 4.8 represents the system overview of our wave propagation simulation environment. First step of our framework allocates required memory space both for texture and geometry data and initializes the memory bindings between CUDA and OpenGL.

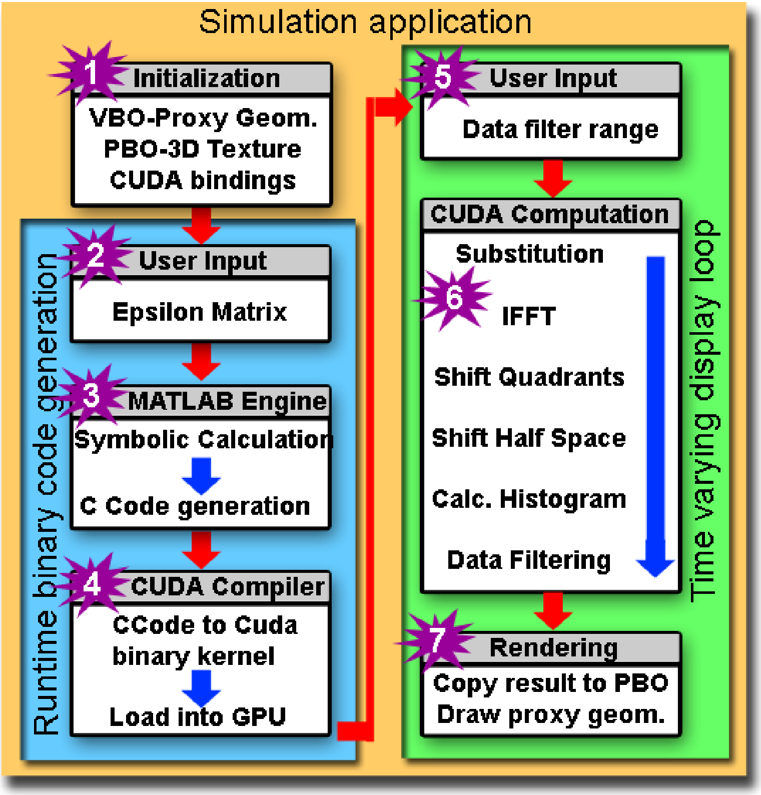


Figure 4.8 Simulation application flow chart.

Because of hardware constraints, we used two different approaches during the Matlab symbolic calculations. In case of user input with diagonal epsilon matrix, pre-calculated symbolic formulation is used; otherwise numeric calculation is performed with symbolic time variable. Using the wave propagation formula generated in the previous step, CUDA kernel code is generated. Here we use again the Matlab feature which generates C code from a mathematical formula. In the next stage where real-time visualization is performed, completely CUDA features are used. Kernel program represented on the right column in Figure 4.8, first makes the substitution of the time variable and if necessary epsilon values. Next, IFFT operation is performed and the result is shifted because of zero frequency. Finally the user input for min-max filter values are used to remove the noisy data. In this stage we implement CUDA based 256 bin data histogram over the resulting data and filtering is based on this histogram. After completing all calculations, resulting data has to be visualized. In this final stage we use the CUDA PBO interaction feature to realize real-time visualization. Because all computations are handled in the GPU, in-place memory operations are performed to eliminate data transmission.

4.8 CUDA Limitations and Possible Solutions

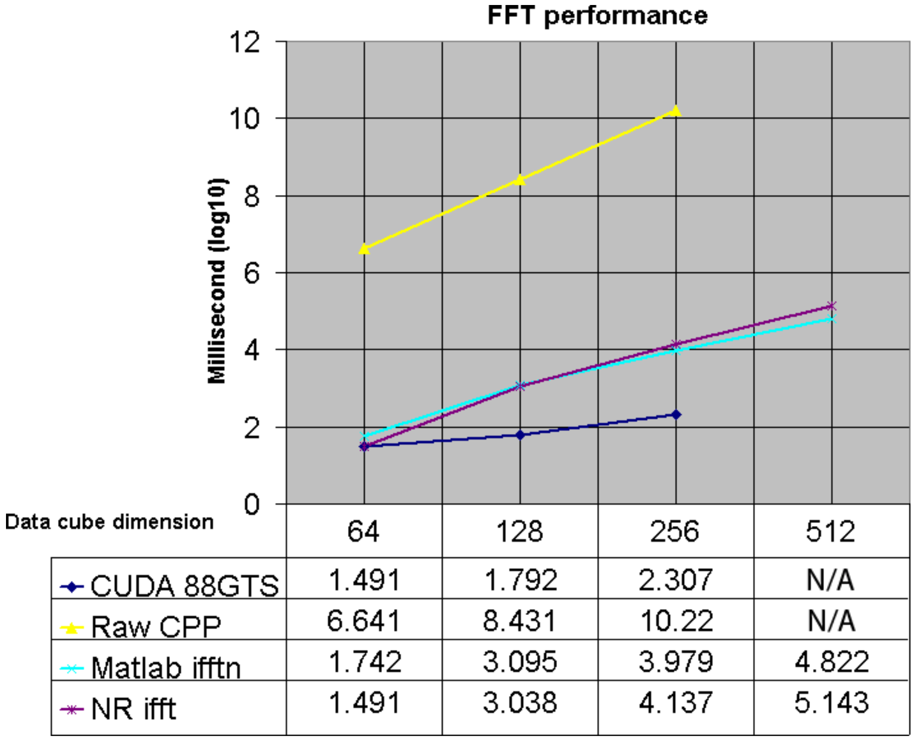
Intermediate symbolic and semi-symbolic equations mentioned in section 4.3 are substituted to pass full numeric calculation space. Before and after this operation, there exist potential numerical precision problems especially in Monoclinic and Triclinic cases. In symbolic case, factors of the symbolic variables can only be represented by double precision data types. So in numeric case we need higher precision data types than double precision.

As it is mentioned in section 4.5.2, NVidia's Cuda compatible GeForce 8 series cards support just single precision data types. For this reason it is not possible to benefit from this high performance computation hardware to compute equations that require double precision data type. Aside from this commodity graphic card hardware, NVidia announced a new graphic card series called Tesla (Tesla, 2008). Main difference of this series is that they can perform double precision computations and they have bigger graphic card memory sizes. These features allow us to overcome the computation constraints like memory size and higher precision data types that we mentioned earlier.

4.9 Implementation

In our framework computation platform is based on Intel DualCore E6600 CPU with 4 GB physical memory and Nvidia Geforce 8800 GTS graphic card. Software packages installed on this system are Windows XP 64 Bit, CUDA1.1 and MATLab 2007.b.

Hardware resources used for CPU based computation stage represented in Figure 4.4 is capable enough to make all calculations. This stage of the calculation is not affecting the overall simulation performance of the framework since all numerical calculation and visualization stage is performed on the GPU. Using power of two size data cubes to get full performance of the built in FFT functions, maximum cube size is limited to 256 because of graphic card's memory limit.



Data cube dimension	64	128	256	512
CUDA 88GTS	1.491	1.792	2.307	N/A
Raw CPP	6.641	8.431	10.22	N/A
Matlab ifftn	1.742	3.095	3.979	4.822
NR ifft	1.491	3.038	4.137	5.143

Figure 4.9 FFT Benchmark between CUDA, CPP, Matlab, NR.

To make clear the analysis of the calculation performance, same computations performed on different environments and computation times represented in Figure 4.9. In our framework, wave propagation simulation stage, where dynamic calculations are performed, consists of inverse fast fourier transform (ifft) and substitution (sub) stages. Doesn't matter how complex the equations are but substitution takes very small amount of time compared to ifft stage, so we omit numerical statistics of this stage. On the other hand we perform GPU and CPU based different ifft implementations with several data cube sizes. Computation timings show that the CUDA based ifft operations are radically faster than the computation of common ifft packages. Because the graphic card that we used for tests has 320 MB memory we couldn't perform ifft operations on 512 size data cubes. With 256 size data cube we achieve 3 frames per second with real time computation. Some screenshots of the experiments from the developed application with different parameters are represented in Appendix B. Also as an example, a parallel matrix multiplication with CUDA architecture is represented in Appendix C.

CHAPTER FIVE
WEB BASED EXPERIMENT LIBRARY

5.1 Introduction

Main problem related with the scientific computation is the requirements to achieve desired results that could be used for further analyzes. Mainly these requirements are coming from the hardware capacity that is necessary for high performance computation. Since the evolution of the computers, many approaches are proposed to overcome these limitations. One of the recent and most attractive approaches is the GRID (Grid, 2008) service for sharing computer power and data storage capacity over Internet for scientific computation. Operation of this service is subscription based and once the scientist becomes subscribed, he/she can submit a function to be computed over data that is provided. Benefit of the service is to compute complex equations or massive data that is not possible to be handled in an ordinary computer. Operation of the service does not include the computation but it is just an interface for the internet resources (computer nodes) to share the computation power of each node. Namely receiving a job from a node and then distributing it to other nodes for computation. From the similar approach but with a different perspective we implemented a web based experiment library that is a single access point where all the computations are done but multiple clients benefit from the result. System caches and stores the computation results in a hierarchical way so that it can be used as a library. Scientist can explore this library to analyze previous test results visually and also compare them. Either the system user can upload his/her own experiment result or send a computation request to the system. To prevent unintended access to the system, security issues are also considered.

5.2 Resource Sharing

Since Internet connectivity becomes a way of standard communication on all over the world, knowledge sharing becomes easier than before. In most cases the resource sharing is considered only under the knowledge category. On the other hand physical sharing of the resources like computer hardware was not possible because of the distance between two nodes. Internet also solves such problems by providing connectivity feature between any numbers of nodes without a distance constraint. Any researcher on the other side of the world can easily access any resource by means of this connectivity. In early years of the Internet it is just aimed for knowledge sharing and for a long time it was used as it was. Later, people start to use Internet for high level message transmission that is similar to the one on a single computer's motherboard architecture. By this way it becomes possible to initiate parallel execution of an algorithm over a huge data. This approach initiates a new research field in algorithms since the benefit of parallel computation is only succeeded by using special algorithms for the same problem.

Resource sharing in computer society can be considered with two general headings: sharing information and sharing hardware resources. In the simplest case information sharing is achieved by web technologies over Internet. Using a standard communication protocol anyone can access public information resources that are available on Internet. Two types of nodes called client and server are the main actors of this scenario where the client requests and server delivers the requested information. If they are standard information sharing nodes, then the software running on the server is called web server and the one running on client side is called web browser. Web browsers are mainly standard software shipped together with the operating systems so we are not considering it as a main requirement for our system's usability.

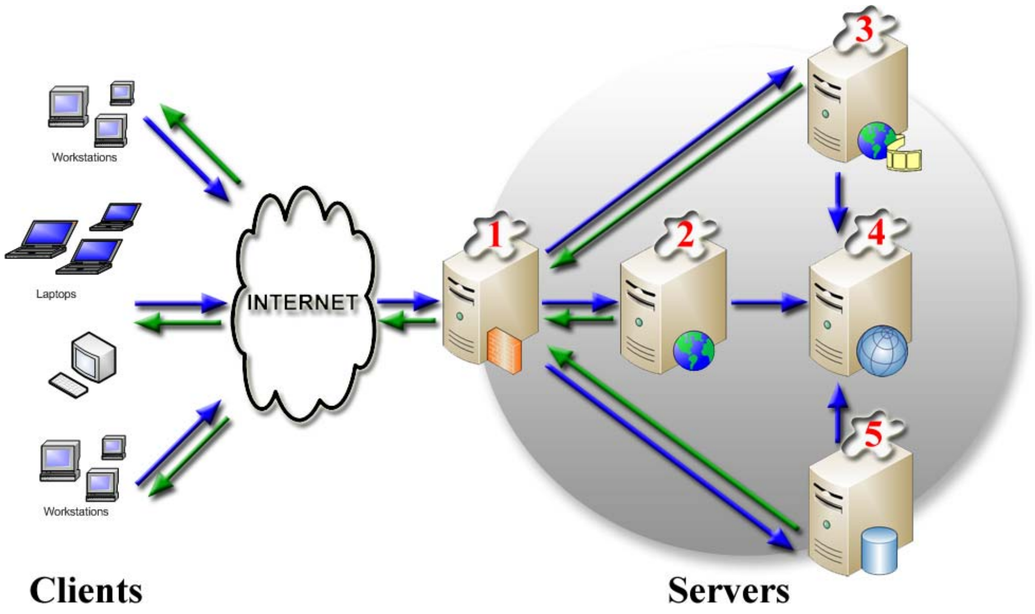


Figure 5.1 Information sharing over Internet.

For information sharing over Internet we tried the server structure represented in Figure 5.1 where all the servers have the ability to connect each other. This figure represents just the main infrastructure that can be extended according to the load/balance rates all over the system. Summary of each server's functionality is listed below with the corresponding server number depicted on the figure.

1. Proxy/Firewall server: To prevent intrusion attack, firewall server is used. By this way data (i.e. formulas, computation results) security is achieved. Firewall server has a different role than the authentication server and the main function is to prevent intrusion. Secondary function of this server is to adjust the load/balance ratio by redirecting the client requests to other servers if available. This is a performance acceleration option in case the numbers of servers are increased to create a web farm structure. This server also includes the SMTP server to handle outgoing messages that will be delivered to a specific domain.

2. Web server: Main component of the system that produces a valid response to the requests already validated by the firewall server. Since all servers are interconnected, web server triggers any database or streaming

operation through other servers and could send the result to the client. All user interface and the applications reside on this server.

3. Stream server: Stream server is used for the optimized data streaming of the requested video file. Depending on the clients connection speed different quality of streaming is provided from a single video file. This optimization process is achieved by switching between different resolution video sizes. In case of communication problem, it is possible for the client to continue from the last play position.

4. Session server: Inter-connected servers that provide different kinds of content directly or indirectly to the client requires same authentication level. For the client there is a single access point visible, that is server 1 as in the Figure 5.1. At the backstage, server 1 is managing the other servers to optimally provide the requested content i.e. user receives a web page with links to video files from server 2 and when the user clicks one of this video files, then stream server 3 directly provides the content to the client by eliminating server 2. In such cases both server 2 and 3 must know whom the user is, and if (s)he is an authenticated user. To achieve this authentication mechanism we use session server which keeps a list of the connected users and their session information. All other servers are referring to session server to authenticate requesting connection.

5. Database server: File indexes (corresponding URL addresses), user provided functions, and function parameters etc. are stored in a database separately stored in a server. Main benefit of the database server is the search facility over the stored data. i.e. it is possible to query a set of experiment and corresponding results for specific parameters.

5.3 Shared Content

In our case, we share the uploaded experiment results or experiments computed directly on the server side. Three types of information is stored on the server side. First one is the experiment type, parameters, time range and code (if available). Second one is the experiment results as a set of images where each image corresponds to a specific time slot. And the final one is the video file of the experiment that demonstrates the propagation of the wave or the combination of images in an animated form. Subscribed user has the ability to access all this information through a specific web site designed to provide mentioned features.

We based experiment library used for management of the experiments that will be handled both by users and scientist. Because there exists, huge amount of experiments that belongs to different kind of crystal materials, it becomes a complicated issue to store and categorize these experiments. Objectives of this web based library have two different access types; administrative and standard. Administrative access has the capability of adding, deleting and updating experiments within the library system. Standard access only has the capability of browsing the library. So there is an authentication system for determining the access type. Experiments are categorized under a tree hierarchy to achieve a user friendly interface. Administrative users will have the following capabilities within the system:

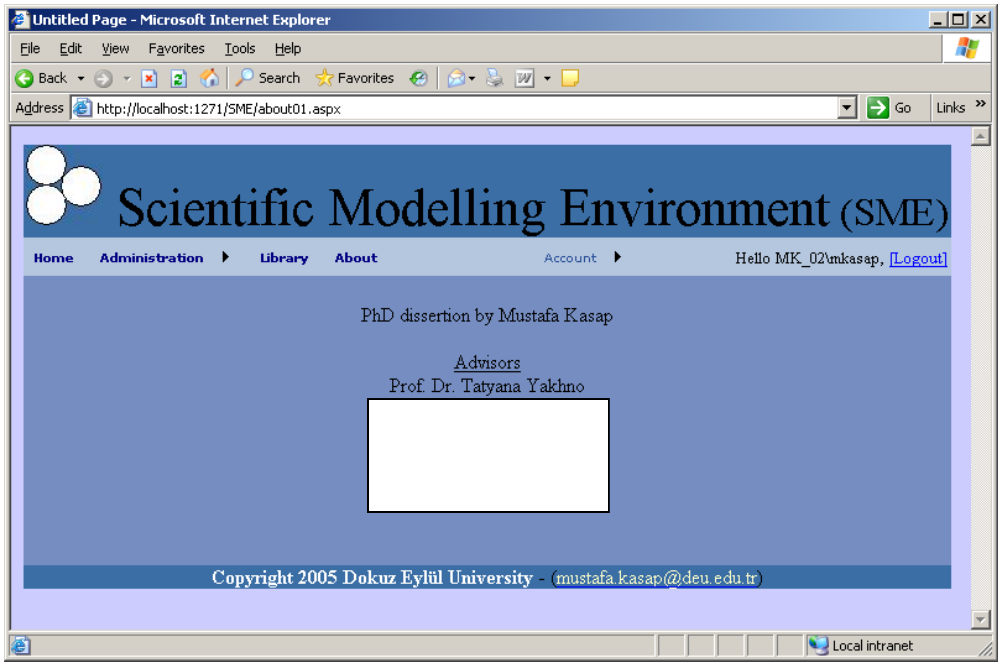
1. Navigate through the library by this hierarchy tree component.
2. Modify the structure of the tree by adding, removing new nodes or changing the node order by replacing the node parents.
3. Each leaf node of the tree has specific types such as frames, animations, experiments etc.
4. Resulting frame node shows the results of the experiment visualized as distinct images. Initially thumbnails of each frame are represented. If user selects one of the thumbnails, high resolution frame is presented. According to the browsed frames, user can generate his/her own favorite frame set for further analyzes. Favorite frame set is associated and stored in the system with the authenticated user credentials.
5. Animation node shows the results of the experiments as a set of animation files within specific time frames. As described in the previous list item, animation results also browsed as frame results.

6. Experiment node has the functionality to show the mathematical definitions of the experiment. User can modify this mathematical symbolic representation.

5.4 System Functionality

All user requests are handled by the web server where the server side applications are executed. Server side applications are similar to normal computer programs but they are executed by the web server with the user specified parameters. These applications either generate some results or handle some operations. If there exists some output generated by the application then the web server redirects this output to the client through web protocol. There exist different type of web servers on the market. In our system we prefer the most popular and academically free web server provided by Microsoft. This server is running ASP.NET 2.0 application framework with many features like extendibility, rich set of user interface components, easy coding etc. Mainly we benefit from rapid development feature of this framework to construct our system. Mainly most of the applications with user interface consist of tree hierarchy or menu controls which are critical for user interaction. For tree view representations structure and menu controls, Microsoft ASP.NET 2.0 introduces two new controls, the TreeView and Menu controls, which can be used to represent hierarchical data. Both controls are feature rich and were designed to work in a wide-variety of scenarios. Both controls can be used to display static data, site-map data, and even database data. In addition, both controls support a rich client-side experience. The TreeView and Menu controls render JavaScript for uplevel browsers, but they are also compatible with down-level browsers.

Below there exist screenshots from the developed system:



http://localhost:1271/SME/about01.aspx

Scientific Modelling Environment (SME)

Hello MK_02\mkasap, [Logout]

PhD dissertion by Mustafa Kasap

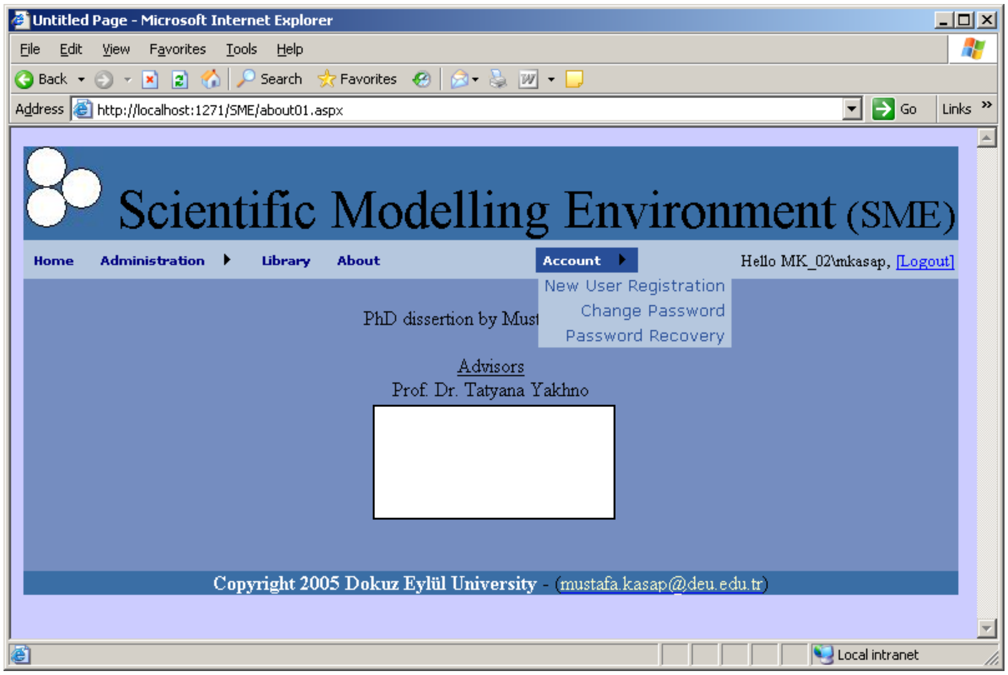
Advisors

Prof. Dr. Tatyana Yakhno

Copyright 2005 Dokuz Eylül University - (mustafa.kasap@deu.edu.tr)

Figure 5.2 Welcome screen of the system

First screenshot represented in Figure 5.2 is the welcome page of the system. This window contains a menu control with a set of menu items where the user can access specific features of the system. Some of the menu items are accessible by authenticated users only. Initially as it is represented in Figure 5.2, account menu is used to login to the system or to change the currently authenticated users profile settings.



Scientific Modelling Environment (SME)

Hello MK_02\mkasap, [Logout]

PhD dissertion by Must

Advisors
Prof. Dr. Tatyana Yakhno

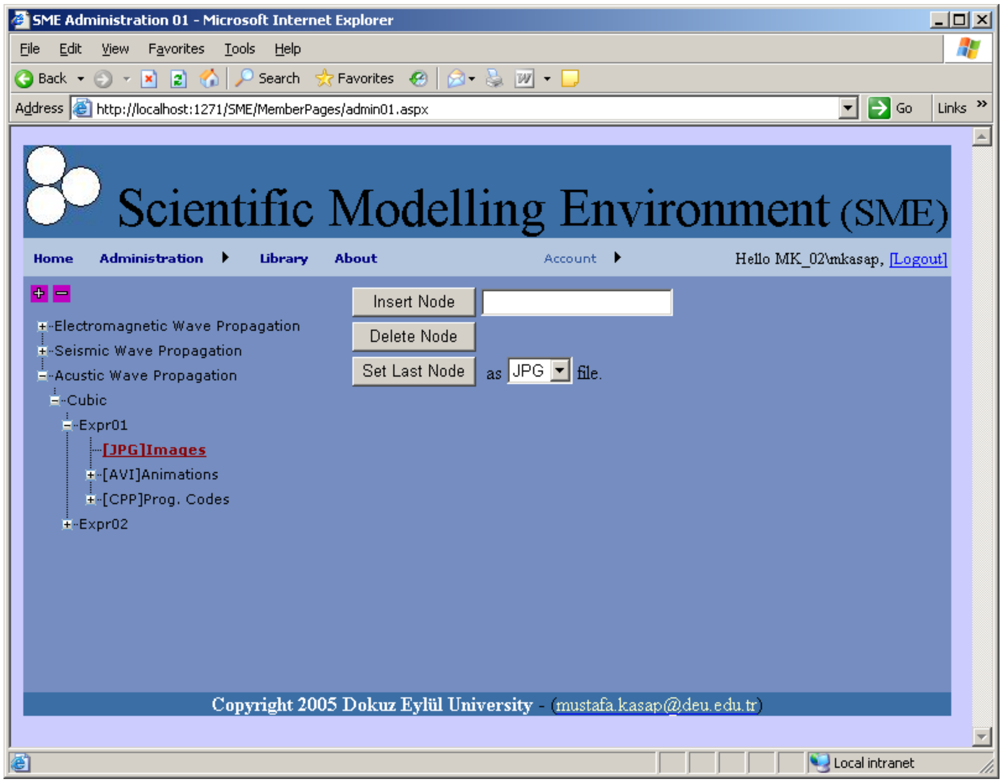
Copyright 2005 Dokuz Eylül University - (mustafa.kasap@deu.edu.tr)

Figure 5.3 Account management menu.

Account management menu consist of following items: New User Registration, Change Password and Password Recovery. Each of this menu items connected with SMTP mail server to send the new user information, forgotten password etc. SMTP server is a simple server running on the firewall side which is a front page of the rest of other servers. Any server can send mail message through this server to the specified destination. It is working as a queue manager to handle the messages that will be sent.

On the administration menu item, there exist two sub menu items: Tree management, File management.

In the file management menu, you can upload experiment files to the system and extract them. Important feature of this file upload operation which differs from the existing ones is it is possible to upload multiple experiment files under a single name with compression. User may prefer to compress experiment files (500 files tested) as a single package. Then upload this package to the system. After upload operation, user may extract this archive to generate the base database of the experiment library session. Other important feature of the upload system is, it has the capability to generate the thumbnails of the uploaded image files. As it is known, web based environments are constrained with the bandwidth of the internet connection. By this way instead of representing ~1MB image file with high resolution, user can preview more simple and reduced quality images before going in depth.


SME Administration 01 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address
Go
Links
Scientific Modelling Environment (SME)
Home Administration Library About
Account
Hello MK_02\mkasap, [Logout]
Electromagnetic Wave Propagation
Seismic Wave Propagation
Acustic Wave Propagation
Cubic
Expr01
[JPG]Images
[AVI]Animations
[CPP]Prog. Codes
Expr02
Insert Node
Delete Node
Local intranet

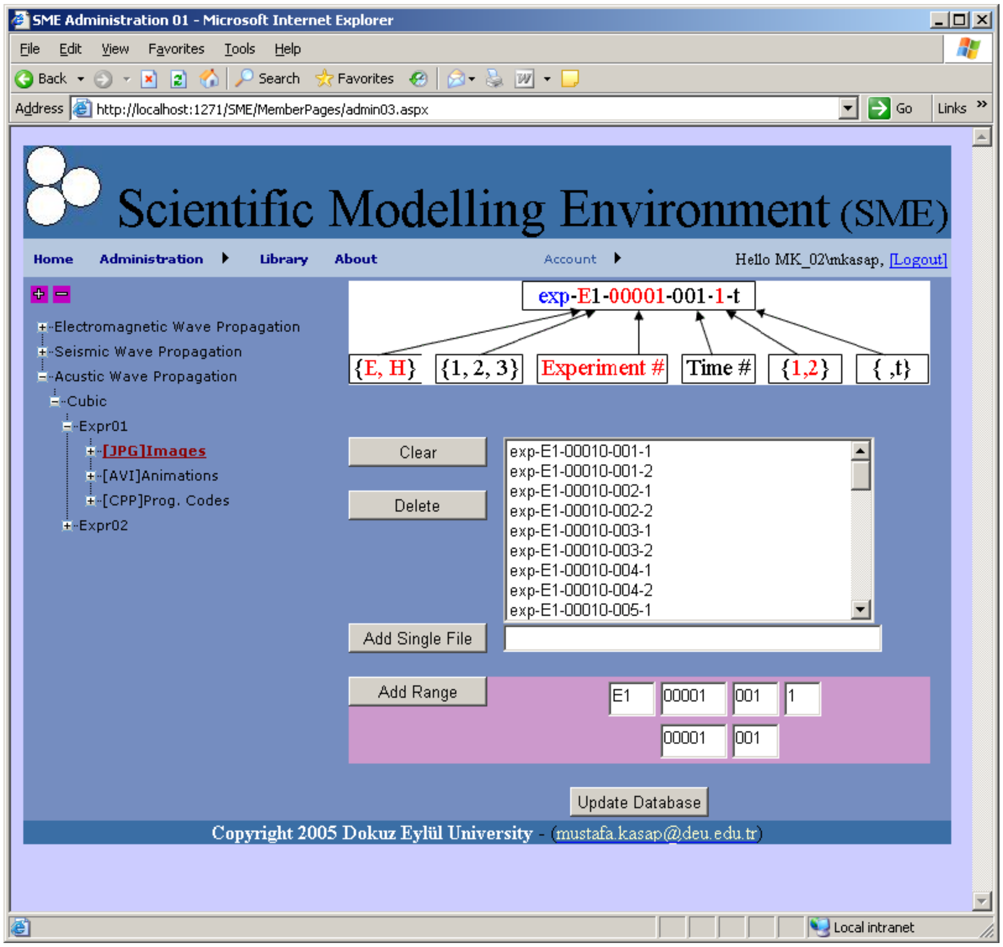
http://localhost:1271/SME/MemberPages/admin01.aspx

Set Last Node as JPG file.

Copyright 2005 Dokuz Eylül University - (mustafa.kasap@deu.edu.tr)

Figure 5.4 Tree administration window

Tree administration window allows user to manage the structure of the organization of the experiment files. User can create animation nodes, CPP code file nodes, MATLAB code file nodes, etc.



http://localhost:1271/SME/MemberPages/admin03.aspx

exp-E1-00010-001-1
exp-E1-00010-001-2
exp-E1-00010-002-1
exp-E1-00010-002-2
exp-E1-00010-003-1
exp-E1-00010-003-2
exp-E1-00010-004-1
exp-E1-00010-004-2
exp-E1-00010-005-1

E1 | 00001 | 001 | 1
00001 | 001

Figure 5.5 Tree node and file association window.

User can associate multiple files to the selected node of the tree. If there exists more than one file related with the experiment result which is uploaded to the system, user can associate these files with the related node. System has the flexibility to generate series of file names automatically.

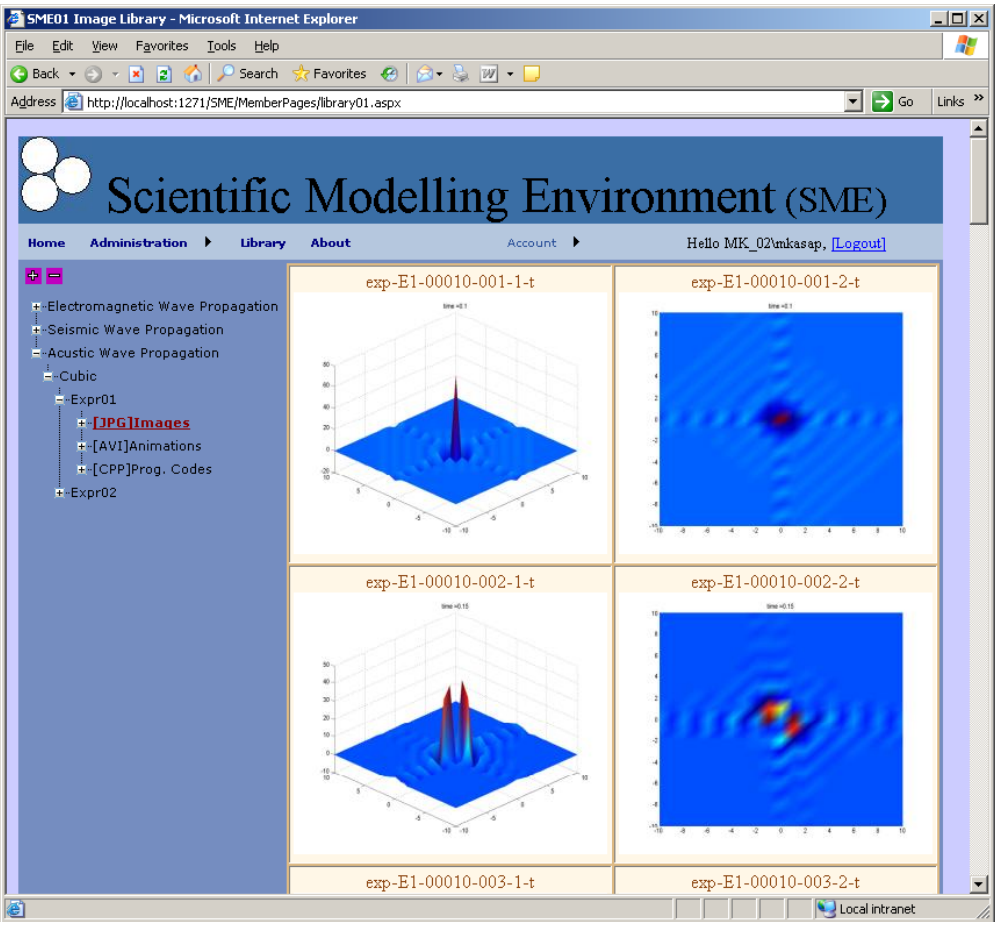


Figure 5.6 Library window while exploring an experiment

5.5 Mathematical Symbol Representation

There exists a syntax highlighting feature together with the mathematical symbol representation. User input may contain mathematical symbols where the syntax must be in Latex (Knuth, 1984) format. There exists latex to web converter plug-in which converts the mathematical equations and symbols to an image file that may be easily published in web content.

For program codes either in Matlab or C, again we are using a third party tool to convert the text based user input into syntax highlighted HTML version. This feature lets it possible to easily interpret the user provided specific text through web browser.

5.6 File Compression and Extraction

Our system also has the functionality of representing the user provided experiment results. For this reason it has the file upload feature where the users can upload the resulting images from their own experiments. Since this image set generally consists of more than hundred files, it is a bit time consuming operation to upload such amount of files to the system. To overcome such problems we let the user to compress (Salomon, Motta, Bryant, 2007) the files in to a single file package and upload just only this file to the system. At the back hand of the system, we automatically extract these compressed files and move them under the user specified experiment. Later on as it is represented in Figure 5.6, these files are demonstrated. File extraction operation is a thread safe operation, so when the file uploads operation has initiated, another application thread is triggered to extract the files without suspending the user interaction.

5.7 Image Thumbnails

Mainly the desire for image thumbnails is to overcome the bandwidth problem of the Internet connection. Thumbnails are the small size, reduced resolution file of the original image. Depending on the compression level, a single original image can be represented by more than ten times smaller image. On the other hand, by reducing the file size we lost the image quality. Since viewing a single large (good quality) image file takes several seconds, it is better to represent more than one image file as a thumbnail and then letting the user to click on any of them for detailed representation. To achieve this functionality again we use the windows operating system features as we did for the compressed file extraction. There exists an API for generating image thumbnail with user specified compression parameter. By adjusting this parameter one can generate very small size images with low resolution.

5.8 System Security

System security is achieved by using the default ASP.NET 2.0 web site security functionality. Normally such functionality is achieved by manual programming and to do so the session ID of the client is used. In our case, ASP.NET framework provides ready to use components to enable this functionality. ASP security tools allow automatically creation of authentication database by connecting to a specified server. Using a set of another authorization tools provided by the framework, it is possible to rapidly generate ready to use secured web sites. Components for login feature are connecting to the database previously created by the framework. After successful login operation, session state is stored by the same components to control the future access. One another advantage of these set of components is feature based user privilige assignment. It is possible to give different access levels and user rights even on the page level.

5.9 Video Streaming

One of the most important features of the system is that it allows to access video files resulting from the images generated by a specific experiments. Since the user uploads a set image files resulting from the experiment, these image files can be animated. As it is represented in Figure 5.5, user must upload image files with specific file names. These file names are encoded in a special format which is self descriptive and also contains information about the time of the propagation. According to the time values succeeding image files are combined as a single frame to generate a video file associated with a specific experiment. This makes it easy to interpret the experiment result by visualizing it in an animated form. It is possible to adjust the frame rate and resolution of the video file for varying internet speeds. Currently these settings are adjusted in system level to generate best quality video files. We handle the streaming quality according to the internet connection bandwidth of the client in an upper layer of the system. Streaming quality is handled in the Windows Media Server layer which is a server application provided by Microsoft. Media server takes a good quality video source and streams it to the clients by adjusting the frame rate, resolution etc. according to the user's connection speed. Also streamed video file is played by Windows media player on the client side. These server and client tools are optimized for better communication protocol. Our client side web pages are playing those video files by embedding a media player plug-in as an ActiveX control. ActiveX controls are recent Microsoft technology that allows running applications inside a web browser in a secure way.

5.10 ActiveX Control for Interactive Visualization

Image and video based visualization of the experiment results are fine for many clients which doesn’t have enough resources for heavy computations. Also most clients don’t have well enough graphic processing capabilities to visualize volume data. There may be two types of clients which may have some constraints about using the system. These are the clients that have limited Internet connection speed and the clients with limited graphical processing resources. For such limitations, image based experiment library and the video streaming server is ideal. Even a user can easily visualize the images from a mobile phone. In case of clients which may have both well internet connection and graphical processing resources, we provide an activeX control that is used to visualize the volume data with user interactivity.

ActiveX controls are Microsoft technologies that make it possible to run computer applications inside a browser. An alternative method for this approach is Java technologies that run on many platforms. In our case we prefer the ActiveX control that is running just on the Microsoft windows system. When we consider the benefit of the whole system’s rapid development feature we stick to the Microsoft technologies. Also volume visualization performance of Java 3D framework, its SDK for graphical card programming doesn’t have well enough support in developer level.

ActiveX control programming is very similar to normal C++ application development. But there exist some several extra steps to add some security certificates to the control for secure downloads through Internet. Apart from development issues this feature enables end user to easily execute an application without any maintenance issue and just over Internet. Behind the plug-in we used OpenSceneGraph (OpenSceneGraph, 2008) graphic library for visualization purpose. This open source graphic programming package contains ready to use file streaming property and rapid development utilities for fast visualization. With this library you can easily create proxy geometry and map several images over this geometry for volume visualization. One another feature of this graphic library is that it enables automatic system capability detection. Depending on the system configuration and the graphic card’s capabilities it can select the best visualization method behind the scene. i.e. if the user has a programmable graphic card then it activates the shader programming otherwise using the normal graphic pipeline. Also this library provides mouse manipulator methods to enable the user interaction. Depending on the mouse movement initiated by end user, you can easily rotate the proxy volume geometry to make visualization from different perspectives. To achieve all these features, you just create an activeX control, which is a simple compressed application package with statically linked dependency files. When the client, end-user, browses a specific web page, then this application package automatically downloaded and executed inside the web browser. Necessary image set that are required for proxy geometry is separately downloaded from a URL based Internet address as it is done for static image based experiment visualization. Once all the images related to a specific experiment are downloaded then they are mapped on the proxy geometry for volume visualization.

CHAPTER SIX
CONCLUSION

6.1 Summary and Contributions

Technological developments within the recent decades show that it is mandatory to make interdisciplinary research and collaboration to have a global perspective over problem analyses. Developments in each research fields make it difficult for a scientist from a specific domain to be dominant in his/her own field without using new tools and research results. Also it requires huge amount of man/month work to analyze and adapt the benefits of one domain to other. This thesis is an example of such collaboration where the latest developments and research results are effectively integrated to solve problems that are impossible to be solved before.

For many years the theory of electromagnetic wave propagation is known and it was not possible to solve the equations of this theory by hand. Mainly it takes pages of formulation and requires high precision computations which usually have the risk of numerical errors because of “by hand” computation. Developments in the computer science field make it possible to solve complex equations with error free computation. On the other hand the tools, mainly the computers that are used in computer science field also have some limitations. These limitations are generally affecting the computation power, memory size and computation precision.

In this thesis we demonstrate a new set of equations that are used to solve electromagnetic wave propagation theory. Main difference of those equations from the existing ones is their form that is suitable for existing computer technology. These equations are explicit, so it is possible to apply effective numerical computations. On the other hand to overcome numerical precision limitations, some parts of those equations are in the form that makes it possible to apply symbolic computation.

In the second chapter, we demonstrate how those equations are generated. Details of the formulation are cited to a paper that we published during the development of this thesis. Numerical solution, moreover the numerical result is not enough to analyze the output. Mainly because of massive data which is impossible to be interpreted. For this reason we explore existing visualization techniques and in the third chapter we deepen into another problem called volume visualization. With the existing desktop computers it was not possible to efficiently compute and visualize the result. For this reason we adapt the recent developments in computer graphics fields which are mainly used for real-time visualization purposes by computer games. Moreover existing scientific computation tools are integrated with computer graphic hardware for parallel computation of the time varying data and visualizing it at the same time. Forth chapter is dedicated for the recent parallel computation and visualization techniques, their constraints and the solutions. During our computations we used single precision floating point numbers because of the existing capability of the graphic cards that we have. With this constraint we already get satisfactory visual results for simple cases of the wave propagation theory. Also symbolic computation engine had some memory limitations that prevent us to simulate more complex cases. In the fifth chapter we demonstrate a framework that makes it possible to share the computation results with other scientists. Since those computations require specific hardware, with this collaboration platform it becomes possible to share the results.

6.2 Future Works

Efficient processing capabilities of desktop computers are not good enough for scientific computations. On the other hand more specific computers that are designed for scientific computation requires special programming skills and the cost of those computers are very high for academic world. With the recent developments in computer entertainment sector, it becomes possible to use commodity computer graphics cards for scientific computation. Such graphic cards getting more popular and so the price decreases. Now every new computers even laptop become a scientific computation platform because of such graphic cards.

The computation and visualization method that we demonstrated within this thesis is applicable on the other fields that require massive data processing. Also it becomes essential to parallelize the parallel processing units since every computer becomes a single parallel processing machine. Like the grid technology that is used to benefit from the CPU power of idle computers, it is possible to use GPU power by developing a special framework. Moreover specific high level scientific computation languages like Matlab’s can be developed for GPU based computation.

REFERENCES

Bracewell, R. (1999). *The Fourier Transform and Its Applications* (3rd ed.). New York: McGraw-Hill.

Canny J. (1986). A Computational Approach to Edge Detection, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *8* (6), 679-698.

Cohen, G.C., Heikkola, E., Joly, P., & Neittaanmaki, P. (2003). *Mathematical and Numerical Aspects of Wave Propagation*. Berlin: Springer.

CUDA Zone—Learn about CUDA, (n.d.). Retrieved August 16, 2008, from http://www.nvidia.com/object/cuda_what_is.html

Dienlesaint, E., & Royer, D. (1980). *Elastic Waves in Solids*. Chichester: John Wiley and Sons.

DirectX, (n.d.). Retrieved April 05, 2008, from http://www.microsoft.com/directx/

Elvins, T.T. (1992). A Survey of Algorithms for Volume Visualization. *Computer Graphics*, Volume 26, Number 3, pages 194–201.

Fedorov F.I. (1968). Theory of Elastic Waves in Crystals. *Journal of Applied Crystallography 1* (5), 328-335.

FFTW, Library for computing the discrete Fourier transform, (n.d.). Retrieved March 30, 2008, from http://www.fftw.org

Frigo, M. & S. G. Johnson. (1998). FFTW: An Adaptive Software Architecture for the FFT. *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, Vol. 3, pp. 1381-1384.

Goldberg, D. (1991). *What Every Computer Scientist Should Know About Floating-Point Arithmetic*. New York: ACM Computing Surveys.

Graff K.F. (1975). *Wave Motion in Solids*. Oxford: Oxford University Press.

Grid, GridCafe, (n.d.). Retrieved May 05, 2008, from http://gridcafe.web.cern.ch/gridcafe/whatisgrid/whatis.html

Humphreys G., Buck I., Eldridge M., & Hanrahan P. (2000). Distributed rendering for scalable displays. *Proceedings of the 2000 ACM/IEEE conference on Supercomputing*.

Humphries S. (1997). *Field Solutions on Computers*. USA: CRC Press.

Kasap, M. (2003). *Three Dimensional Simulations of Complex Structures*. Dokuz Eylül Üniversitesi, Fen Bilimleri Enstitüsü.

Kaufman A. (1994). Voxels as a Computational Representation of Geometry. *In The Computational Representation of Geometry*. SIGGRAPH '94 Course Notes.

Knuth D. (1984), *The Texbook*, Massachusetts: Addison-Wesley.

Levoy M. (1988). Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, *8* (3), 29-37.

Mapple, Computational tool, (n.d.). Retrieved March 30, 2008, from http://www.maplesoft.com/products/Maple11/professionals/main.aspx

Mathematica, Mathematical computing environment, (n.d.). Retrieved March 30, 2008, from http://www.wolfram.com/products/mathematica/features/

Matlab, High-level technical computing language and interactive environment, (n.d.). Retrieved March 30, 2008, from http://www.mathworks.com/products/matlab/description1.html

Max N. (1995). Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics, 1* (2), 99-108.

Microsoft Windows, (n.d.). Retrieved March 30, 2008, from http://www.microsoft.com/windows/default.aspx

Netlib, Mathematical software collection, (n.d.). Retrieved March 30, 2008, from http://www.netlib.org/na-net/

Numerical Recipes, (n.d.). Retrieved March 30, 2008, from http://www.nr.com/

NVIDIA - World Leader in Visual Computing Technologies, (n.d.). Retrieved August 16, 2008, from http://www.nvidia.com/page/home.html

OpenGL, (n.d.). Retrieved April 05, 2008, from http://www.opengl.org/

OpenSceneGraph, *(n.d.).* Retrieved February 10, 2008, from http://www.openscenegraph.org

Phong, B.T. (1973). *Illumination of Computer-Generated Images*, Department of Computer Science, University of Utah, UTEC-CSs-73-129.

Ramo, S., Whinnery, J.R. & Duzer, T. (1994). *Fields and Waves in Communication Electronics*. New York: John Wiley and Sons.

Real-Time Volume Graphics, Siggraph 2004 Course notes, (2004). Retrieved 20 August, 2007, from http://www.vrvis.at/via/resources/course-volgraphics-2004/

Salomon D., G. Motta & D. Bryant. (2007). *Data Compression: The Complete Reference*. London: Springer-Verlag.

Semcad, (n.d.). Retrieved April 05, 2008, from http://www.semcad.com/simulation/

Tesla, High performance computing, (n.d.). Retrieved March 30, 2008, from http://www.nvidia.com/object/tesla_computing_solutions.html/

The Visible Human Project, Library of Medicine, United States National Institutes of Health 2003, (2003). Retrieved April 05, 2008, from http://www.nlm.nih.gov/research/visible/visible_human.html/

Yakhno, V.G., Yakhno, T.M., & Kasap M. (2003). Simulation of Electromagnetic Wave Propagation in Anisotropic Media. *Selcuk Journal of Applied Mathematics, Vol. 4, N. 2*, 113-122.

Yakhno, V.G., Yakhno, T.M., & Kasap M. (2006). A novel approach for modelling and simulation of electromagnetic waves in anisotropic dielectrics. *International Journal Solids Structures 43*, 6261–6276.

APPENDIX A

MATLAB CODE TO SOLVE SYMBOLIC EQUATIONS

Following MATLAB code is used for testing the symbolic calculations performed in the experiments. Our symbolic calculation module that we used in the final application is part of the following code. Below two functions are also provided which we used to test if the user input matrices are satisfying all the conditions.

A.1 Positive Definite Function

```
% Function to test if a Matrix is positive definite or not
% If positive definite then returns zero
% Input matrix must be 3x3 matrix
function result = mkIsPositiveDefinite(Matrix)
result1 = det(Matrix(1,1));
disp(sprintf('Positive definiteness: 1th subsquare det
result: %f', result1));
result2 = det(Matrix(1:2,1:2));
disp(sprintf('Positive definiteness: 2th subsquare det
result: %f', result2));
result3 = det(Matrix(1:3,1:3));
disp(sprintf('Positive definiteness: 3th subsquare det
result: %f', result3));

if (result1 <=0) | (result2 <=0) | (result3 <=0)
    result = 1;
else
    result = 0;
end
```

A.2 Test Function for Inverse Square Root of Epsilon

```
% Function to test the properties of Inverse Square root of
% Epsilon and Square root of epsilon.
% Function returns 0 if all properties satisfied else it returns
% a error value greater then 0. Error value indicates which
% property doesn’t satisfy.
% Input parameter is epsilon.
 function result = TestInvSqrtEpProperties(Ep)
InvSqrtEp = inv(sqrtm(Ep));
SqrtEp = sqrtm(Ep);
result = 1;
```

```
% Check Property 1: Ep^(1/2) and Ep^-(1/2) positive symmetric
result = mkIsPositiveDefinite(InvSqrtEp);
result = result + mkIsPositiveDefinite(SqrtEp);
if result > 0
    result = 1;
    disp('ERROR : Ep^(1/2) and Ep^-(1/2) positive symmetric')
    return
end
if (tril(SqrtEp) ~= triu(SqrtEp)) | (tril(InvSqrtEp) ~=
triu(InvSqrtEp))
    result = 1;
    disp('ERROR : Ep^(1/2) and Ep^-(1/2) positive symmetric')
    return
end
disp('OK : Ep^(1/2) and Ep^-(1/2) positive symmetric')

% Check Property 2: Ep^(1/2) = (Ep^-(1/2))^-1
if (single(inv(InvSqrtEp)) ~= single(SqrtEp))
    result = 2;
    disp('ERROR : Ep^(1/2) = (Ep^-(1/2))^-1')
    return
end
disp('OK : Ep^(1/2) = (Ep^-(1/2))^-1')

% Check Property 3: Ep^-(1/2)*Ep = Ep^(1/2)
if (single(InvSqrtEp * Ep) ~= single(SqrtEp))
    result = 3;
    disp('ERROR : Ep^-(1/2)*Ep = Ep^(1/2)')
    return
end
disp('OK : Ep^-(1/2)*Ep = Ep^(1/2)')

% Check Property 4: (Ep^-(1/2))' = Ep^-(1/2)
if (InvSqrtEp' ~= InvSqrtEp)
    result = 4;
    disp('ERROR : (Ep^-(1/2))'' = Ep^-(1/2)')
    return
end
disp('OK : (Ep^-(1/2))'' = Ep^-(1/2)')
```

A.3 MATLAB Code to Find Explicit Equation for E

```
clc
clear all;

e = [0 0 1]';
Eps = zeros(3,3);

%Experiment 01
Eps(1, 1) = 1;
Eps(1, 2) = 0;
Eps(1, 3) = 0;
Eps(2, 1) = 0;
Eps(2, 2) = 1;
Eps(2, 3) = 0;
Eps(3, 1) = 0;
Eps(3, 2) = 0;
Eps(3, 3) = 1;

V1           = sym('V1', 'real');  % -Create symbolic variables
V2           = sym('V2', 'real');  %  that will be used to create
the S(V)
V3           = sym('V3', 'real');  %
tm           = sym('tm', 'real');  %  matrix

% Calculate the Square root and Inverse Square root of Epsilon
% User selects epsilon or inv.sqrtEps. matrix...
 disp('Calculating inverse square root');

 [Egvc, Egvl] = eig(Eps);
 P  = Egvc;
 Pi = inv(Egvc);
 M = Egvl;
 Mh = sqrt(M);
 Eh = P * Mh * Pi;
 Ehi = inv(Eh);

 if strcmp(class(Ehi), 'sym')
     InvSqrEps = simple(Ehi);
 else
     InvSqrEps = Ehi;
 end

% Create the matrix S(V)
% S(V) is symbolic and 3x3 matrix.
disp('Calculating S');
S            = [   V2^2 + V3^2,       -V1 * V2,         -V1 * V3;
                      -V1 * V2,    V1^2 + V3^2,         -V2 * V3;
                      -V1 * V3,       -V2 * V3,     V1^2 + V2^2];

% To find Q and D, first calculate the multiplication of
% InverseSquareRootofEpsion, S(Y) and again
InverseSquareRootofEpsion.
% Use simplify method to make symbolic simplification over the
result. "Simplify" method takes an symbolic input then produces
the simplified version of the input parameter as an output.
% store this temporal result (simplified version of the
multiplication result) in the variable A.
disp('Calculating A');
A = simple(InvSqrEps * S * InvSqrEps);

% Find the eigen vectors end eigen values of the multiplication
result, which is stored in temporal variable A
% Here "eig" method used. Eig methods takes one input and
produces two output. First output is EigenVector, second
parameter is the Eigenvalue of the input parameter.
disp('Calculating Eigen Value and Vector of A');
[EigVecA, EigValA]        = eig(A);

% Find D and Q. There is no need to make any calculation.
Because D is equal to the EigenValue of the multiplication
```

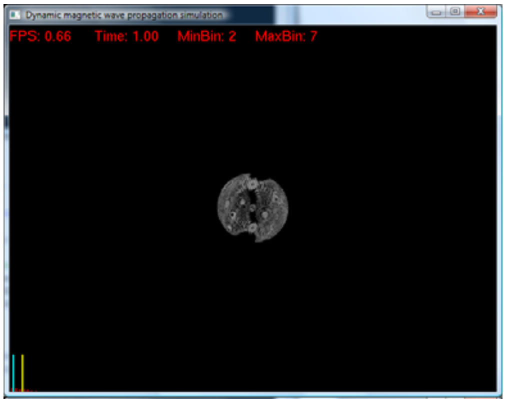
```
result A (temporal result). Q is the EigenVector of the
multiplication result A (temporal result). But before assigning
[EigVecA, EigValA] values to Q and D. We make a simplification.
Again we use "simplify" method for these operation.
Simplification is optional operation, which reduces the
complexity of the symbolic result. And make the calculations
faster.
% Here Q and D are symbolic, 3x3 matrices.
% Each !!!column!!! of the Q and D matrices are vectors. (i.e.
first column Q matrix is the first eigen vector.)
disp('Calculating D');
D       = simple(EigValA);

disp('Calculating Q');
Q       = simple(EigVecA);


% We normalize the vectors in Q matrix.
% To do the normalization operation we find the length of each
vector in the matrix. Then divide each component of the vector
to its length.
% I.e: Q(:, 1)  gives the first column of the 3x3 matrix Q.
Which means the first vector of the 3x3 matrix Q. sqrt(sum(Q(:,
1).^2)) command find the length of first vector

Q(:, 1) = Q(:, 1) ./ sqrt(sum(Q(:, 1).^2));
Q(:, 2) = Q(:, 2) ./ sqrt(sum(Q(:, 2).^2));
Q(:, 3) = Q(:, 3) ./ sqrt(sum(Q(:, 3).^2));

disp('Calculating Transpose of Q');
transQ = Q.';

% After normalization of Q we multiply the
InverseSquareRootofEpsilon with Q to fint the Variable T.
disp('Calculating T');
T       = InvSqrEps * Q;

transT = T.';

% For future calculations we need the eigen values as a single
vector. In matlab eigen values are the diagonal values of the
resulting matrix of "eig" command. So we extract the diagonal of
D which contains the EigenValues in 3x3 matrix. Extraction
operation is done with "diag" command. "Diag" command takes
square matrix and returns the diagonal elements as a vector. We
store the result of this operation in the variable named diagD
for furthure usage.
diagD   = diag(D);

% Find the Transpose of T and multiply the result with p_e
variable. p_e variable holds the direction vector e. Final
result of this calculation stored in the variable named TransTe.
transTe  = T.' * e;

% Calculate the vector Y. Y is a 3x1 matrix or a vector.
Components of Y is calculated according to the values of
components of p_diagD. We know that one of the components of D
must be zero. For the zero value of component of D requires
```

```
special calculations for corresponding component of Y.
for i = 1:3
    if diagD(i) == 0
        Y(i) = transTe(i);
    else
        Y(i) = transTe(i) * (cos(sqrt(diagD(i))* tm) );
    end
end

% After the calculations above Y is a row vector. But all our
vectors are column vectors so we take the transpose of Y.  Then
we take the negative of Y
Y       = -1 * Y.';

% Fourier image of the Electromagnetic field is constructed.
% We use two time simplification to reduce the complexity of
resulting value. Sometimes it is useless to make more then one
simplification.

% Image of the fourier transform
FFT_E    = simple(simple(T * Y));
FFT_E    = simplify(FFT_E);
```

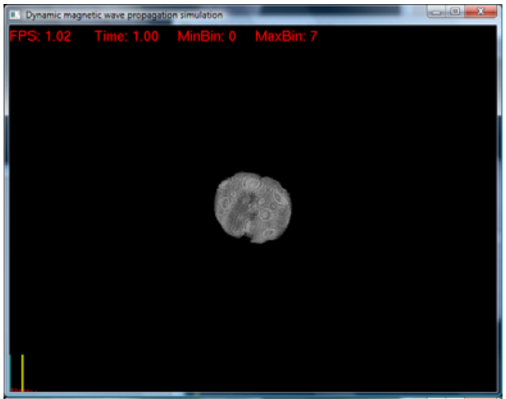
APPENDIX B
3D VISUALIZATION OF PROPAGATION

Using the mentioned techniques in this thesis for real-time visualization of the electromagnetic wave propagation requires special hardware. The time that we perform our experiments it was difficult to obtain such devices easily. In this section we demonstrate some screenshots captured from the real-time visualization application. All experiments represent the propagation of electric field with specific component namely $E_1, E_2\ and\ E_3$. Two distinct column of the page are used to demonstrate the result with different histogram based filtering parameters. With higher histogram values, only the high frequency waves are visualized with less detail. With low histogram values, low frequency waves visualized as well as the data noise resulting from Fourier transformation.

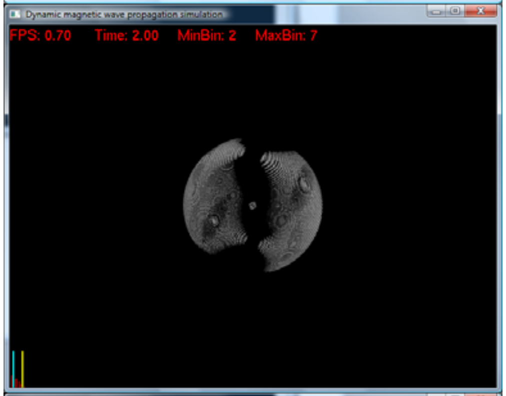
Experiment: 01
Epsilon = [1 0 0]
[0 1 0]
[0 0 1]
Component = E3



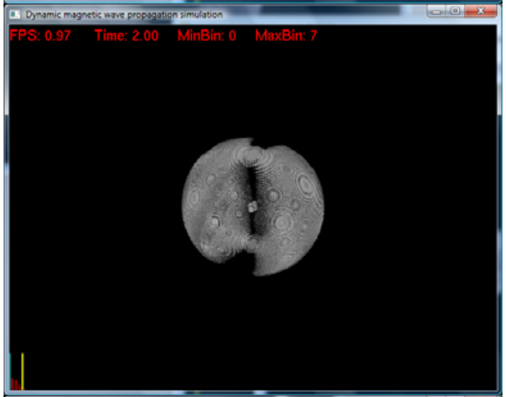
FPS: 0.66 Time: 1.00 MinBin: 2 MaxBin: 7



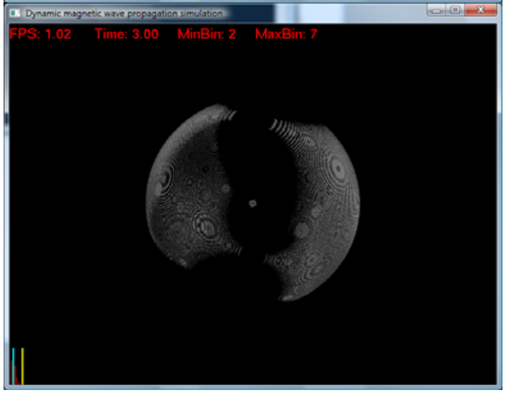
FPS: 1.02 Time: 1.00 MinBin: 0 MaxBin: 7



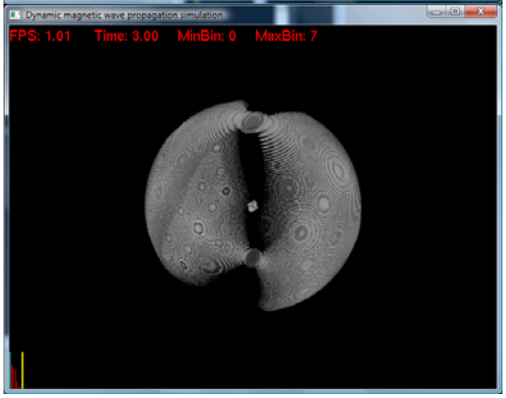
FPS: 0.70 Time: 2.00 MinBin: 2 MaxBin: 7



FPS: 0.97 Time: 2.00 MinBin: 0 MaxBin: 7

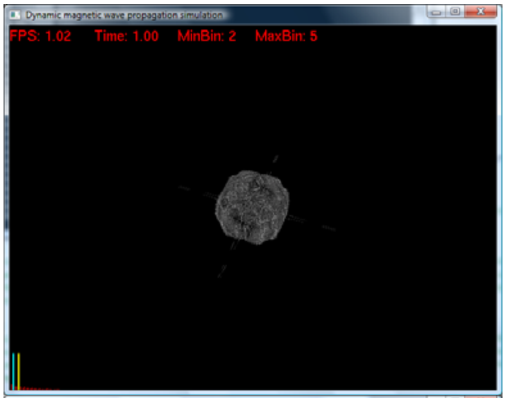


FPS: 1.02 Time: 3.00 MinBin: 2 MaxBin: 7

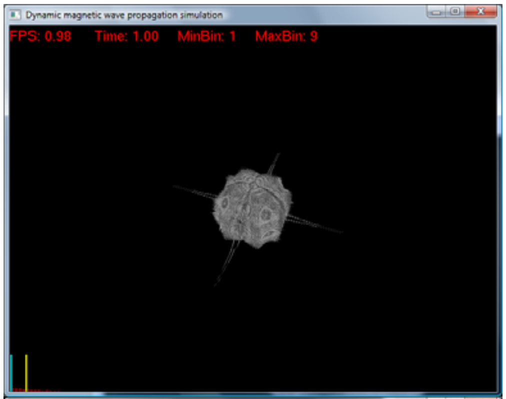


FPS: 1.01 Time: 3.00 MinBin: 0 MaxBin: 7

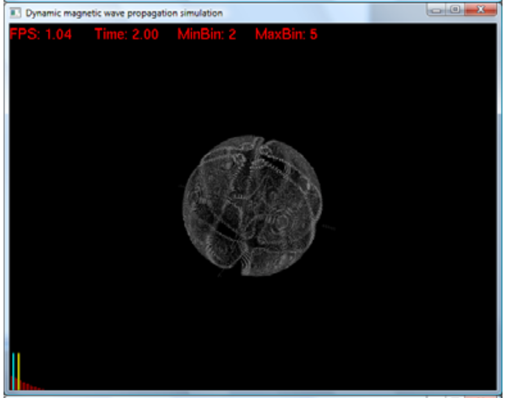
Experiment: 01
Epsilon = [1 0 0]
[0 1 0]
[0 0 1]
Component = E1



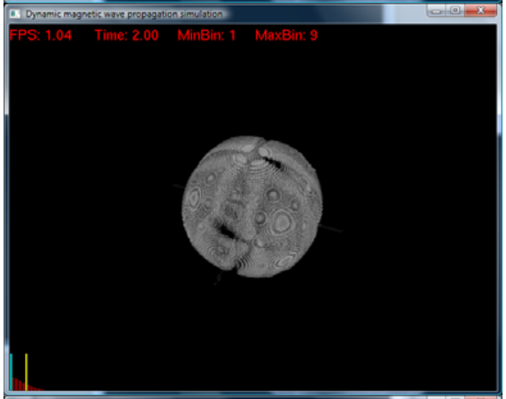
FPS: 1.02 Time: 1.00 MinBin: 2 MaxBin: 5



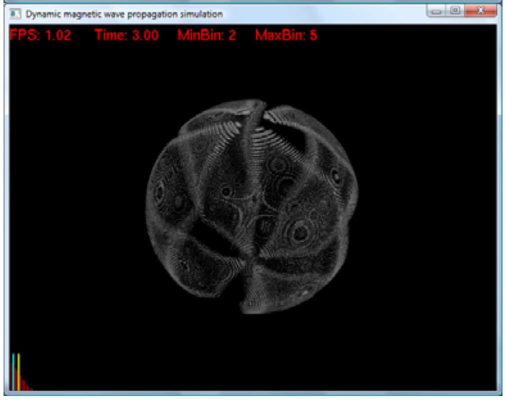
FPS: 0.98 Time: 1.00 MinBin: 1 MaxBin: 9



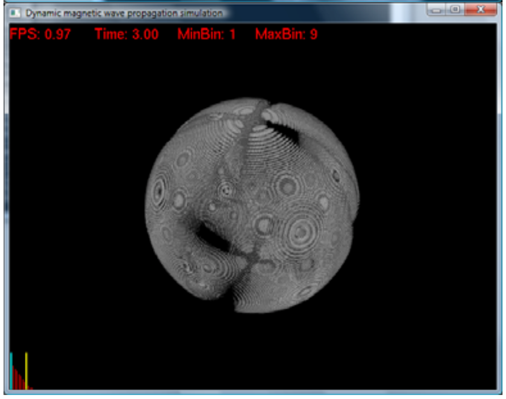
FPS: 1.04 Time: 2.00 MinBin: 2 MaxBin: 5



FPS: 1.04 Time: 2.00 MinBin: 1 MaxBin: 9

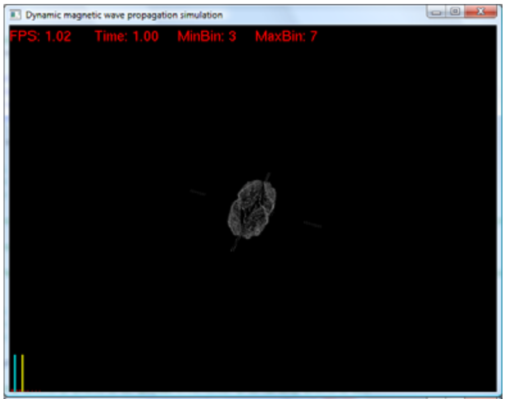


FPS: 1.02 Time: 3.00 MinBin: 2 MaxBin: 5

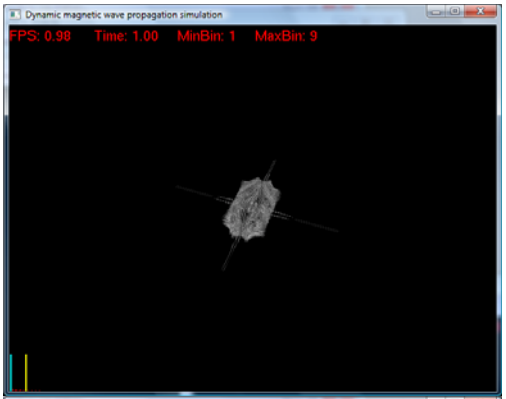


FPS: 0.97 Time: 3.00 MinBin: 1 MaxBin: 9

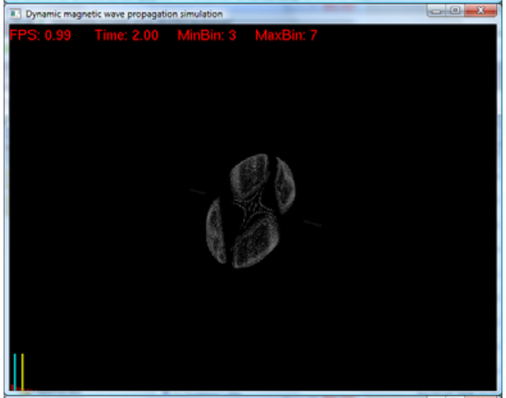
Experiment: 02
Epsilon = [1 0 0]
[0 1 0]
[0 0 3]
Component = E1



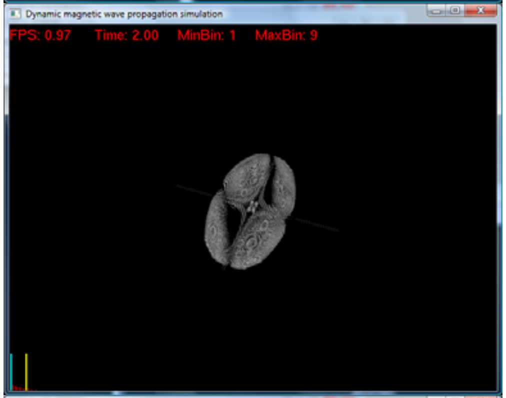
FPS: 1.02 Time: 1.00 MinBin: 3 MaxBin: 7



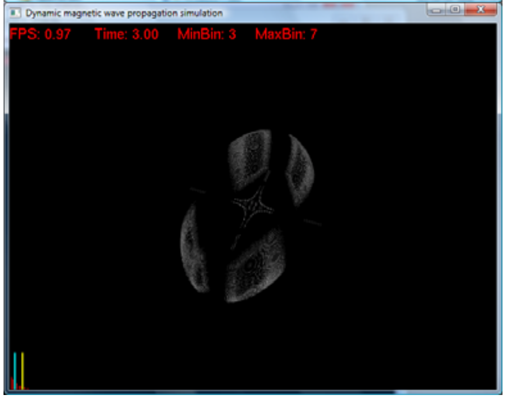
FPS: 0.98 Time: 1.00 MinBin: 1 MaxBin: 9



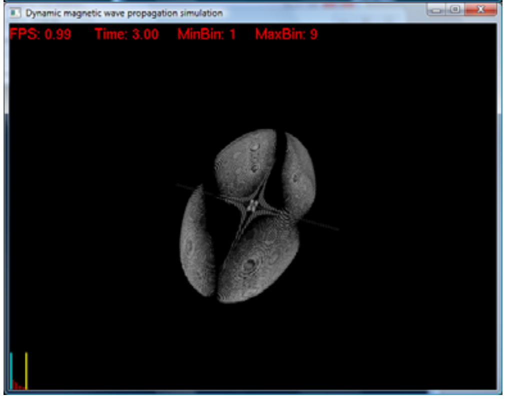
FPS: 0.99 Time: 2.00 MinBin: 3 MaxBin: 7



FPS: 0.97 Time: 2.00 MinBin: 1 MaxBin: 9

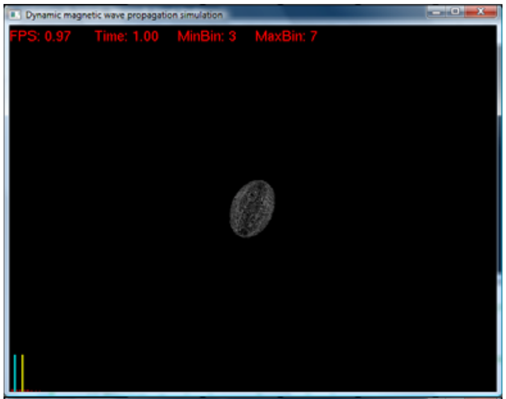


FPS: 0.97 Time: 3.00 MinBin: 3 MaxBin: 7

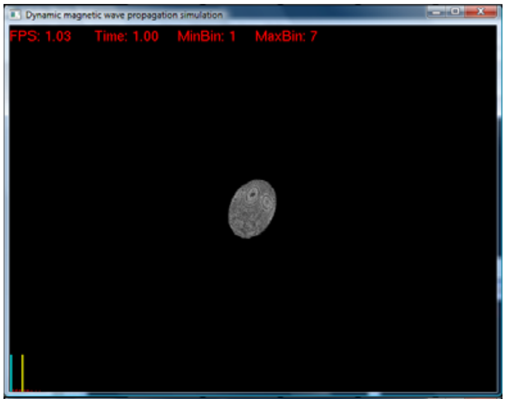


FPS: 0.99 Time: 3.00 MinBin: 1 MaxBin: 9

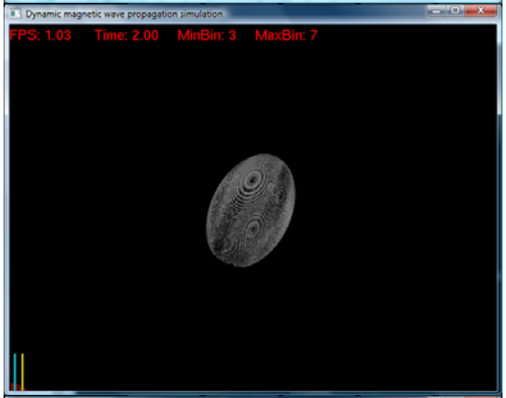
Experiment: 02
Epsilon = [1 0 0]
[0 1 0]
[0 0 3]
Component = E3



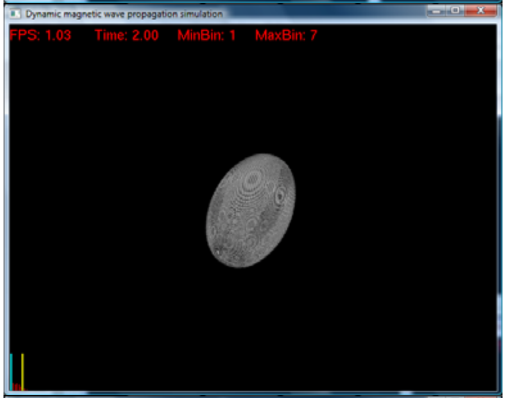
FPS: 0.97 Time: 1.00 MinBin: 3 MaxBin: 7



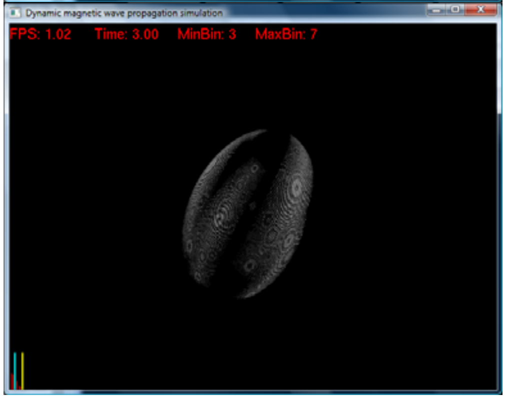
FPS: 1.03 Time: 1.00 MinBin: 1 MaxBin: 7



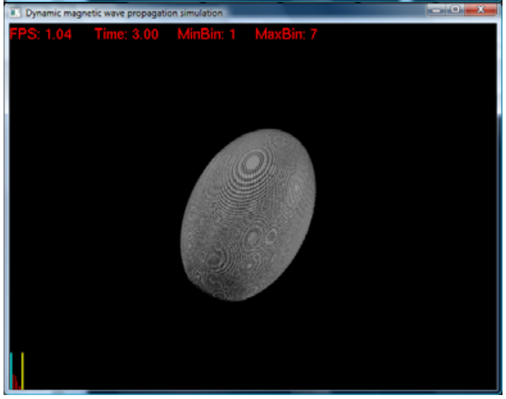
FPS: 1.03 Time: 2.00 MinBin: 3 MaxBin: 7



FPS: 1.03 Time: 2.00 MinBin: 1 MaxBin: 7



FPS: 1.02 Time: 3.00 MinBin: 3 MaxBin: 7



FPS: 1.04 Time: 3.00 MinBin: 1 MaxBin: 7

APPENDIX C
SAMPLE CUDA COMPUTATION

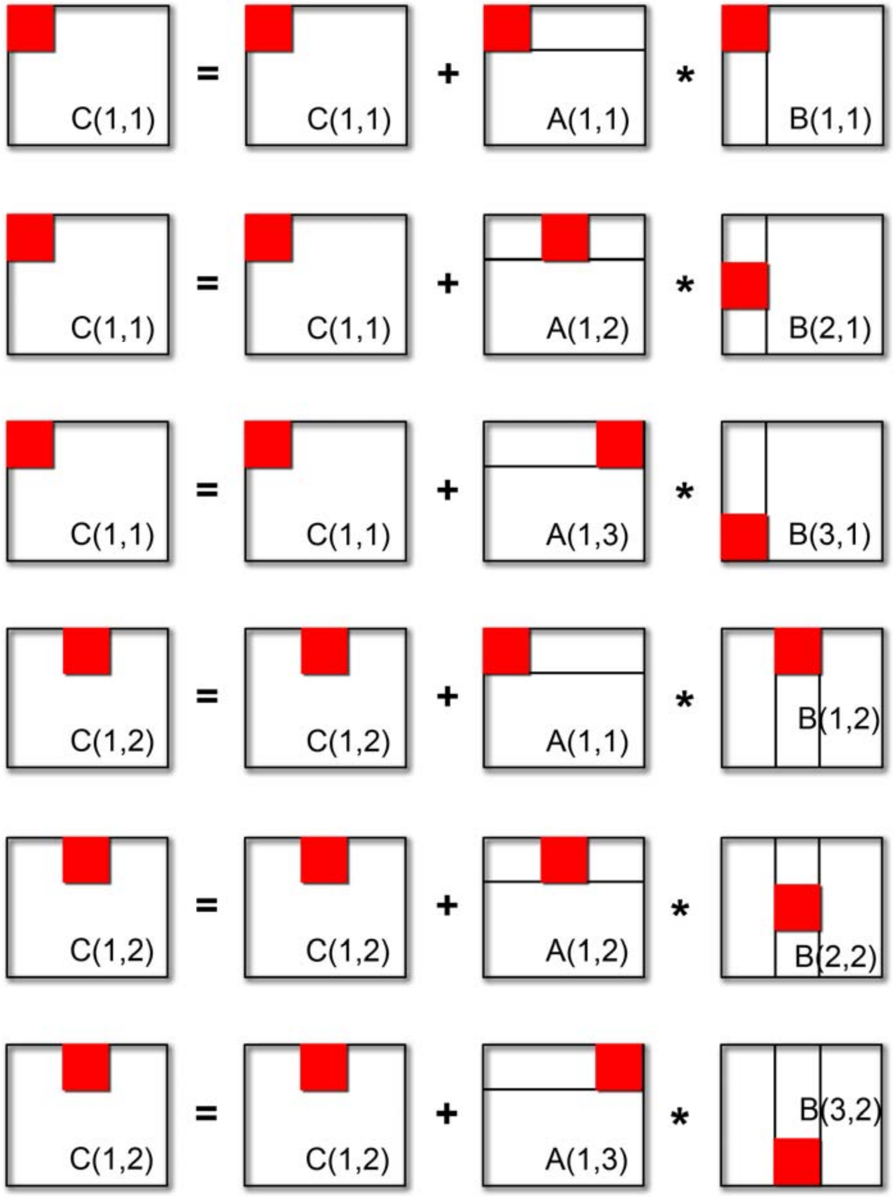
C.1 Matrix Multiplication by Sub-blocks

Consider A, B and C are N by N matrices with b by b subblocks. Aim is to find

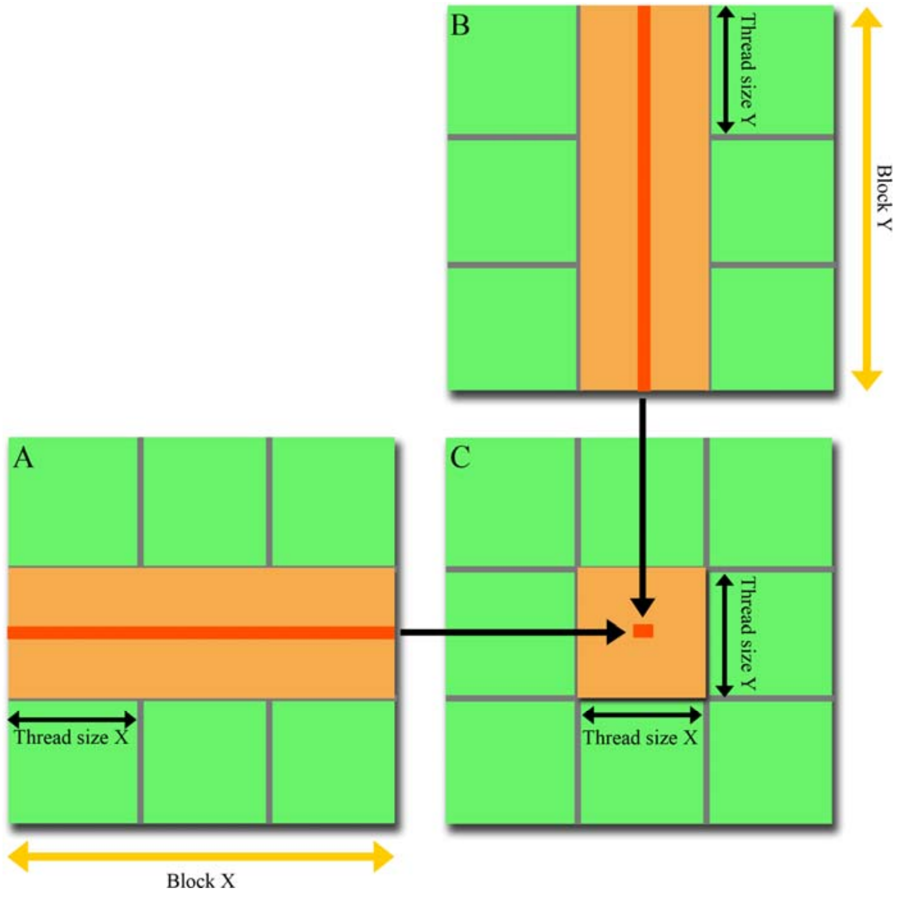
C = A * B

This operation is performed by multiplying sub blocks as it is schematically represented in the figure below.

```
// Pseudo code for sub block based fast matrix multiplication,
//n is number of blocks
for i = 1 to n
        for j = 1 to n
                {read block C(i,j) into L1 memory}
                for k = 1 to n
                        {read block A(i,k) into L1 memory}
                        {read block B(k,j) into L1 memory}
                        C(i,j) = C(i,j) + A(i,k) * B(k,j) { multiply matrix blocks}
                        {write block C(i,j) back to global memory}
```


C(1,1)
=
C(1,1)
+
A(1,1)
*
B(1,1)
C(1,1)
=
C(1,1)
+
A(1,2)
*
B(2,1)
C(1,1)
=
C(1,1)
+
A(1,3)
*
B(3,1)
C(1,2)
=
C(1,2)
+
A(1,1)
*
B(1,2)
C(1,2)
=
C(1,2)
+
A(1,2)
*
B(2,2)
C(1,2)
=
C(1,2)
+
A(1,3)
*
B(3,2)

C.2 CUDA Based Fast Matrix Multiplication



```
// Some part of this code is taken from Nvidia CUDA1.0 toolkit samples, 2008.
// Setup the execution configuration
dim3 dimBlock(WIDTH, WIDTH);
dim3 dimGrid(1, 1);
// Launch the device computation threads!
MatrixMulKernel<<<dimGrid, dimBlock>>>(A, B, C);
// Matrix multiplication kernel – thread specification
__global__ void MatrixMulKernel(Matrix A, Matrix B, Matrix C)
{
    // 2D Thread ID
    int tx = threadIdx.x;
    int ty = threadIdx.y;

    // Pvalue is used to store the element of the matrix
    // that is computed by the thread
    float Pvalue = 0;
    for (int k = 0; k < M.width; ++k)
    {
        float Melement = M.elements[ty * M.pitch + k];
        float Nelement = Nd.elements[k * N.pitch + tx];
        Pvalue += Melement * Nelement;
    }
    // Write the matrix to device memory; each thread writes one element
    P.elements[ty * P.pitch + tx] = Pvalue;
}
```