Hardware-based Parallel Computing for Real-time Simulation of Soft-object Deformation

HARDWARE-BASED PARALLEL COMPUTING FOR REAL-TIME SIMULATION OF SOFT-OBJECT DEFORMATION

BY

RAMIN MAFI, B.Sc.

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

Master of Applied Science (2008) McMaster University

(Electrical & Computer Engineering) Hamilton, Ontario, Canada

TITLE:	Hardware-based Parallel Computing for Real-time Simulation of Soft-object Deformation
AUTHOR:	Ramin Mafi B.Sc., (Biomedical Engineering) B.Sc., (Electrical Engineering) Amirkabir University (Tehran Polytechnic), Tehran, Iran
SUPERVISOR:	Dr. Shahin Sirouspour
CO-SUPERVISOR:	Dr. Nicola Nicolici
NUMBER OF PAGES:	xii, 87

To my beloved family and

in loving memory of my dear uncle:

Vahab Montazery

Abstract

In the last two decades there has been an increasing interest in the field of haptics science. Real-time simulation of haptic interaction with non-rigid deformable object/tissue is computationally demanding. The computational bottleneck in finite-element (FE) modeling of deformable objects is in solving a large but sparse linear system of equations at each time step of the simulation. Depending on the mechanical properties of the object, high-fidelity stable haptic simulations require an update rate in the order of $100 - 1000$ Hz. Direct software-based implementations that use conventional computers are fairly limited in the size of the model that they can process at such high rates.

In this thesis, a new hardware-based parallel implementation of the iterative Conjugate Gradient (CG) algorithm for solving linear systems of equations is proposed. Sparse matrix-vector multiplication (SpMxV) is the main computational kernel in iterative solution methods such as the CG algorithm. Modern microprocessors exhibit poor performance in executing memory-bound tasks such as SpMxV. In the proposed hardware architecture, a novel organization of on-chip memory resources enables concurrent utilization of a large number of fixed-point computing units on a FPGA device for performing the calculations. The result is a powerful parallel computing platform that can iteratively solve the system of

equations arising from the FE models of object deformation within the timing constraint of real-time haptics applications.

Numerical accuracy of the fixed-point implementation, the hardware architecture design, and issues pertaining to the degree of parallelism and scalability of the solution are discussed in details. The proposed computing platform in this thesis is successfully employed in a set of haptic interaction experiments using static and dynamic linear FE-based models.

Acknowledgements

I would like to express my great gratitude to my supervisor, Dr. Shahin Sirouspour for his invaluable suggestions, support and encouragement throughout my graduate program at McMaster university. In addition, I give my special thanks to my co-supervisor, Dr. Nicola Nicolici for his constructive comments and suggestions during the development of the hardware design in this project. I also would like to thank my other committee members, Dr. Patriciu and Dr. Shirani for their interest in my work.

My deepest gratitude goes to my family, for their unwavering love and encouragement throughout my life.

I am indebted to my mentor, friend and uncle, Vahab Montazery, whom without his encouragement and support, my ambition to study abroad could hardly be realized. Although he is no longer with us, he is forever remembered.

I would like to acknowledge all my friends, specially Fabiola De Vierna, who supported me and made my time a lot more fun.

Additionally thanks to my friends and lab mates, Brian Moody and Behzad Mahdavikhah, who were in parallel involved in this project and had a significant part to accomplish it.

The generous support by Quanser Consulting Inc., The Health Technology Exchange (HTX), and the Ontario Centres of Excellence (OCE) for this research is greatly appreciated.

Contents

Abstract **iv**

Acknowledgements **vi**

1 Introduction **1**

1.1 Problem Definition 3

1.2 Thesis Objectives and Contributions 6

1.3 Thesis Outline 8

1.4 Related Publications 9

2 Literature Review **10**

2.1 Deformable Body Modeling 10

2.1.1 Mass-Spring Model 13

2.1.2 Finite Element Model 16

2.2 Real-time Simulation of FE-based Models 17

3 Finite Element Formulation **20**

3.1 Problem Statement 21

3.2 Differential Equations Describing Static Equilibrium 24

3.3 Geometric Discretization 25
3.4 Interpolation Functions 26
3.5 Elemental Stiffness Matrix and Assemblage 30
3.6 Dynamic FE Analysis 33
3.6.1 Newmark Integration 35
3.7 Modified Force Vector 36

4 Conjugate Gradient Algorithm **39**
4.1 A Comparison Between Different Linear Solvers 40
4.2 Fixed-point Implementation of the CG method 41
4.3 Preconditioning 46
4.4 Data Dependence Analysis 50

5 Hardware Architecture **53**
5.1 Data Structures 54
5.2 Sparse Matrix by Vector Multiplication 55
5.3 Hardware Parallelism 57
5.4 Memory Structure 61
5.5 Critical Path 64
5.6 Scalability 65

6 Performance Analysis and Experimental Results **67**
6.1 Hardware Accelerator Performance 68
6.2 Experimental Platform 70
6.2.1 Hardware-based Accelerator for the CG algorithm 71
6.2.2 Haptic Control and Communication Process 71

6.2.3 Collision Detection and Graphics 72
6.2.4 Experimental Results 73

7 Conclusions and Future Work **75**

A PROCStar II Technical Specifications **78**

List of Figures

1.1 Haptic Tools in Surgery Simulation 2
2.1 Spline Parametric Surface 12
2.2 2D Chainmail Configuration 13
2.3 Mass-Spring System 14
3.1 General 3D Body For Structural Analysis 21
3.2 Typical Finite Elements Geometries 25
3.3 A Four-Node Tetrahedron 32
3.4 FE Procedure 33
3.5 Calculating the Modified Force Vector 38
4.1 Static-Scaling 43
4.2 Error Vector v.s. the Exact Solution in CG 45
4.3 Error Vector v.s. the Exact Solution in PCG 49
4.4 Computation Flow in the CG Algorithm 52
5.1 3x3 Block Structure of Non-zero Elements 55
5.2 Graph Partitioning Techniques Used in Matrix Permutation 57
5.3 First Level of Parallelization 58
5.4 Second Level of Parallelization 59
5.5 The Connection Of MAC Units to Memory Blocks 60

5.6 Matrix Partitioning For Increased Parallelism 61
5.7 Third Level of Parallelization 61
5.8 Basic Memory Structure For NZ Components Of Matrix 63
5.9 Basic Memory Structure For Vectors In The CG Algorithm 64
5.10 Critical Path 65
6.1 Setup Block Diagram 70
6.2 Haptic Control and Communication Process 73
6.3 Experimental Results 74
A.1 PROCStar II Block Diagram 78

Chapter 1

Introduction

Real-time computer simulations of virtual environments have found numerous applications in recent years [1–4]. Examples include computer games, virtual reality-based training simulators, computer-aided virtual prototyping and design, and computer-assisted surgery. A growing number of such applications are evolving from mere graphical representation of the virtual environment into more interactive forms of simulations involving force feedback [5,6]. They often employ a haptic device, a bidirectional human-machine interface that allows users to feel object interaction forces generated based on mathematical models of the virtual world.

Early haptics systems were mostly limited to the simulation of rigid objects. As such, physics-based modeling of interaction with rigid objects for real-time simulation has been extensively studied in the past [5,7,8]. More recent applications of haptics are in *medical training simulators* and *computer-assisted medical interventions* involving non-rigid deformable objects. Fig. 1.1 provides some examples of haptic applications in these areas.

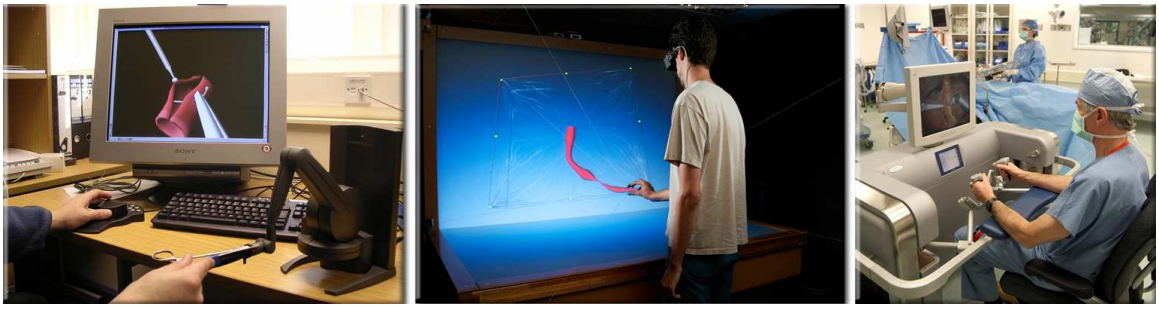


Figure 1.1: Haptic tools in surgery simulation *(Figure courtesy of Geobench project, data ©CEA-DAM; Richard Holbrey)*

As an example, medical students can develop their operational skills by practicing surgical procedures in virtual environments. Such systems provide a number of advantages over traditional forms of training. First, there is no real consequence to making mistakes since the training is performed in a virtual environment. Second, the trainee's progress can be quantitatively measured and the task difficulty level can be adjusted accordingly. Finally, the trainee can be actively assisted by providing corrective force-feedback through the haptic interface.

Real-time virtual modeling of medical procedures, with or without haptics, can also be indispensable in computer-assisted surgical systems where the ability to integrate multiple sources of information for planning and execution of the intended procedure is critical to success. For instance, in percutaneous therapy pre-operative intervention plans often have to be monitored and revised in real-time to account for soft-tissue deformation due to needle insertion as well as organ movement and respiratory motion. Independent of the imaging modality used, intra-operative images can be registered to the pre-operative surgical plan in order to provide the operator with real-time feedback as the procedure unfolds [9].

Such registration has to take into account organ motion and deformation. Similarly, automated needle steering algorithms can greatly benefit from the use of real-time tissue deformation models to continuously correct any deviation from a preplanned needle trajectory [10].

There has been considerable research in modeling of real-time interaction with non-rigid deformable objects, e.g. see [11–14]. In early systems, mass-spring-damper models were popular due to their relative simplicity [15–18]. However, they often produce physically inaccurate and unreliable results. More accurate models of soft-tissue deformation based on the continuum mechanics have also been proposed in the literature [19]. In such approaches, the finite element method (FEM) is usually used to partition the object into smaller elementary shapes and then derive the equations of motion for the mesh nodes resulting in a discretization of the model in the spatial domain [19,20]. Details of FEM will be discussed in Chapter 3. The price of the better accuracy of FEM is a significant increase in computation. In the next section the research problem and the challenges addressed by this thesis will be outlined.

1.1 Problem Definition

Static linear elastic FE analysis requires the solution to a *large* but *sparse* linear system of equations in the form of $\mathbf{KU} = \mathbf{f}$, where $\mathbf{K}$ is a global sparse stiffness matrix, $\mathbf{U}$ is a vector of node deformation and $\mathbf{f}$ is a vector of external forces applied to the nodes. Similarly, an incremental form of such equations can be derived for some nonlinear deformation models in which $\mathbf{K}$ and $\mathbf{f}$ can be functions of the actual deformation [21].

In dynamic analysis, inertial body and velocity dependent damping forces are taken into account. In this case spatial discretization is followed by a temporal discretization using explicit or implicit integration techniques [22]. Explicit integration routines are easier to implement but can suffer from poor numerical stability. Depending on the size and characteristics of the elements, very small integration time steps may be required to maintain stability. Implicit methods, on the other hand, exhibit robust numerical behavior independent of the time step used [22,23]. However, similar to the static case, a large and sparse linear system of equations must be solved at each time step.

FE-based real-time simulation of soft-tissue deformation has been hindered by the large amount of computations that must be completed within each time step, e.g. $1 - 10$ msec if haptic feedback is needed. Various simplifications and workarounds have been proposed in the literature to address this challenge, but mostly at the expense of the fidelity of the simulation [24–27]. A review of the previous work in the literature reveals a prevailing tendency in the research community towards using algorithmic software-based solutions for real-time simulation of object deformation. A notable exception is a recent paper in [28] which proposed element-level parallelization of the computations on a Graphics Processing Unit (GPU). Due to the use of an explicit integration routine in this approach, there is no need for solving a system of equations, significantly reducing the computations at each time step. Explicit integration methods, however, are known for their numerical instability issues which make them rather unreliable for real-time applications using a fixed simulation time step.

Parallel computer grids have been previously employed in solving large-scale

computational problems. However, access to computer girds is limited due to size and cost of operation, and solutions that rely on them cannot be deployed with stand-alone equipment. Moreover, the parallelism achieved by computer grids is limited by the inherent execution overhead associated with programmable processors, which are the core of each grid node.

In this thesis an FPGA-based parallel computing platform is proposed that can greatly speed up the calculations in the FE-based deformation analysis. In the last few years, FPGAs have significantly advanced both in terms of speed and resources, i.e. the number of arithmetic units, programmable logic cells and embedded memories. Compared to programmable processors, including vector processing engines such as GPUs, FPGA-based custom computing architectures can accelerate scientific calculations by tailoring the hardware computing unit to the problem at hand, and by replicating it to exploit parallelism.

The proposed hardware accelerator is a highly parallel implementation of the iterative method of Conjugate Gradient (CG) [29] for solving the system of equations derived from either static or dynamic FE-based deformation analysis. The CG method has been widely used in the literature due to its robust numerical behavior and will be discussed in detail in Chapter 4. It is worth mentioning that hardware-based solutions for sparse matrix by vector multiplication and for solving linear systems of equations have been discussed in a few previous papers, e.g. see [30–32]. These approaches, which use floating point operations, are rather abstract and cannot be scaled to solve practical problems using exiting FPGA devices. Only architectures based on fixed-point operations can truly provide the degree of

parallelization needed for FE-based real-time simulation of soft-tissue deformation. A more detailed study on fixed-point implementation versus floating-point is provided in Sec. 4.2.

The computation speed-up gained by the hardware platform would not only depend on the number of multiplier/adder units employed, but also on their degree of utilization at each clock cycle. A critical challenge for achieving massive parallelization is in addressing the issue of memory access bandwidth. In an improperly structured architecture, the amount of data that can be transferred to the computing units at each cycle can be fairly limited hence reducing the degree of parallelism. A key novelty of the proposed solution is in its ability to continuously supply data operands to a large number of computing units within a scalable architecture. The hardware solver is largely independent of the FE mesh configuration and can be scaled up based on available FPGA resources to solve problems of larger size. The utility of the proposed computing platform is demonstrated in a set of hardware-in-the-loop haptic simulations for interaction with deformable objects using linear FE models.

1.2 Thesis Objectives and Contributions

Although FEM can provide accurate results for soft-object deformation, its real-time applications have been limited due to its computational complexity. Real-time computation requirements of large FE-based models far exceed the capabilities of existing computers. In this thesis, a highly parallel FPGA-based computing architecture for solving the system of equations arising from FE soft-object deformation

models is proposed. The main objectives of the architecture design are *speed*, *scalability* of the solution, and *optimal usage of resources*. Through a high degree of parallelization of the computations, the hardware architecture must meet the real-time response requirement of soft-tissue haptic interaction. The hardware implementation should be compatible with matrices of different sizes and different structures. The available on-chip resources should also be utilized optimally to increase the size of the largest FE mesh that can be processed.

A computational platform meeting the above requirements can be instrumental in the development of real-time planning/assistive tools for medical interventional procedures.

The main premise of this work is that the steps involved in the FE simulation of object deformation can be classified as:
(i) performing algorithmically complex but computationally inexpensive routines.
(ii) solving a large linear system of equations.
The later can be delegated to a new customized parallel-computing platform whereas the former can be simply executed on a conventional computer.

The proposed hardware accelerator is a highly parallel implementation of the iterative method of Conjugate Gradient (CG) [29] for solving the system of equations in (ii).
In summary, the main contributions of the thesis are:

- Proposing a *fixed-point* implementation of the CG algorithm for solving a linear system of equations arising from FE models of deformation.
- Design, implementation and verification of a novel FPGA-based hardware architecture for the CG method that overcomes the fundamental challenge of

memory access bandwidth for massive parallelization of the computations. A key feature of the proposed architecture is its scalability which allows for solving FE problems of different size based on available hardware resources.

- Demonstrating the effectiveness of the proposed FPGA-based parallel computing tool through a set of hardware-in-the-loop haptic simulations for interaction with a deformable object using linear static and dynamic FE models.

It should be emphasized that the application of the proposed micro-architecture is not limited to real-time haptics rendering. The proposed system can be easily employed in any application requiring a fast solution to a system of linear equations or simply involving a sparse matrix by vector multiplication.

1.3 Thesis Outline

The rest of this thesis is organized as follows. In Chapter 2, some of the existing techniques in the literature for object deformation modeling are reviewed and compared with each other. A brief survey of prior work on real-time implementation of FE-based deformation models is also provided at the end of this chapter. In Chapter 3, an overview of the FEM formulation of soft-object deformation is given. In this context, the partial differential equations for static equilibrium, geometric discretization, interpolation functions, elemental characteristic equations and dynamic analysis are discussed in details. Chapter 4 compares direct and iterative solvers as the two main approaches for solving the large and sparse linear system of equations produced by the FEM. A modified version of the iterative CG method for working with fixed-point computing units is proposed. The use of

pre-conditioners to improve the convergence of the CG method is also discussed. Chapter 5 provides details of hardware architecture design for implementing the fixed-point CG algorithm on FPGA. An overview of the problem, the proposed micro-architecture properties and also some of the challenges involved in attaining the design objectives are explained in this chapter. In Chapter 6, the performance of the proposed FPGA design is evaluated. The experimental setup and the results of hardware-in-the-loop haptic simulations using linear elastic FE models of deformation are also presented in this chapter. Finally, the thesis is concluded in Chapter 7 where some possible directions for future work are also suggested.

1.4 Related Publications

- R. Mafi, S. Sirouspour, B. Moody, B. Mahdavikhah, K. Elizeh, A. Kinsman, N. Nicolici, M. Fotoohi and D. Madill, "Hardware-based Parallel Computing for Real-time Haptic Rendering of Deformable Objects" submitted to *IROS 2008. IEEE/RSJ International Conference on Intelligent Robots and Systems, Niece, France*.

- Ramin Mafi, Shahin Sirouspour, Behzad Mahdavikhah, Brian Moody, Kaveh Elizeh, Adam Kinsman and Nicola Nicolici, "A Parallel Computing Platform for Real-time Haptic Interaction with Deformable Bodies" submitted to the *IEEE Transactions on Haptics*.

Chapter 2

Literature Review

This chapter provides a brief survey on the deformable body models in the literature. In real-time applications of deformable body modeling, there is a trade-off between the accuracy and complexity of the solution. These factors are compared with each other throughout this chapter. In our application, we have chosen FE-based modeling. The chapter is concluded by a review of prior work on real-time implementation of FE-based deformation models.

2.1 Deformable Body Modeling

Although deformable body modeling has only been an active area of research for about two decades [33], extensive research has already been conducted on the subject. Deformation modeling has has found an ever-increasing significance in different applications. Meier et al. [33] categorize these applications into three(3) groups:

- Pre-computed animations [34, 35]

- Image segmentation and registration (eg. 3D image reconstruction based on MRI or CT scans) [36,37]

- Haptics, bidirectional human-machine interfaces that allow users to interact with computer generated virtual worlds while receiving force-feedback based on a model [33,38]

In this chapter, some of the deformable body models used in the literature are briefly reviewed and a comparison among prose and cones of each method is provided. For a more comprehensive survey of the subject, the reader is addressed to [33,39,40]. Depending on whether the deformation models are physics-based/continuum mechanics-based or not, they can be generally classified in different categories. Throughout the rest of this chapter, first heuristic deformation models such as Spline and Chainmail modeling techniques are introduced. Mass-spring models as discrete physics-based methods and also finite element analysis of constitutive models based on continuum mechanics are briefly discussed subsequently.

Two main classes of deformable body models have have been most popular among others in the field of surgery/training simulation, namely mass-spring model (MSM) and finite element method (FEM). In the next chapter the FEM, which is employed in this thesis, will be discussed in more details.

Deformable Spline Model

Deformable Spline method (also known as *active contours*) is generally based on defining the deformation of smooth curves, surfaces or volumes as a function of

some certain control points. Figure 2.1 demonstrates an example of a Spline parametric surface. The curves can be adjusted by changing the location of the control points, or by adding or removing the control points or changing their weight [40]. Deformable Splines were the first models used in surgery simulation [41]. However due to the complexity of the computations, inaccuracies of non-physics-based modeling and simulation, and inconsistency for displaying the curves in modern graphic cards, this method is no longer employed [33].

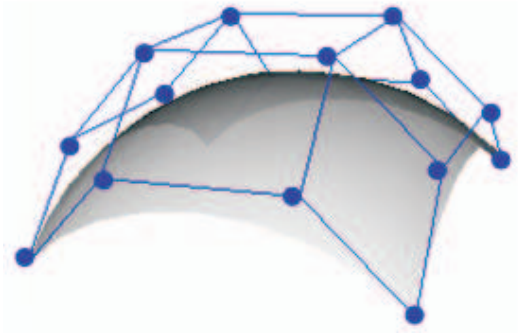


Figure 2.1: Spline parametric surface. *(Figure courtesy of Evgeny Demidov).*

Chainmail Model

Chainmail algorithm suggested by Gibson [42] is another heuristic method for deformable body modeling. In this method, each element as shown in Fig. 2.1 is interconnected to its neighbors as links of a chain, having a certain level of freedom. As the first step of deformation modeling, large enough displacement of an element results in a position change of its neighbor elements, and their neighbors and so on. In the second step, the raw displaced positions of the elements are adjusted through minimizing an energy function defined in the system. There is no clear mechanism for determining the reaction forces in this method. In one

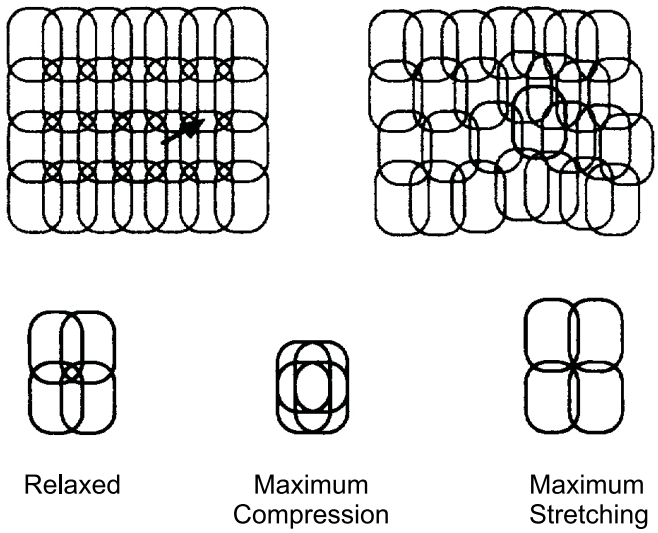


Figure 2.2: 2D Chainmail configuration and its corresponding deformation when one element is displaced. *(Figure courtesy of Sarah F.F. Gibson [42]).*

approach, the force is assumed to be proportional to the displacement and is calculated correspondingly. However this assumption is not true for inhomogeneous materials.

Despite the fact that this method offers a relatively simple approach for modeling complex mechanical behaviors, determining the parameters of the model for achieving realistic results is not an easy task. Additionally, simultaneous multiple contact points in this model result in an extensive computational cost for determining the propagation of the imposed displacements on the elements [33]. These disadvantages have hampered widespread use of the Chainmail model.

2.1.1 Mass-Spring Model

Due to a relative simplicity in implementation and real-time behavior, MSM has been one of the common methods in the literature for a variety of problems such

as soft tissue modeling, cloth simulation and facial animation [43–46]. Another advantage of this method is its ability to allow topological changes in the object without significant computational burden, making it suitable for operations such as cutting [47].

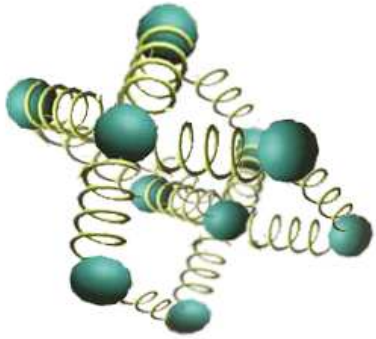


Figure 2.3: Mass-Spring system

A mass-spring system, as shown in Fig. 2.1.1, is consisted of a set of mass points, also referred to as *nodes*, that are connected by springs in the model mesh. For simplicity, damper elements similar to mass are defined locally for each node. Using the Newton's second law, the equation of motion for a mass particle i in the lattice structure is derived as:

$$m_i\ddot{\mathbf{x}}_i = -c_i\dot{\mathbf{x}}_i + \sum_j \mathbf{g}_{ij} + \mathbf{f}_i \tag{2.1}$$

where $\mathbf{x}_i \in \Re^3$ is the position and m_i is the mass of node i. In the right hand side of (2.1), c_i represents the damping of the node and $\sum_j \mathbf{g}_{ij}$ is the sum of elastic forces coming from the neighbor nodes connected to node i and $\mathbf{f}_i$ is the total external forces at that point.

Assembling Eq. (2.1) for all the n nodes of the MSM mesh and representing them in the matrix form, the following equation is derived:

$$\mathbf{M}\ddot{\mathbf{x}} + \mathbf{C}\dot{\mathbf{x}} + \mathbf{K}\mathbf{x} = \mathbf{f} \tag{2.2}$$

where $\mathbf{M}$ and $\mathbf{C} \in \Re^{3n}$ are diagonal matrices for mass and damping; $\mathbf{K} \in \Re^{3n}$ is a sparse stiffness matrix. The system of equations in (2.2) can be broken into two first order equations and be integrated over time to find the displacement vector $x(t)$.

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \ddot{\mathbf{x}} \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ -\mathbf{M}^{-1}\mathbf{K} & -\mathbf{M}^{-1}\mathbf{C} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \dot{\mathbf{x}} \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \mathbf{f} \end{bmatrix} \tag{2.3}$$

The MSM has some main drawbacks prohibiting its use in applications requiring accurate simulation and realistic results. The MSM not only dicretizes the geometry of the model, but also it discretizes the continuous equations of the motion, resulting in less accurate results compared with that achievable with the FEM. The method also lacks a clear mechanism for selecting the model parameters, i.e. mass, spring and damper values to produce a physically correct deformation response. It has been shown that homogenous linear properties of a material can not be achieved through assigning the same spring stiffness to its mass-spring model [48]. Therefore a method for determining MSM parameters is needed. There has been several attempts to develop methods for estimating the model parameters in the MSM. These methods can be generally categorized into two groups. In the first approach, it is attempted to obtain the MSM parameters through the known physical characteristics of the material. Methods in this class are rather general and may

need ad hoc modifications for improved results. The second group includes the approaches based on optimization algorithms that minimize the error between the model response and some reference data. Methods in this group are more application specific.

Following the first category, Van Gelder proposed an approach for defining the stiffness of the spring based on the sum of the area/volume of the triangles/tetrahedra containing that edge in the triangular/tetrahedral mesh [48]. As some examples from the second category, Bianchi et al. proposed a method based on genetic algorithms identifying topology of the mesh in addition to MSM parameters, using FE-based model as the training reference [49]. In contrast, Zerbato et al. suggested a solution for a user predefined mesh without modifying it, calibrating the MSM model by a genetic algorithm [47].

Another deficiency of MSM is its lack of potential for accurate simulation of torques applied to the model making it inappropriate for applications with multiple contact points. Due to the nature of the defined springs, it is not easy to model torsion in the mesh. In addition, large mass-spring-damper meshes can impede rapid global propagation of deformations resulting in a localized deformation [33].

2.1.2 Finite Element Model

The more accurate behavior of FE-based models compared to MSM makes them more suitable for medical applications requiring high-fidelity response. Contrary to mass-spring models that define the discrete equations of motion for each node of the model mesh (i.e. Eq. (2.1)), there are methods that consider the object as a continuum and define a distributed energy within the body. In such approaches,

the FEM is usually used to partition the object into small elementary shapes such as tetrahedrons and then derive the equations of motion for each element as a function of the mesh nodes resulting in a discretization of the model in the spatial domain [19, 20]. Meshless techniques including meshless FE methods have also been proposed for modeling of object deformation and cutting [50–52].

Static linear elastic FE analysis requires the solution to a large but sparse linear system of equations in the form of $\mathbf{KU} = \mathbf{f}$, where $\mathbf{U}$ is a vector of node deformation and $\mathbf{f}$ is a vector of external forces applied to the nodes. Similarly, an incremental form of such equations may be derived for some nonlinear deformation models in which $\mathbf{K}$ and $\mathbf{f}$ can be functions of the actual deformation [21]. In dynamic analysis, inertial body and velocity dependent damping forces are taken into account. In this case spatial discretization is followed by a temporal discretization using explicit or implicit integration techniques [22]. Explicit integration routines are easier to implement but can suffer from poor numerical stability. Depending on the size and characteristics of the elements, very small integration time steps may be required to maintain stability. Implicit methods, on the other hand, exhibit robust numerical behavior independent of the time step used [22]. Similar to the static case, a linear system of equations must be solved at each time step. FEM formulation will be discussed in more details in Chapter 3.

2.2 Real-time Simulation of FE-based Models

FE-based real-time simulation of soft-object deformation has been hindered by the large amount of computations that must be completed within each time step, e.g. $1-10$ msec if haptic feedback is needed. Off-line pre-computation of $\mathbf{K^{-1}}$ has been

suggested in [25,53]. The result is calculated through a dense matrix by force vector multiplication in each time step. However, it should be noted that in most haptic simulations it is the displacement not the force at the contact node that is known. A change of role between these force and position variables would result in a new $\mathbf{K}$ matrix that is dependent on the contact node hence rendering pre-computation of $\mathbf{K}^{-1}$ impractical. Additionally, the precision of this method suffers from numerical errors. A similar approach relies on the superposition principle in linear elastic models to compute the deformation response based on pre-computed responses to deformations at all possible interaction nodes. Aside from their massive memory requirement, pre-computation methods are unable to model nonlinear behavior or accommodate for changes in the FE mesh, e.g. due to cutting.

Bro-Nielsen and Cotin [25] used a matrix condensation approach in their experiments. In this method, the equations for a volumetric model are rearranged in a way that only the deformation of surface nodes will be calculated. This approach, to some extent, would reduce the size of the problem, but at the expense of producing a stiffness matrix that is no longer sparse. In most cases, the increase in the density of the matrix would nullify any gain in size reduction for the matrix by vector multiplications in iterative methods such as the CG. Moreover, there are many applications in which the motion of internal nodes is of significance and must be included in the simulation.

Zhuang and Canny [26] worked on geometric nonlinear FE models. They use graded-mesh for increasing the computation speed at the expense of reducing the accuracy of simulation. In their approach, mass lumping approximation is also a

key assumption for reducing the computation load. They solve the problem using explicit integration scheme with a frequency of about 20 Hz for 1331 elements on a 400MHz Pentium II PC. To obtain the real-time performance for haptics, the simulation is delayed for one cycle, and using interpolation the force is computed at the desired rate. Xunlei Wu et al. [54] employed both material and geometric nonlinearities in FEM. Similar to Zhuang and Canny they rely on mass-lumping assumption and use explicit integration method to solve the dynamics of the problem. They reported a similar frequency of 20 Hz for 2,200 nodes using 800MHz Pentium III PC.

Taylor et al. [28] have implemented a non-linear FE model on a general-purpose graphics processing unit (GPU). By parallel utilization of the hardware resources available on GPU, they have gained solution speeds of up to $\times 16.4$ faster compared with equivalent CPU implementations. They use nonlinear total Lagrangian explicit FE formulation. Again, the use of an explicit integration scheme significantly reduces the complexity of the problem in their case. Some other attempts at real-time FE simulations have been reported in [24, 27, 55].

Chapter 3

Finite Element Formulation

In this chapter, we will discuss how to model linear elastic deformations through finite element formulation. Considering our application, FEM is reviewed and discussed from an engineering point of view rather than its abstract mathematical formulation. For more information about FEM, the reader is addressed to [20, 22, 23].

Compared to other techniques such as MSM, FEM results in a more accurate physical modeling in the sense that it is formulated based on the original continuum mechanics differential equations. For this reason, it is often easier to calibrate FE-based models to obtain the desired deformation response. In FEM, the idea is to extract the partial differential equilibrium equations for a general problem. Next the geometry of the problem (domain of the differential equations) is discretized into smaller elements with known geometrical shapes. Then using interpolation functions, the continuous equations over each element is defined as a function of its nodal displacements. Finally the continuous equilibrium equations are rewritten based on the nodal displacements per element and are assembled

into one global matrix of equations. In the next chapter, we will discuss different approaches for solving this system of equations. Steps involved in FEM are further elaborated in the following sections.

3.1 Problem Statement

As the first step of FEM formulation, the problem has to be stated more precisely. Consider a general example of 3-D body is in equilibrium conditions in Fig. 3.1.

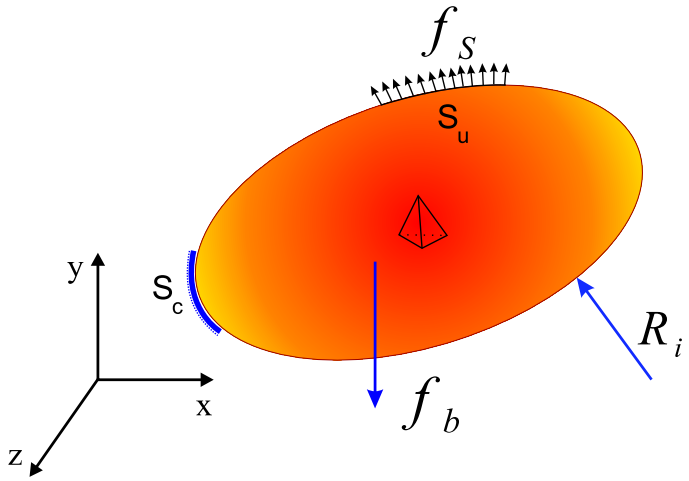


Figure 3.1: General 3D body for structural analysis

The body is constrained at surface S_c with prescribed displacement of $\mathbf{u}_S$ and it is exposed to body forces distributed over volume denoted by $\mathbf{f}_b$ (e.g. gravity force), surface forces $\mathbf{f}_S$ distributed on S_u (eg. pressure) and concentrated loads $\mathbf{R_i}$.

The relationship between strain vector $\epsilon^T = [\epsilon_{xx}\ \epsilon_{yy}\ \epsilon_{zz}\ \epsilon_{xy}\ \epsilon_{xz}\ \epsilon_{yz}]$ and the displacement vector $\mathbf{u}^T = [\ u\ v\ w]$ can be expressed as [56]

$$\begin{aligned}\epsilon_{xx} &= \frac{\partial u}{\partial x} + \frac{1}{2}\left[(\frac{\partial u}{\partial x})^2 + (\frac{\partial v}{\partial x})^2 + (\frac{\partial w}{\partial x})^2\right] \\ \epsilon_{xy} &= \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} + \left[\frac{\partial u}{\partial x}\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\frac{\partial v}{\partial y} + \frac{\partial w}{\partial x}\frac{\partial w}{\partial y}\right]\end{aligned} \tag{3.1}$$

The remaining terms of ϵ are defined similarly. For small displacements higher order derivatives in Eq. (3.1) can be neglected. This approximation leads to linear strain-displacement relation. Considering the higher order derivatives in (3.1) results in geometrical nonlinearity [21]. Equations in (3.1) can be expressed in the matrix form

$$\boxed{\epsilon = \mathbf{L}\mathbf{u}} \tag{3.2}$$

where the strain-displacement matrix $\mathbf{L}$ is a linear differential operator matrix

$$\mathbf{L} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} \\ 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \end{bmatrix}$$

The relationship between stress $\sigma^T = [\sigma_{xx}\ \sigma_{yy}\ \sigma_{zz}\ \sigma_{xy}\ \sigma_{xz}\ \sigma_{yz}]$ and strain ϵ using the generalized Hooke's law can be represented as [56]

$$\boxed{\sigma = \mathbf{D}\epsilon} \tag{3.3}$$

For linear homogenous and isotropic materials the stress-strain matrix $\mathbf{D}$ is related to Poisson's ratio ν and Young's elasticity module E in the form of

$$\mathbf{D} = \begin{bmatrix} \lambda + 2\mu & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda + 2\mu & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & \lambda + 2\mu & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu \end{bmatrix} \tag{3.4}$$

where λ and μ are called $lamé$'s constants and are defined as

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \qquad\qquad \mu = \frac{E}{2(1+\nu)}$$

Based on how the stress-strain relationship is established, material nonlinearity can be taken into account in more complex forms [22].

Now the problem can be stated as the calculation of the displacements $\mathbf{u}$ and the corresponding stress σ and strain ϵ, given the external forces (body, surface and concentrated loads), boundary conditions on S_c as well as the strain-displacement and stress-strain laws for the deformable body.

3.2 Differential Equations Describing Static Equilibrium

At this state, the objective is to express the equilibrium equations in terms of the unknown object displacement vector $\mathbf{u}$ that must be calculated. After determining $\mathbf{u}$ based on these equations, it is straight forward to compute stress σ and strain ϵ according to Eqs.(3.2) and (3.3).

The development of differential equations of equilibrium for FEM in structural analysis can be based on either the *virtual work principle* or the *minimum total potential energy principle*. The virtual work principle, also known as virtual displacement principle, is more general as it is applicable to both linear and non-linear material behaviors. This principle states the deformable body in Fig. 3.1 is in equilibrium if and only if for any small virtual displacement [1] imposed on the body, the whole external virtual work done on the body equals to the total internal virtual work, i.e.

$$\int_V \bar{\epsilon}^T \sigma dV = \int_V \bar{\mathbf{u}}^T \mathbf{f}_b \, dV + \int_S \bar{\mathbf{u}}_S^T \mathbf{f}_S \, dS + \sum\nolimits_i \bar{\mathbf{u}}_i^T \mathbf{R}_i \tag{3.5}$$

where $\bar{\epsilon}$ is the virtual strain corresponding to the virtual displacement $\bar{\mathbf{u}}$. The stress σ corresponds to the external loads of the right hand side at the equilibrium state. The left hand side of Eq. (3.5) is the internal virtual work, whereas the right hand side of this equation is the virtual work by external forces. The overbar notation is used to emphasize that the virtual displacements and the corresponding strains are unrelated to the real deformation and strain that the body undergoes given the

[1] These virtual displacements are zero at the prescribed displacements on S_c

external loads and restrains of the problem stated in the previous section.

In the following sections the discretization of the geometry will be discussed. Then it is followed by expressing Eq. (3.5) in terms of the nodal displacements.

3.3 Geometric Discretization

As stated earlier, generally there is no direct analytical solution to differential equations obtained in 3.5 for a given object geometry. For this reason the body in Fig. 3.1 is approximated as an assemblage of a set of smaller elements with known regular geometries. These elements are disjointed and interconnected to each other only through nodes on the element boundaries. Fig 3.2 shows some typical 2D and 3D elements used in FEM. It also gives an example of the approximation of a larger geometry using tetrahedral elements.

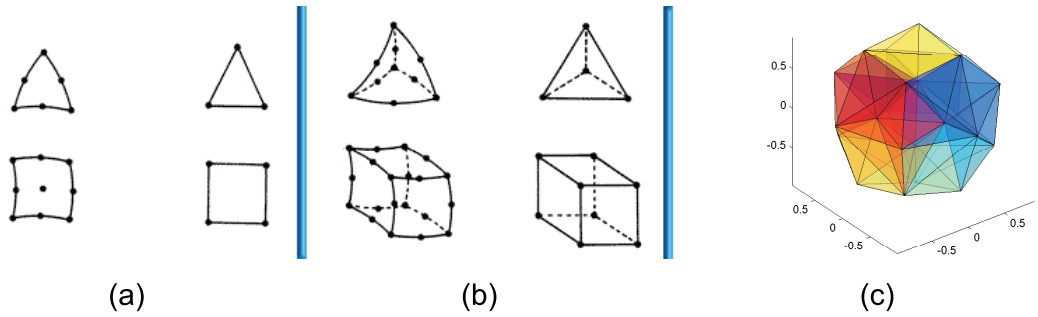


Figure 3.2: **(a)** Typical 2D elements including: linear triangular and rectangular elements, quadratic triangular element with 6 nodes and Lagrangian with 9 nodes **(b)** Typical 3D elements including: linear tetrahedral and hexahedral elements, 10 node tetrahedral and 20 node brick element **(c)** A more complex geometry assembled with tetrahedral elements

In determining the number of the nodes, there is a trade off between the simulation accuracy and the complexity of the calculations. Since the accuracy is dependent on the number of nodes, meshes using elements with larger number of nodes generally require fewer number of elements for the same degree of accuracy. As a rule of thumb for a proper mesh in FEM, elements with angles close to 0 or 180 degrees should be avoided [57, 58]. Additionally, the aspect ratio in each element defined as the ratio of the longest edge of the element to the shortest one should be a small number, typically less than or equal to $2 \sim 4$). Following these tips leads to a better accuracy in FEM results.

Using mesh refinement techniques, it is possible to gain more accurate results with the same number of the nodes. To this end, a finer mesh is employed in the regions exposed to larger stress and deformation whereas a coarser mesh is used in other areas [57].

3.4 Interpolation Functions

In Sec. 3.2, the equilibrium equation was derived as a function of the continuous displacement of the body. This section discusses how FEM approximates this continuous deformation based on the displacement of mesh nodes in the discretized geometry of the body.

Focusing on elements in the mesh, an *interpolation function* (also known as *shape function*) can be defined for each element to give an estimation of the varying internal quantities as a function of its nodal displacements. To derive the shape function (denoted as $\mathbf{H}$), in the first step the interpolation equation should be defined per

element. Interpolation equations (denoted as $\mathbf{\Phi}$) are usually in the form of polynomials. Depending on the dimension of the space, $\mathbf{\Phi}$ can be a function of x (1D), x and y (2D) or x, y and z (3D). The number of coefficients in the polynomial should be equal to the number of elemental nodes to provide enough degrees of freedom. Some examples of polynomial $\mathbf{\Phi}$ are given below

Element	*Polynomial Interpolation Equation*
3 node triangle (2D)	$\mathbf{\Phi} = \mathbf{a_0} + \mathbf{a_1}x + \mathbf{a_2}y$
4 node rectangle (2D)	$\mathbf{\Phi} = \mathbf{a_0} + \mathbf{a_1}x + \mathbf{a_2}y + \mathbf{a_3}xy$
6 node triangle (2D)	$\mathbf{\Phi} = \mathbf{a_0} + \mathbf{a_1}x + \mathbf{a_2}y + \mathbf{a_3}xy + \mathbf{a_4}x^2 + \mathbf{a_5}y^2$
4 node tetrahedron (3D)	$\mathbf{\Phi} = \mathbf{a_0} + \mathbf{a_1}x + \mathbf{a_2}y + \mathbf{a_3}z$
8 node hexahedron (3D)	$\mathbf{\Phi} = \mathbf{a_0} + \mathbf{a_1}x + \mathbf{a_2}y + \mathbf{a_3}z + \mathbf{a_4}xy + \mathbf{a_5}yz + \mathbf{a_6}xz + \mathbf{a_7}xyz$

Where $\mathbf{\Phi}$ and $\mathbf{a}_i$ coefficients are 2D or 3D vectors. In structural FE-based analysis $\mathbf{\Phi}$ represents the interpolated internal displacement vector $(\ u\ v\ w)$ while $(\ x\ y\ z)$ indicates the position in coordination system. Interpolation equations must be invariant under linear coordination transformations. Therefore in deriving interpolation equations, $\mathbf{\Phi}$ should be a symmetric polynomial, eg. it can not have the x^2 and y^2 terms while missing z^2.

Having more nodes in an element translates into higher degree polynomials for interpolation equations resulting in smoother simulations. However this will increase the complexity of the equations per element. In the following through an example, it is shown how interpolation functions can be derived based on $\mathbf{\Phi}$. For

the simplest two dimensional element, a three-node triangle, the linear interpolation equation as is given by

$$\mathbf{\Phi}^T = [1 \quad x \quad y] \begin{bmatrix} \mathbf{a_0}^T \\ \mathbf{a_1}^T \\ \mathbf{a_2}^T \end{bmatrix} \tag{3.6}$$

It should be noted that $\mathbf{\Phi}$ as well as $\mathbf{a_0}$, $\mathbf{a_1}$ and $\mathbf{a_2}$ are two dimensional vectors in this case. To obtain the interpolation function, Eq. (3.6) is constrained at node positions (x_i, y_i), to nodal displacement values $\mathbf{\Phi_i}$, $i = 0, 1, 2$.

$$\begin{bmatrix} \mathbf{\Phi_0}^T \\ \mathbf{\Phi_1}^T \\ \mathbf{\Phi_2}^T \end{bmatrix} = \begin{bmatrix} 1 & x_0 & y_0 \\ 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \end{bmatrix} \begin{bmatrix} \mathbf{a_0}^T \\ \mathbf{a_1}^T \\ \mathbf{a_2}^T \end{bmatrix} \tag{3.7}$$

Solving Eq. (3.7) for $\mathbf{a_i}$ and substituting in Eq. (3.6) yield

$$\begin{bmatrix} \mathbf{a_0}^T \\ \mathbf{a_1}^T \\ \mathbf{a_2}^T \end{bmatrix} = \begin{bmatrix} 1 & x_0 & y_0 \\ 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{\Phi_0}^T \\ \mathbf{\Phi_1}^T \\ \mathbf{\Phi_2}^T \end{bmatrix} \tag{3.8}$$

$$\mathbf{\Phi}^T = [1 \quad x \quad y] \begin{bmatrix} 1 & x_0 & y_0 \\ 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{\Phi_0}^T \\ \mathbf{\Phi_1}^T \\ \mathbf{\Phi_2}^T \end{bmatrix} \tag{3.9}$$

Eq. (3.9) expresses the continuous internal displacement vector $\mathbf{\Phi}$ in terms of the nodal displacements $\mathbf{\Phi_0}$, $\mathbf{\Phi_1}$, and $\mathbf{\Phi_2}$. Writing the equations for column vectors we have the interpolation function $\mathbf{H}$ as

$$\mathbf{\Phi}(x,y) = \mathbf{H}(x,y) \begin{bmatrix} \mathbf{\Phi_0} \\ \mathbf{\Phi_1} \\ \mathbf{\Phi_2} \end{bmatrix} \tag{3.10}$$

$$\mathbf{\Lambda}(x,y) = [1 \quad x \quad y] \begin{bmatrix} 1 & x_0 & y_0 \\ 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \end{bmatrix}^{-1} = [\Lambda_0 \;\; \Lambda_1 \;\; \Lambda_2]$$

$$\mathbf{H}(x,y) = \begin{bmatrix} \Lambda_0 & 0 & \vdots & \Lambda_1 & 0 & \vdots & \Lambda_2 & 0 \\ 0 & \Lambda_0 & \vdots & 0 & \Lambda_1 & \vdots & 0 & \Lambda_2 \end{bmatrix}$$

With a change of notation, Eq. (3.10) can be written as

$$\boxed{\mathbf{u} = \mathbf{H}\hat{\mathbf{u}}} \tag{3.11}$$

where $\mathbf{u} = \mathbf{\Phi}$ and $\hat{\mathbf{u}}$ represents the nodal displacement vector of the element. Generally in this context, variables with '$\hat{}$' denote discrete nodal parameters. The procedure for deriving interpolation function based on interpolation equation for other elements with different number of nodes and dimensions is similar to that of this example and will not be repeated here.

3.5 Elemental Stiffness Matrix and Assemblage

By rewriting Eq. (3.11) for each element in the mesh based on the displacement vector of all the nodes in the general three-dimensional case, we have

$$\mathbf{u}^e(x, y, z) = \mathbf{H}^e(x, y, z)\hat{\mathbf{U}} \tag{3.12}$$

where $\hat{\mathbf{U}}$ is the *general nodal displacement vector* of the mesh and the superscript e denotes element e. For d-dimensional mesh with n nodes the length of this vector is $d \times n$. Similarly we can express the elemental strain parameter based on the general nodal displacement vector $\hat{\mathbf{U}}$.

$$\epsilon^e(x, y, z) = \mathbf{L}\mathbf{u}^e(x, y, z) = \mathbf{L}\mathbf{H}^e(x, y, z)\hat{\mathbf{U}} = \mathbf{B}^e(x, y, z)\hat{\mathbf{U}} \tag{3.13}$$

matrix $\mathbf{L}\mathbf{H}^e(x, y, z)$ is denoted as $\mathbf{B}^e(x, y, z)$. Defining the element displacements and strains in terms of the general displacement vector $\hat{\mathbf{U}}$ makes the assemblage process straight forward.

At this state, the equations of equilibrium derived in Sec. 3.2 based on virtual displacement can be written as a sum of integration over the volume of all the elements.

$$\sum_e \int_{V^e} \bar{\epsilon}^{eT} \sigma^e dV^e = \sum_e \int_{V^e} \bar{\mathbf{U}}^{eT} \mathbf{f}_b^e \, dV^e + \sum_e \int_{S^e} \bar{\mathbf{U}}_S^{e\,T} \mathbf{f}_S^e \, dS^e + \sum_i \bar{\mathbf{U}}_i{}^T \mathbf{R}_i \tag{3.14}$$

where $\bar{\mathbf{U}}^e$ is the virtual displacement within each element and $\bar{\epsilon}^e$ is the corresponding strain. Substituting Eqs. (3.12) and (3.13) and (3.3) in Eq. (3.14), we obtain

$$\bar{\hat{\mathbf{U}}}^T \left(\sum_e \int_{V^e} \mathbf{B}^{eT} \mathbf{L}^e \mathbf{B}^e \, dV^e \right) \hat{\mathbf{U}} = \bar{\hat{\mathbf{U}}}^T \Bigg[\left(\sum_e \int_{V^e} \mathbf{H}^{eT} \mathbf{f}_b^e \, dV^e \right) + \left(\sum_e \int_{S^e} \mathbf{H}_S^{eT} \mathbf{f}_S^e \, dS^e \right) + \sum_i \mathbf{R}_i \Bigg] \tag{3.15}$$

Given that the virtual quantities are arbitrary and independent imaginary variables, virtual displacement $\bar{\hat{U}}$ can be removed from both sides of the above equation. Denoting the global nodal displacement by $\mathbf{U}$ from now on (i.e. $\mathbf{U} \equiv \hat{\mathbf{U}}$), the remaining terms can be expressed as

$$\boxed{\mathbf{KU} = \mathbf{f}} \tag{3.16}$$

where $\mathbf{K}$ is the global stiffness matrix of the mesh given by

$$\mathbf{K} = \sum_e \mathbf{K}^e = \sum_e \int_{V^e} \mathbf{B}^{eT} \mathbf{L}^e \mathbf{B}^e \, dV^e \tag{3.17}$$

The right hand side of Eq. (3.16) is the external force vector on exerted the nodes.

Now the assemblage can be performed by simply adding the elemental stiffness matrices $\mathbf{K}^e$. To make this step more clear an example for a tetrahedral mesh is provided below. Consider the tetrahedron in Fig. 3.3 with node numbers m, i, j and k. In a mesh with n nodes, m, i, j and k can be any unique number from 1 to n.

Since there are four nodes in the tetrahedron, the elemental stiffness matrix will

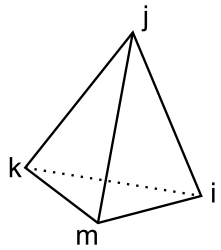


Figure 3.3: A four-node tetrahedron

be 12×12, with the following structure

$$\mathbf{K}^e = \begin{pmatrix} \mathbf{K}_{i,i} & \mathbf{K}_{i,j} & \mathbf{K}_{i,k} & \mathbf{K}_{i,m} \\ \mathbf{K}_{j,i} & \mathbf{K}_{j,j} & \mathbf{K}_{j,k} & \mathbf{K}_{j,m} \\ \mathbf{K}_{k,i} & \mathbf{K}_{k,j} & \mathbf{K}_{k,k} & \mathbf{K}_{k,m} \\ \mathbf{K}_{m,i} & \mathbf{K}_{m,j} & \mathbf{K}_{m,k} & \mathbf{K}_{m,m} \end{pmatrix} \tag{3.18}$$

It should be noted that $\mathbf{K}_{i,i}$ and the other similar terms are 3×3 matrices. In the assemblage process, $\mathbf{K}_{i,i}$ from location (1,1) [2] in $\mathbf{K}^e$ gets accumulated in location (i, i) in the global stiffness matrix, i.e.

$$\boxed{\mathbf{K}_{p,q_{(3\times3)}} = \sum_e \mathbf{K}^e_{p,q_{(3\times3)}}} \qquad \begin{array}{l} p, q = 1, \cdots node - number \\ \\ e = 1, \cdots element - number \end{array} \tag{3.19}$$

Fig. 3.4 on Page 33 summarizes the steps involved in the FE procedure. In Step 1, the structure being analyzed is discretized into a sufficient number of basic elements. In Step 2, expressions for local element characteristics, i.e. the elemental stiffness matrices and nodal forces are derived. These expressions can be obtained

[2] the term 'location' is refered to the location of each 3×3 block matrix, and not a single component of the matrix

according to the virtual displacement principle. In Step 3, the elemental stiffness matrices and nodal forces are assembled into one general matrix equation. Finally in Step 4, the boundary conditions of the problem are applied and the resulting system of equations is solved. The next chapter will focus on algorithms for solving the system of equations in the fourth step.

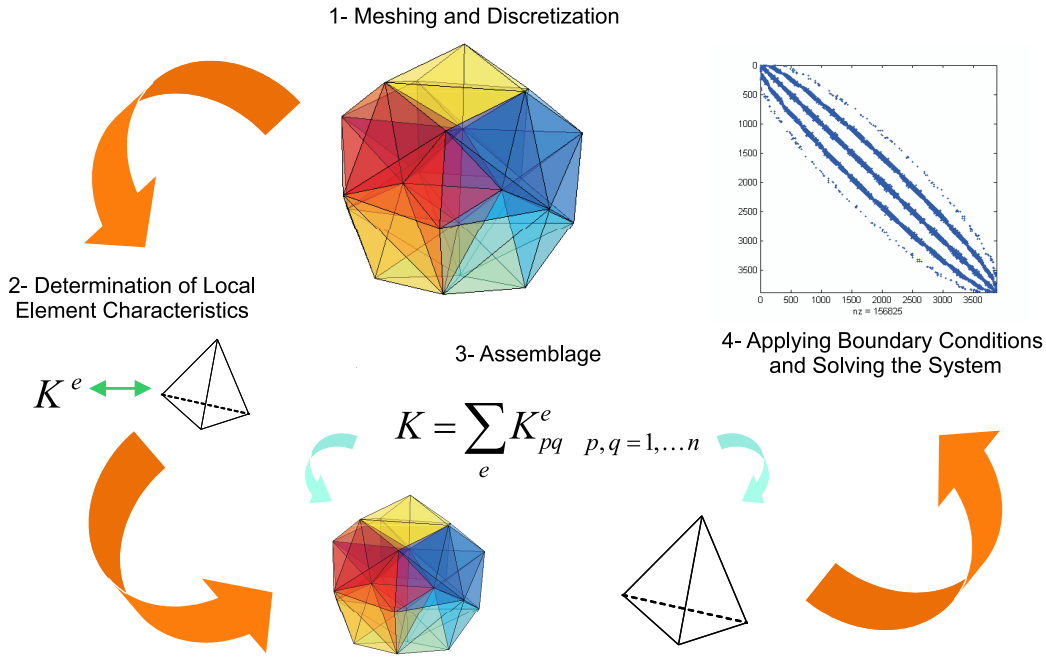


Figure 3.4: FE procedure

3.6 Dynamic FE Analysis

In dynamic analysis, inertial body and velocity dependent damping forces are taken into account. The governing equations for dynamic analysis in the finite

element system are derived as:

$$\mathbf{M}\ddot{\mathbf{U}} + \mathbf{C}\dot{\mathbf{U}} + \mathbf{K}\mathbf{U} = \mathbf{f} \tag{3.20}$$

where $\dot{\mathbf{U}}$ and $\ddot{\mathbf{U}}$ are the first and second time derivatives of the global nodal displacement vector $\mathbf{U}$, representing velocity and acceleration vectors respectively. $\mathbf{M}$ and $\mathbf{C}$ are the mass and damping matrices. Similar to assemblage of stiffness matrix $\mathbf{K}$, mass matrix $\mathbf{M}$ can be assembled by adding up the mass matrices per element. Using the interpolation function $\mathbf{H}$ the mass matrix $\mathbf{M}$ can be defined as [22]

$$\mathbf{M} = \sum_e \mathbf{M}^e = \sum_e \int_{V^e} \rho^e \mathbf{H}^{eT} \mathbf{H}^e dV^e \tag{3.21}$$

In many cases for simplification, mass of the body is approximated to be concentrated at the mesh nodes resulting in a diagonal mass matrix. This technique is known as *mass lumping*.

The damping matrix $\mathbf{C}$ can be defined similar to $\mathbf{M}$ in (3.21), however in practice it is difficult to determine the actual damping parameters. Therefore $\mathbf{C}$ matrix is often constructed as a linear combination of mass and stiffness matrices known as Rayleigh damping

$$\mathbf{C} = \alpha\mathbf{M} + \beta\mathbf{K} \tag{3.22}$$

There are different numerical techniques for solving the differential equations in (3.20) in the discrete-time domain depending on how position, velocity and acceleration terms are updated at each time step. While explicit approaches such as the central difference method are, in general, easier to implement, they are only *conditionally* stable. To maintain stability, the simulation time step Δt must

be smaller than a critical value Δt_{cr} which would depend on the characteristic of the worst element in the mesh. In contrast, implicit methods such as Houbolt, Wilson and Newmark [22] guarantee numerical stability independent of the time step used. They do, however, require solving a linear system of equations similar to that in (3.16) at each time step. In the following, Newmark integration scheme, as a common method used in dynamic analysis is briefly reviewed.

3.6.1 Newmark Integration

Newmark integration scheme [22, 23] is a common method for implicit time discretization of dynamics equations arising from FE models. A brief overview of this algorithm is presented below. Consider the standard linear second-order dynamics of object deformation given by

$$\mathbf{M}\ddot{\mathbf{U}}_{\mathbf{n+1}} + \mathbf{C}\dot{\mathbf{U}}_{\mathbf{n+1}} + \mathbf{K}\mathbf{U}_{\mathbf{n+1}} + \mathbf{f}_{\mathbf{n+1}} = \mathbf{0} \tag{3.23}$$

The position and velocity vector at time step $n+1$ can be approximated as follows

$$\begin{aligned} \mathbf{U}_{n+1} &= \mathbf{U}_n + \Delta t\dot{\mathbf{U}}_n + \frac{1}{2}\Delta t^2\Big((1-\beta)\ddot{\mathbf{U}}_n + \beta\ddot{\mathbf{U}}_{n+1}\Big) \\ \dot{\mathbf{U}}_{n+1} &= \dot{\mathbf{U}}_n + \Delta t\Big((1-\gamma)\ddot{\mathbf{U}}_n + \gamma\ddot{\mathbf{U}}_{n+1}\Big) \end{aligned} \tag{3.24}$$

where γ and β are two constants yet to be chosen. The above equations can be solved for $\ddot{\mathbf{U}}_{\mathbf{n+1}}$ and $\dot{\mathbf{U}}_{\mathbf{n+1}}$ resulting in

$$\begin{aligned} \ddot{\mathbf{U}}_{n+1} &= \ddot{\hat{\mathbf{U}}}_{n+1} + \frac{2}{\beta\Delta t^2}\mathbf{U}_{n+1} \\ \dot{\mathbf{U}}_{n+1} &= \dot{\hat{\mathbf{U}}}_{n+1} + \frac{2\gamma}{\beta\Delta t}\mathbf{U}_{n+1} \end{aligned} \tag{3.25}$$

where

$$\begin{aligned}\ddot{\hat{\mathbf{U}}}_{n+1} &= -\frac{2}{\beta\Delta t^2}\mathbf{U}_n - \frac{2}{\beta\Delta t}\dot{\mathbf{U}}_n - \frac{1-\beta}{\beta}\ddot{\mathbf{U}}_n \\ \dot{\hat{\mathbf{U}}}_{n+1} &= -\frac{2\gamma}{\beta\Delta t}\mathbf{U}_n + (1-\frac{2\gamma}{\beta})\dot{\mathbf{U}}_n + (1-\frac{\gamma}{\beta})\Delta t\ddot{\mathbf{U}}_n\end{aligned} \tag{3.26}$$

Substituting (3.25) and (3.26) into the dynamics equation of (3.23) yields

$$\hat{\mathbf{A}}\mathbf{U}_{n+1} = \hat{\mathbf{f}} \tag{3.27}$$

where

$$\begin{aligned}\hat{\mathbf{A}} &= \frac{2}{\beta\Delta t^2}\mathbf{M} + \frac{2\gamma}{\beta\Delta t}\mathbf{C} + \mathbf{K} \\ \hat{\mathbf{f}} &= -(\mathbf{f}_{n+1} + \mathbf{C}\dot{\hat{\mathbf{U}}}_{n+1} + \mathbf{M}\ddot{\hat{\mathbf{U}}}_{n+1})\end{aligned} \tag{3.28}$$

Note that to obtain the displacement vector $\mathbf{U}_{n+1}$ one needs to solve the system of linear equations in (3.27) similar to that of the static case in 3.16. The following choice of the parameters β and γ guarantees the stability of the discretized system [22, 23]

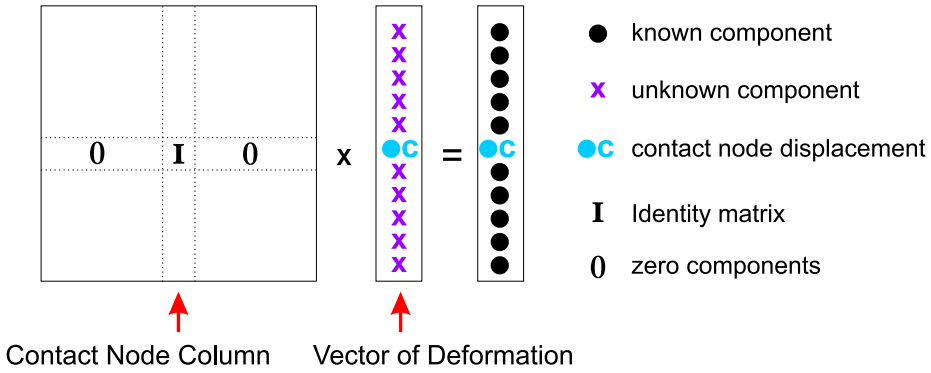
$$\gamma \geq 0.5 \qquad \beta \geq 0.5(0.5+\gamma)^2 \tag{3.29}$$

3.7 Modified Force Vector

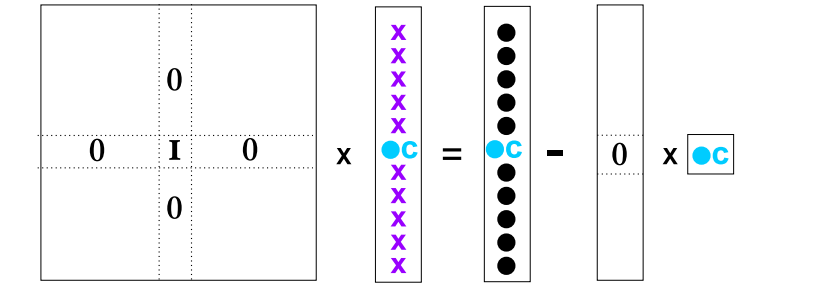
In this work modeling of single-point interaction with deformable objects using an impedance-type haptic device is considered. In such case, the external force at the contact node is unknown whereas the displacement of the node is constrained

to the device measured position. Obviously this type of boundary condition is inconsistent with the static or dynamic equations in (3.16) and (3.20) where $\mathbf{f}$ represents the vector of external forces acting on the nodes. To address this problem, as shown in Fig. 3.5.a, all but the diagonal elements of the rows and columns corresponding to the contact node are set to zero. The diagonal entries are set to one [3]. The force vector $\mathbf{f}$ is also modified to account for the multiplication of the omitted columns by the known displacement of the contact node, i.e. see Fig. 3.5.b. After solving the resulting equations for $\mathbf{U}$, the unknown force at the contact point can be calculated as shown in Fig. 3.5.c. It should be noted that throughout the process of modification, matrix $\mathbf{K}$ preserves its symmetry and positive-definiteness.

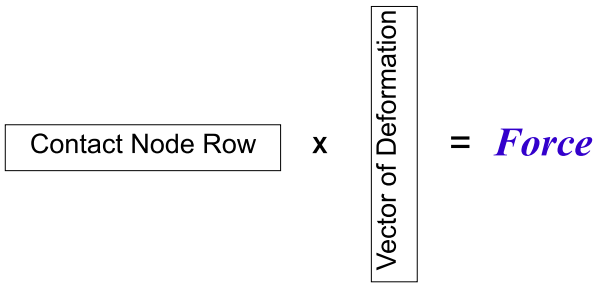
[3] To improve the condition number of the matrix, as a heuristic method instead of 1, the diagonal values can be set to the average value of diagonal elements of the sparse matrix, however its effect should be compensated in the right hand side of the equation. This trick, results in better numerical results.



(a)



(b)



(c)

Figure 3.5: Contact node boundary condition and modified force calculation: a) zeroing all the elements of the corresponding row except the diagonal ones which are set to one; b) zeroing the columns to keep the matrix symmetric and modifying the force vector accordingly; c) calculation of the force at the contact node after obtaining the displacement vector.

Chapter 4

Conjugate Gradient Algorithm

Both static and dynamic FE modeling require that the solution to a large and sparse linear system of equations (i.e. (3.16) and (3.27)) be computed at each time-step in haptics simulation. In this chapter, the system of equations is denoted as $\mathbf{Ax} = \mathbf{b}$. Several direct and iterative methods are available in the literature for solving such systems [59, 60]. Based a comparison among existing linear solvers, we choose the iterative CG algorithm as the best approach for the intended application of this thesis. To implement the solver on an FPGA, the algorithm should be able to reliably work with fixed-point values and operations. A fixed-point implementation of the CG algorithm is proposed in Sec. 4.2. This is followed by a discussion of the effect of preconditioning on the convergence of the CG method. Accuracy and convergence properties of the fixed-point CG and Preconditioned CG (PCG) are analyzed through a set of numerical experimentations.

4.1 A Comparison Between Different Linear Solvers

One of the main advantages of iterative methods in the case of sparse systems of linear equations is their efficiency in terms of memory usage. For example, direct methods based on matrix factorization can lead to the decomposition of a sparse matrix into new matrices which are not as sparse as the original matrix. Iterative methods, in contrast, only need to store the non-zero elements of the main matrix. This reduces the usage of memory resources from an $O(N^2)$ to $O(N)$, where $N \times N$ is the size of the matrix. In general, iterative methods are more suitable for the intended application of this thesis as they lend themselves rather neatly to parallelization of the computations [59]. It should also be noted that in haptics applications, even when a linear elastic model is used, matrix $\mathbf{A}$ can change depending on the contact node. Moreover, nonlinear FE modeling of deformation can lead to a matrix $\mathbf{A}$ which would be dependent on the deformation $\mathbf{x}$. A changing $\mathbf{A}$ eliminates the possibility of an off-line calculation of the inverse matrix.

The CG method is one of the most prominent iterative approaches used for solving large systems of linear equations. The original algorithm requires that the matrix $\mathbf{A}$ be symmetric and positive-definite. Matrices arising from FEM or Finite Difference Method (FDM) are of this type. From a numerical perspective, the CG method is usually more robust and less computationally intensive than some other more general iterative methods such as BiCGStab or GMRES [59, 61].

4.2 Fixed-point Implementation of the CG method

A critical decision in hardware-based implementation of the CG algorithm is whether to work with fixed-point or floating-point operations. Floating-point/fixed-point representations approximate a range of real numbers with constant relative/absolute error, respectively. Due to the practical usefulness of constant relative error, which enables representation of a much larger dynamic range of values than fixed-point of the same size, modern general purpose computing platforms are almost universally equipped with double precision floating-point arithmetic units. The standard availability of such units has led to the large majority of existing scientific software to rely on floating-point implementations. However, in order to take full advantage of FPGA parallelism, the size of each individual computing unit should be minimized. Floating-point implementations have a high hardware cost, severely limiting the parallelism and thus the performance of the hardware accelerator. For most scientific calculations the use of high precision is motivated more out of the availability of the hardware in general purpose computers rather than the need for such high precision. In fact in many applications, a reduced precision floating-point or fixed-point calculation will often suffice.

Fixed-point computation poses its own unique challenges particularly in iterative methods such as the CG. The limited dynamic range of data representation in fixed-point computing can easily result in data overflows generating erroneous outcome. An absolute error representation leads to relative quantization errors that can become significant over time and hence may prevent the algorithm from converging to the actual solution. Special consideration are given to these issues in the design of the hardware accelerator proposed in this thesis. A key factor in the

success of the design is reaching a good tradeoff between error, calculation time and hardware cost.

A pseudo-code of the CG algorithm for solving $\mathbf{Ax} = \mathbf{b}$ for $\mathbf{x}$ is given below. $\#m$ is the maximum number of iterations and ε is the error tolerance:

1. $\mathbf{x} = \mathbf{init}$; %initial guess for solution of Ax=b
2. $\mathbf{r} = \mathbf{b} - \mathbf{A} \times \mathbf{x}$; %residue
3. $\mathbf{d} = \mathbf{r}$; %initial "search direction"
4. $cntr = 1$;
5. $rr = \mathbf{r}' \times \mathbf{r}$;
6. $rr0 = rr$;
7. while $(rr > \varepsilon^2 \times rr0 \quad \& \quad cntr < \#m)$
8. α = rr / $(\mathbf{d}' \times \mathbf{A} \times \mathbf{d})$;
9. $\mathbf{x} = \mathbf{x} + \alpha \times \mathbf{d}$; %update approximate solution
10. $\mathbf{rn} = \mathbf{r} - \alpha \times \mathbf{A} \times \mathbf{d}$; %update the residue
11. $rrn = \mathbf{rn}' \times \mathbf{rn}$;
12. $\beta = rrn/rr$;
13. $\mathbf{d} = \mathbf{rn} + \beta \times \mathbf{d}$; %update search direction
14. $\mathbf{r} = \mathbf{rn}$;
15. $rr = rrn$;

16. $cntr = cntr + 1$;

17. end

In this thesis two types of scalings, namely *static* and *dynamic* are proposed for the fixed-point implementation of the CG method in order to improve its numerical accuracy, reliability and convergence rate. From the first three(3) lines of the pseudo-code, it can be observed that $\mathbf{b}$, $\mathbf{r}$ and $\mathbf{d}$ vectors are in a different scale compared to $\mathbf{x}$ vector since they are all computed based on $\mathbf{Ax}$. If 16-bit and 12-bit words were chosen to represent the vector and matrix elements, respectively, it is easy to see that the results of matrix by vector multiplication $\mathbf{Ax}$ can easily exceed the designated word length. This is particularly problematic if the vector $\mathbf{x}$ were to utilize its full 16-bit range. In order to be able to represent the results of the multiplications in 16 bits, as shown in Fig.4.1, $\mathbf{b}$, $\mathbf{r}$ and $\mathbf{d}$ are scaled down by a *static scaling* factor, i.e. shifted to the right by m bits where *static scale* $= 2^m$.

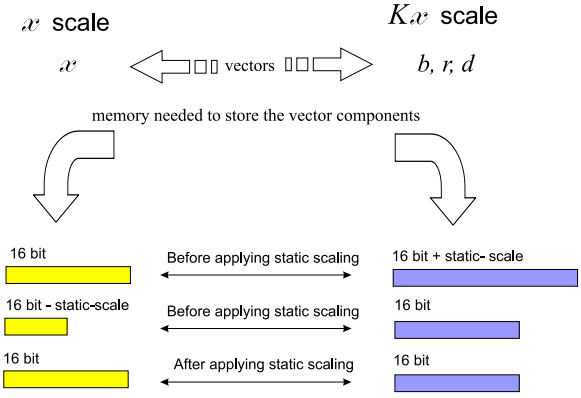


Figure 4.1: Static-Scaling

The proposed scaling can significantly reduce memory utilization since we still use only 16-bit words for those vectors while preventing overflow in the multiplication results. This, however, is gained at the expense of some loss of data in the

least significant bits. The proper scaling factor m generally depends on the norm of the actual matrix $\mathbf{A}$. However assuming that the matrix elements are scaled to utilize their full 12-bit resolution, a constant scaling factor in the order of 9-12 bits would usually work under most circumstances.

As the CG algorithm progresses towards the final solution, it is expected that norms of $\mathbf{r}$ and $\mathbf{d}$ vectors gradually decrease. Consequently relative quantization errors in $\mathbf{r}$ and $\mathbf{d}$ become significant resulting in large errors in the updated $\mathbf{x}$ vector. Conceptually, the $\alpha\mathbf{d}$ steps taken towards the solution in Line 9 of the algorithm become gradually smaller and as such a finer resolution is required to properly represent these steps. To address this issue, a *dynamic scaling* is proposed to re-scale the $\mathbf{r}$ and $\mathbf{d}$ vectors when the value of $\|\mathbf{r}\|$ falls below a certain threshold. Finally, it should be pointed out that the nearest rounding mode has been employed in all division operations.

The convergence properties of the proposed fixed-point implementation of the CG algorithm were examined using Matlab's Fixed-point toolbox. A linear elastic model with a 3D mesh of 1000 nodes was used in the tests. The vectors were represented by 16-bit signed numbers whereas the matrix elements were 12-bit singed numbers. Scalers $\mathbf{d^T A d}$, $\mathbf{r^T r}$, and $\mathbf{r_n^T r_n}$ were represented by 64-bit signed values whereas α and β were 16-bit signed numbers. The tests were repeated with different random $\mathbf{b}$ vectors and $\mathbf{A}$ matrices and in each case the algorithm started from a zero initial guess. The condition number of $\mathbf{A}$ matrices was within the range of $2500 - 4000$. The average results of these numerical experiments are given in Table 4.1 where $\mathbf{x_0}$ is the actual solution and $\mathbf{err}$ is the error vector in the solution of the fixed-point CG algorithm. Fig. 4.3 shows one of the numerical test results,

plotting the $\mathbf{x_0}$ and $\mathbf{err}$ in the same frame. The error vector is based on a solution obtained with an initial guess of zero and after 60 iterations of the fixed-point CG algorithm.

# CG iterations	30	40	50	60	70	80
$\|\mathbf{x_0}\|$	2.47e+5					
$\|\mathbf{err}\|$	6422	5336	4655	3875	2429	1255
$\|\mathbf{err}\|/\|\mathbf{x_0}\|$	2.6e-2	2.2e-2	1.9e-2	1.6e-2	9.8e-3	5.1e-3

Table 4.1: Simulation results for fixed-point CG algorithm.

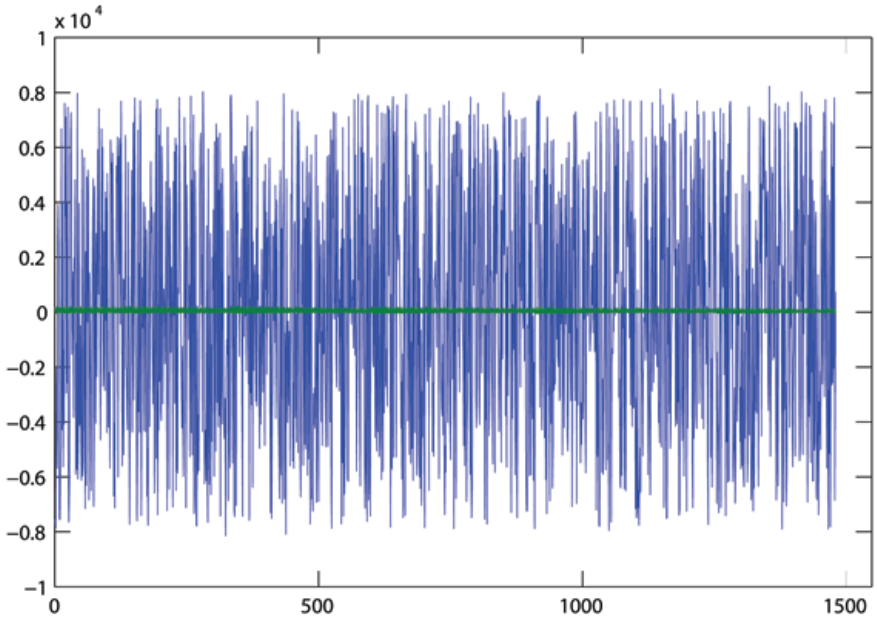


Figure 4.2: Error vector v.s. the exact solution for CG algorithm in 60 iterations.

In haptics applications, the update rate is in the order of 300-1000Hz and therefore the change in object deformation within one sample period is expected to be small. This implies that the solution at the previous sample time is usually a good initial guess for the CG algorithm at the present time step. Given that the results

of Table 4.1 were actually calculated with an initial guess of zero, it can be concluded that 60 iterations of the fixed-point CG algorithm would provide accurate results for a 3D mesh of 1000 nodes. In the hardware-based implementation of the CG algorithm, the number of iterations can be fixed since exiting the algorithm at an earlier iteration does not provide any timing advantage. However, this would simplify the architecture design to some extent.

4.3 Preconditioning

In theory the CG algorithm is guaranteed to find the solution to $\mathbf{Ax} = \mathbf{b}$ at a number of iterations not exceeding the dimension of the unknown vector $\mathbf{x}$. In practice a solution is usually obtained within a much smaller number of steps. The actual number of iterations required to converge to the solution depends on the size of the problem, initial guess, desired error tolerance, and the condition number of the matrix $\mathbf{A}$. In hardware-based FE simulation, the size of the problem and the number of iterations are fixed and known, and the starting point is usually close to the actual solution. It is required that the algorithm converge to the error tolerance sphere at the given number of iterations. However due to changes in the matrix condition number, the convergence to the solution is not necessarily guaranteed. The condition number could depend on the quality of the FE mesh and the tissue model parameters. Moreover, the condition number of FE models of second-order elliptic boundary value problems posed on d-dimensional domains is in the order of $\kappa \sim O(N^{2/d})$ and increases by the problem size [29].

Preconditioning techniques [60] can be employed to improve the condition number of the matrix $\mathbf{A}$ and ensure that a solution is obtained within the fixed

number of iterations imposed by the real-time response constraint. In the so-called left preconditioning, $\mathbf{Ax} = \mathbf{b}$ is replaced with

$$\mathbf{P^{-1}Ax} = \mathbf{P^{-1}b} \tag{4.1}$$

where $\mathbf{P}$ is an approximation of $\mathbf{A}$ whose inverse is much easier to compute. The idea is that $\mathbf{P^{-1}A}$ would have a smaller condition number than that of the original matrix $\mathbf{A}$. In practice instead of using (4.1), the preconditioning is directly integrated into the CG algorithm [29]. A pseudo-code for the preconditioned conjugate gradient (PCG) is given below. The cost of hardware implementation of a simple PCG like the *Jacobi* method [60] will be discussed later in the thesis.

1. $\mathbf{x} = \mathbf{init}$; %initial guess for solution of Ax=b
2. $\mathbf{r} = \mathbf{b} - \mathbf{Ax}$; %residue
3. $\mathbf{z} = \mathbf{P^{-1}r}$;
4. $\mathbf{d} = \mathbf{z}$; %initial "search direction"
5. $cntr = 1$;
6. $zr = \mathbf{z'^{T}r}$;
7. $zr0 = zr$;
8. while $(cntr < \#m)$
9. α = zr/$(\mathbf{d'^{T}Ad})$;
10. $\mathbf{x} = \mathbf{x} + \alpha\mathbf{d}$; %update approximate solution

11. $\mathbf{rn} = \mathbf{r} - \alpha\mathbf{Ad}$; %update the residue
12. $\mathbf{zn} = \mathbf{P^{-1}rn}$;
13. $zrn = \mathbf{zn'^{T}rn}$;
14. $\beta = zrn/zr$;
15. $\mathbf{d} = \mathbf{zn} + \beta\mathbf{d}$; %update search direction
16. $\mathbf{r} = \mathbf{rn}$;
17. $\mathbf{z} = \mathbf{zn}$;
18. $zr = zrn$;
19. $cntr = cntr + 1$;
20. end

As will be seen in Section 5.6, the word length of $\mathbf{d}$ vector has a significant impact on the memory usage and the main reason for using static scaling is to restrict its size. From the first four(4) lines of the pseudo-code it is evident that, unlike in the original CG algorithm, $\mathbf{d}$ vector is not in the same scale of $\mathbf{b}$ and $\mathbf{r}$ and hence requires no scaling. Therefore, an improvement in the condition number of the matrix $\mathbf{A}$ also helps avoid information loss due to static scaling.

The results of numerical simulations for the fixed-point implementation of the Jacobi-based PCG for a 1000-node 3D FE mesh are summarized in Table (4.2). A comparison of the data with the results in Table 4.1 shows that preconditioning has reduced the error by an order of magnitude using the same number of iterations.

Fig. 4.3 demonstrates a random $\mathbf{x_0}$ vector and its corresponding error vector produced by the fixed-point PCG in 60 iterations. Zero vector is chosen as the initial guess for the solution.

# PCG iterations	30	40	50	60	70	80
$\|\mathbf{x_0}\|$	2.47e+5					
$\|\mathbf{err}\|$	4146	2652	599	309	239	220
$\|\mathbf{err}\|/\|\mathbf{x_0}\|$	1.7e-2	1.1e-2	2.4e-3	1.3e-3	1.0e-3	9.0e-4

Table 4.2: Simulation results for fixed-point PCG algorithm.

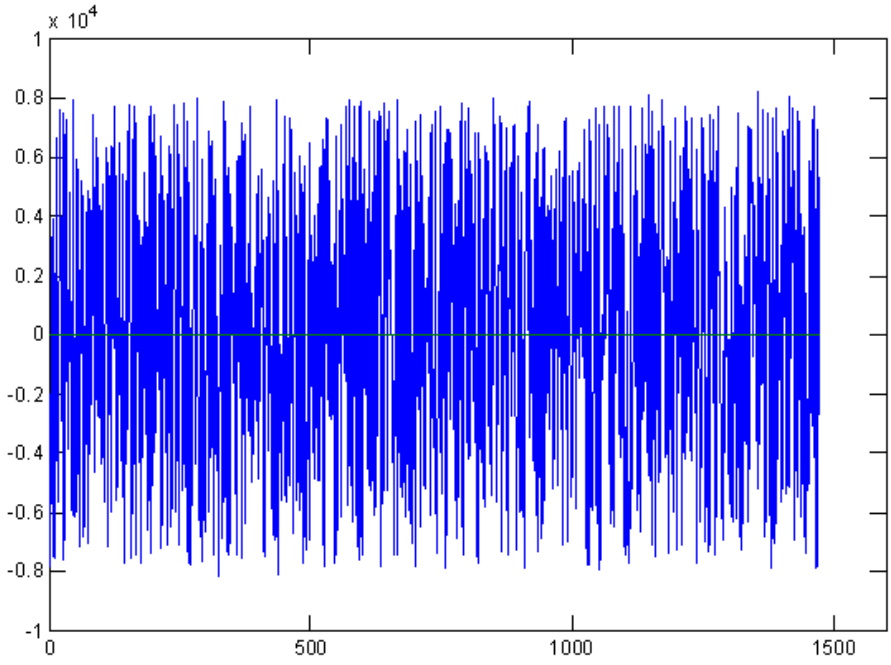


Figure 4.3: Error vector v.s. the exact solution for PCG algorithm in 60 iterations.

4.4 Data Dependence Analysis

A data dependence analysis is needed to discover the instruction-level parallelism (ILP) in the CG and PCG algorithms. This information will be instrumental in the design of efficient hardware architectures for the implementation of the algorithms in the next chapter. The analysis is performed for the computations in the "while loop" that must be repeated at each iteration of the algorithm. From the pseudo-code of the CG method in 4.2, it is clear that the inner vector product of $\mathbf{d^T Ad}$ (Line 8) can be executed in parallel with the matrix by vector multiplication $\mathbf{Ad}$. In particular once the multiplication of five(5) nodes (15 rows) by the vector is complete, the results can be multiplied by the corresponding elements from the vector $\mathbf{d}$ and be accumulated in temporary buffers to be stored in the memory. In Line 8 of the algorithm, α can be calculated using the result of the vector inner product $\mathbf{d^T Ad}$. Due to the data independency between $\mathbf{r_n}$ and $\mathbf{x}$ in Lines 9 and 10, these vectors can be updated in parallel. The inner product of $\mathbf{r_n}^\mathbf{T}\mathbf{r_n}$ (Line 11) can also be computed as the components of the $\mathbf{r}$ vector are being updated. Similar to α, β in Line 12 is calculated in series. Finally, the scalar by vector multiplication as well as the vector addition in updating $\mathbf{d}$ vector in Line 13 can be executed in parallel. Fig. 4.4 displays a block diagram of the computation flow in the CG algorithm.

Cost of Preconditioning

The overhead cost of using a Jacobi preconditioner can be estimated as follows. Based on the PCG pseudo-code given in Section 4.3, one additional vector $\mathbf{z}$ has to be defined and stored in memory. Matrix $\mathbf{P}$ in *Jacobi* preconditioning is constructed

from the elements of the main diagonal of matrix $\mathbf{A}$. Therefore, $\mathbf{P}^{-1}$ can be simply stored as a vector containing the inverse values of the diagonal entries. As stated earlier, avoiding to use static scaling results in longer word lengths for vectors $\mathbf{b}$ and $\mathbf{r}$ to circumvent any overflow, consequently causing a slight increase in the memory resource usage. In the "while loop" of PCG pseudo-code, the task of updating vector $\mathbf{z}$ using N scalar multiplications in Line 12 is the main addition to the original CG algorithm. This task requires some extra multipliers but since vector $\mathbf{z}$ can be updated in parallel with vector $\mathbf{r}$ the computation timing would not be affected.

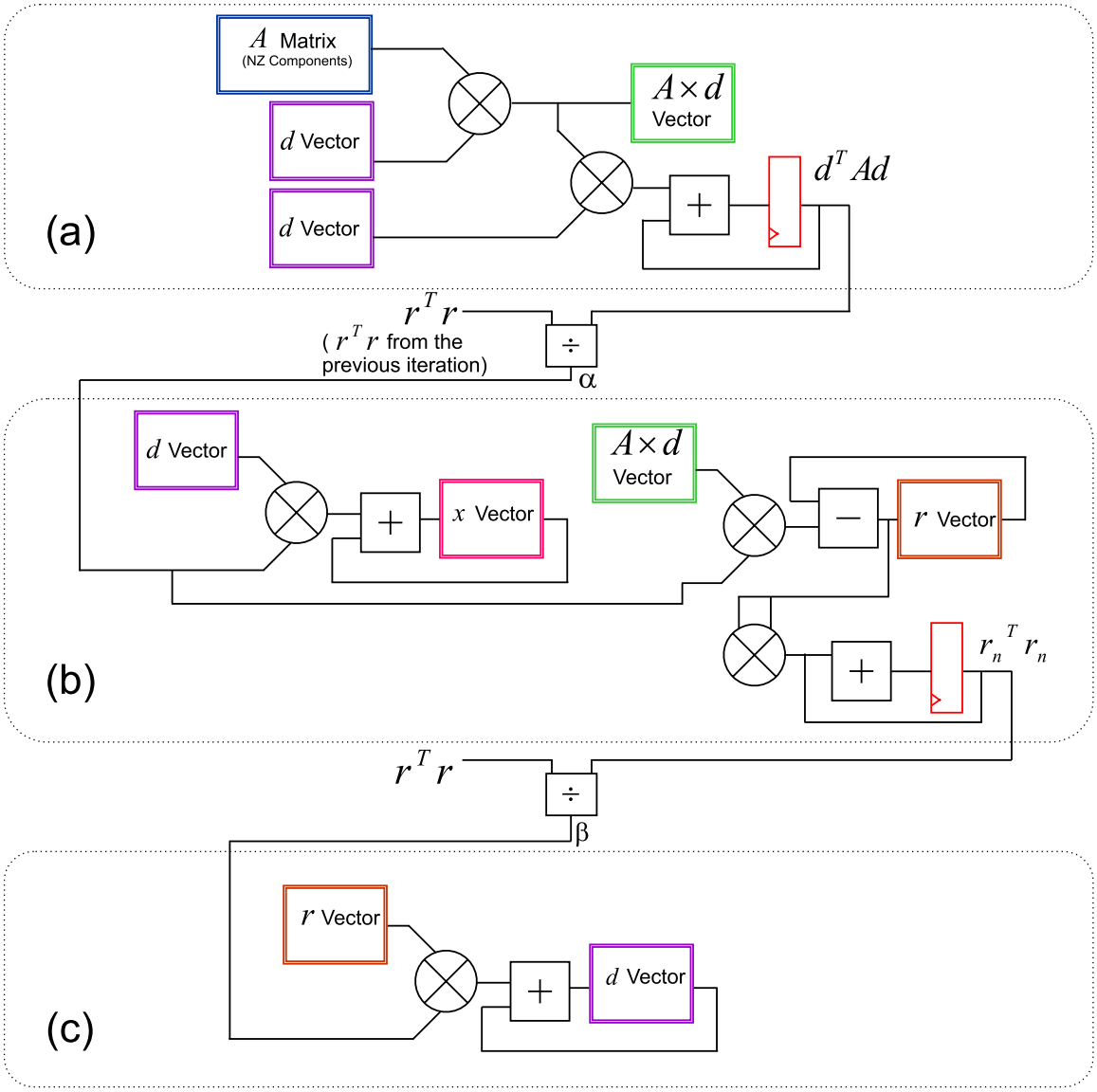


Figure 4.4: Computation flow in the CG algorithm: *a*, *b* and *c* group the tasks that run in parallel with each other. The three dashed blocks are executed sequentially one after the other; α and β are calculated at the borders; (a) $\mathbf{Ad}$ and $\mathbf{d'^T Ad}$; (b) $\mathbf{x_n} = \mathbf{x} + \alpha\mathbf{d}$, $\mathbf{r_n} = \mathbf{r} - \alpha\mathbf{Ad}$ and $\mathbf{r_n}^\mathbf{T}\mathbf{r_n}$; (c) $\mathbf{d_n} = \mathbf{r_n} + \beta\mathbf{d}$.

Chapter 5

Hardware Architecture

This chapter discusses the details of hardware architecture design for the CG algorithm to solve a set of linear equations in the form of $\mathbf{Ax} = \mathbf{b}$. The main objectives of the design are speed, scalability of the solution, and optimal usage of resources. Through a high degree of parallelization of the computations, the hardware architecture must meet the real-time response requirement of soft-tissue haptic interaction. The hardware implementation should be compatible with matrices of different sizes and different structures. The available on-chip resources should also be optimally utilized to increase the size of the largest FE mesh that can be processed.

The hardware architectures presented here are the result of numerous design iterations and revisions to satisfy the objectives stated above. The details of the design steps are rather lengthy and are omitted for brevity. Instead, an overview of the problem, its unique properties and some of the challenges involved in attaining the design objectives is given in the following sections. A particular emphasis is placed on describing parallel usage of multipliers at different architectural layers for implementing the sparse matrix by vector multiplication in the CG algorithm

(i.e. Line 8 of the pseudo-code in Sec. 4.2). The scalability of the overall solution will be discussed at the end of this chapter.

5.1 Data Structures

For notational convenience, throughout the rest of the thesis $\mathbf{K}$ represents the equivalent sparse matrix obtained through static or dynamic FE modeling $\big($i.e. Eqs. (3.16) and (3.24)$\big)$ and $\mathbf{U}$ denotes the node displacement vector. The design of the hardware architecture can be optimized to fully exploit any special data structure in the CG-based solution of $\mathbf{KU} = \mathbf{b}$. The data include the equivalent stiffness matrix $\mathbf{K}$, the unknown vector $\mathbf{U}$, the known vector $\mathbf{b}$, the search direction vector $\mathbf{d}$, and the residue vector $\mathbf{r}$ [1]. Among these variables, the matrix $\mathbf{K}$ constitutes the largest data block to be stored and processed by the hardware accelerator. A 3D FE mesh with n nodes would result in a $\mathbf{K}$ matrix of the size $3n \times 3n$. In this context, each node is associated with three consecutive rows of the matrix $\mathbf{K}$ corresponding to its x, y, and z displacements. The non-zero elements of the matrix are due to connections among the corresponding nodes in the mesh. The number of these connections is determined by the meshing algorithm used and is nearly independent from the number of the nodes. Since each node in the mesh is usually connected to only a few neighbor nodes, the resulting matrix $\mathbf{K}$ is sparse. The sparsity of the matrix grows by number of nodes. Our numerical experimentation has shown that the number of non-zero (NZ) elements in the matrix $\mathbf{K}$ is typically less than 3% of the size of the matrix for meshes with more than 400 nodes.

The matrix $\mathbf{K}$ is also symmetric and positive definite meeting the requirements

[1] $\mathbf{r}$ and $\mathbf{d}$ are some internal vectors in the CG algorithm, defined in the CG pseudo-code in Sec. 4.2

of the CG algorithm. Its other useful property that can be understood through the assemblage process expressed in Eq. (3.19), is that each three(3) rows/columns of $3i,\ 3i-1,\ 3i-2$ (for $i = 1, \cdots, n$), as shown in Fig. 5.1, have the same indices of non-zero elements. This implies that the non-zero components in $\mathbf{K}$ for 3D meshes are in blocks of 3×3, regardless of the meshing algorithm employed. The vectors $\mathbf{r}$, $\mathbf{d}$, $\mathbf{b}$, and $\mathbf{U}$ usually have no particular pattern that can be exploited in the hardware design.

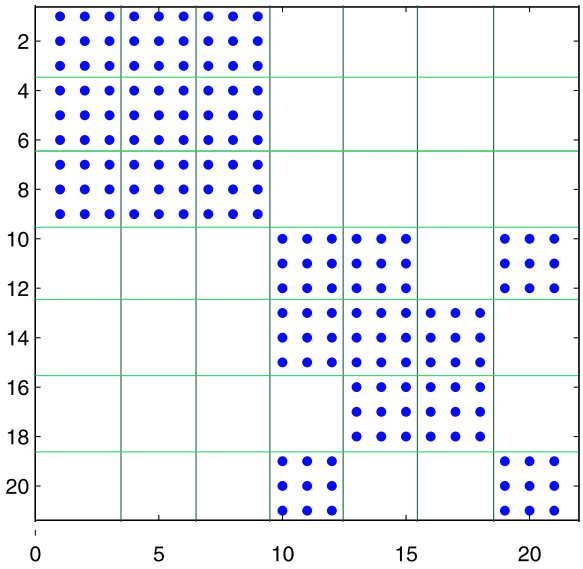


Figure 5.1: 3x3 Blocks of non-zero elements of a portion of $\mathbf{K}$ matrix with the corresponding node numbering.

5.2 Sparse Matrix by Vector Multiplication

The computational bottleneck of the CG algorithm in Sec. 4.2 is the sparse matrix by vector multiplication (SpMxV). The hardware architecture should be able to utilize the sparsity of the matrix $\mathbf{K}$ in order to meet the real-time computation

constraint of haptic simulations. At the same time, it must be reasonably scalable by the problem size and be independent from the meshing algorithm used.

A full matrix by vector multiplication is impractical both in terms of memory usage as well as the time required to complete the task. Graph partitioning techniques [60,62] can be employed to reorder the rows and columns of the matrix such that the non-zero elements fall into isolated sub-blocks which have a denser structure compared with that of the original matrix. The multiplications can then be performed on these smaller blocks which may or may not have overlap with each other, e.g. see Figs. 5.2. Multiplication based on matrix partitioning can significantly simplify the hardware architecture design. However, experimentation with various graph partitioning methods such as the vertex based greedy method, the element based method, and the edge based approach using METIS [63] revealed that typically no more than 60% of the zeros may be isolated. This is far less than the 97% sparsity of the original matrix. The architecture proposed here mainly stores the non-zero elements of the matrix[2] and hence utilizes its sparsity. This is achieved at the expense of having to design a memory indexing logic. In addition to the data values, the indices of the non-zero components in the sparse matrix are saved separately and are used to drive the address lines of the multiplicand vector in SpMxV.

[2]Our analysis in Section 5.4 shows that the proposed approach in average stores 11 % extra components as non-zero elements; however considering the sparsity of the matrix, this extra memory usage is relatively minor.

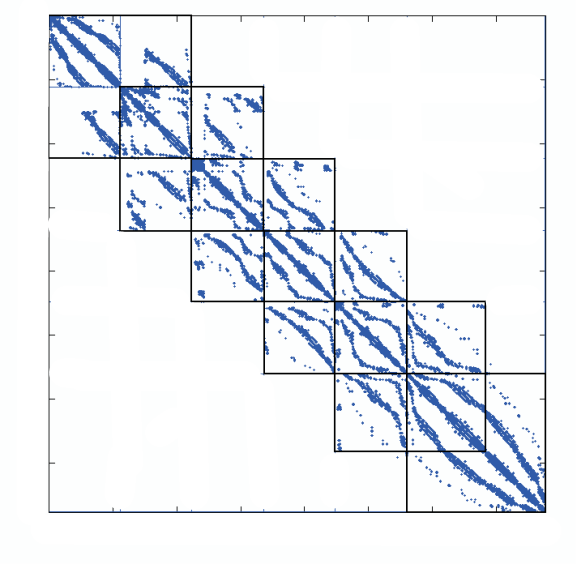


Figure 5.2: Graph partitioning is used to isolate non-zero elements of the matrix.

5.3 Hardware Parallelism

To meet the real-time computation requirement, in principle, the hardware architecture must employ as many number of multipliers as possible in parallel. However memory bandwidth limitation restricts the amount of data that can be transferred to the multipliers at each clock cycle posing a challenging design problem. To demonstrate the trade-off involved between the complexity of the architecture and its use of resources, one may consider the case in which m multipliers are used to multiply m rows of the matrix $\mathbf{K}$ by the vector $\mathbf{d}$. Since the multipliers access the same $\mathbf{d}$ vector simultaneously, there is always potential for memory contention. A simple solution that avoids a complex contention resolution logic circuitry is to create multiple dedicated copies of the vector $\mathbf{d}$ that can be accessed independently. However this is obviously not a viable solution given the limited amount of on-chip memory and the restricted bandwidth of off-chip memory access.

In the proposed architecture, parallelism is achieved at several levels. Based on the ordered structure of the NZ elements in the matrix $\mathbf{K}$, each three(3) consecutive rows can be stored and accessed in three(3) similar blocks of memory with the same addressing logic. This results in parallel processing of three(3) values from three(3) rows at a time at the *first level* (*PL1*), i.e. see Fig. 5.3.

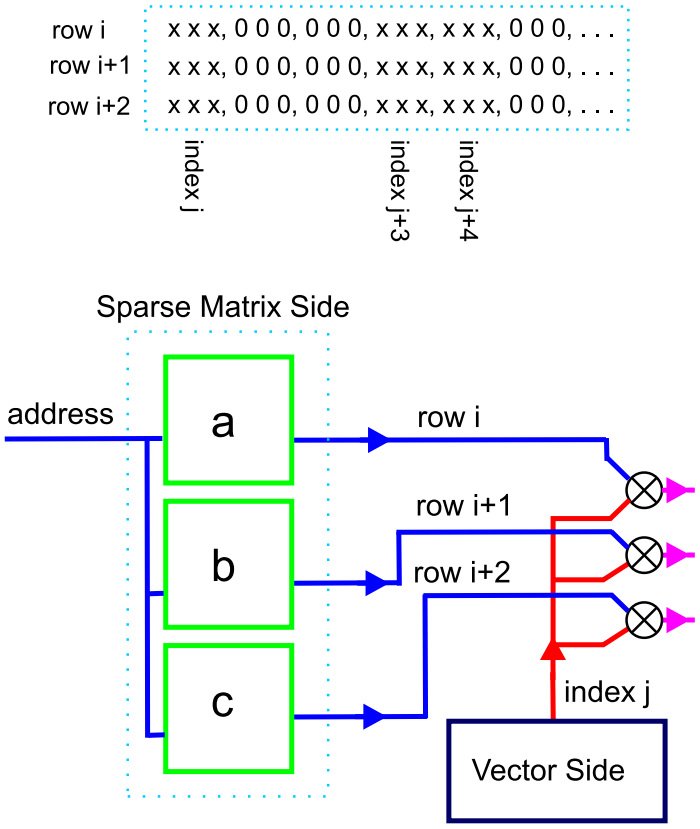


Figure 5.3: First level of parallelization: multiplication of three(3) rows by the vector $\mathbf{d}$.

The parallelism can be further increased by processing several elements of each node at the same time. This increase is determined by the memory structure explained in the next section. Given the memory resources of our current FGPA device, the maximum number of columns per node that are processed concurrently

is restricted to eight(8). This constitutes the *second level* (*PL2*) of parallelization. Fig. 5.4 shows the structure of the 3x8 multipliers and accumulation (MAC) units used for each node.

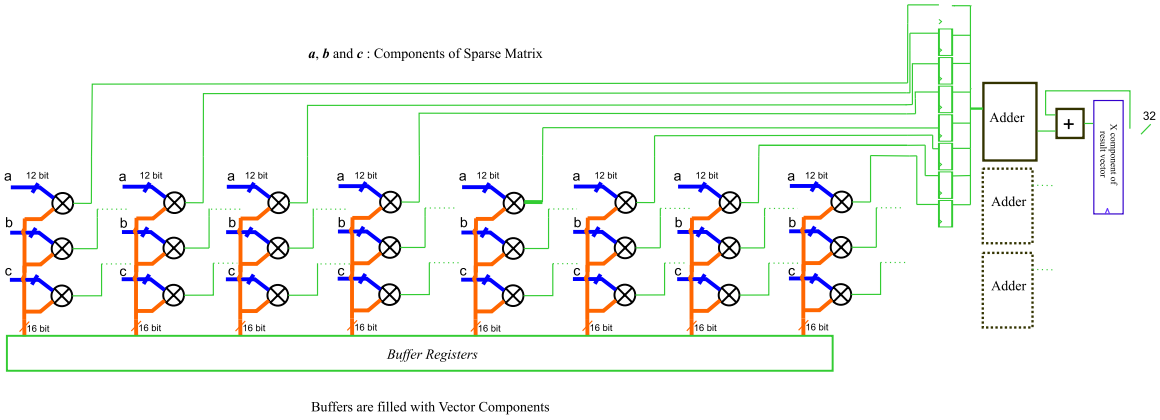


Figure 5.4: Second level of parallelization: the 3x8 MAC units structure per three(3) consecutive rows of the matrix.

As mentioned earlier, the number of the NZ entries per row in the matrix $\mathbf{K}$ is determined by the number of the nodes connected to the corresponding node in the mesh. Our numerical experiments have shown that the maximum number of NZ elements per row using the particular meshing algorithm used in this work does not exceed 54. This number may vary if a different meshing routine is employed. The proposed architecture can employ s series of multiplications per node and hence be capable of processing up to $8 \times s$ non-zero elements per each row of the sparse matrix. A proper value of s can be determined based on the meshing algorithm used. Fig. 5.5 depicts the connection of the MAC units with the memory blocks containing the multiplicand vector and a portion of the sparse matrix. This block which is based on the first and second level of parallelization is referred to as 'SpMxV sub-block' hereinafter.

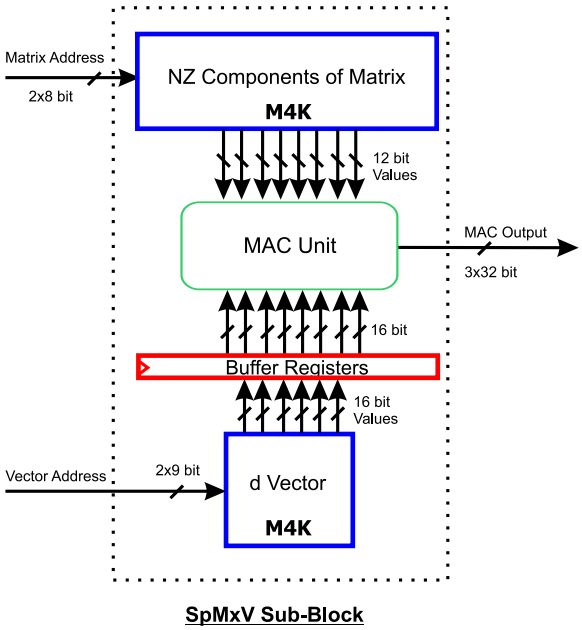


Figure 5.5: The connection of MAC units to memory blocks.

In the *third level* (*PL3*), the parallelism is further improved by dividing the matrix **K** into multiple partitions, each containing several nodes as shown in Fig. 5.6. These partitions should be balanced in terms of number of their NZ elements. Each new partition requires one extra SpMxV sub-block as shown in Fig.5.7.

Obviously a lager number of partitions leads to a higher degree of parallelization. In our case this number is limited to five(5) due to the available on-chip memory and multiplier units. The matrix partitioning allows for concurrent utilization up to 120 multipliers per FPGA device for sparse matrix by vector multiplication.

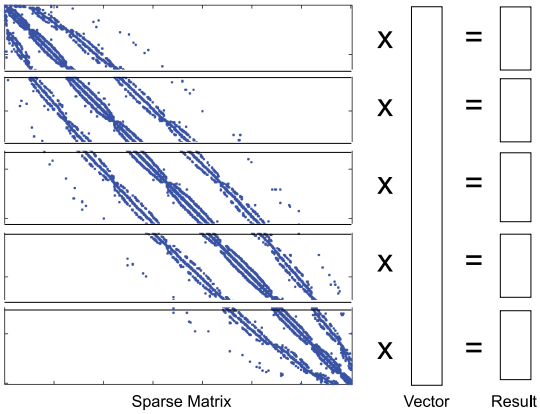


Figure 5.6: Matrix partitioning for increased parallelism.

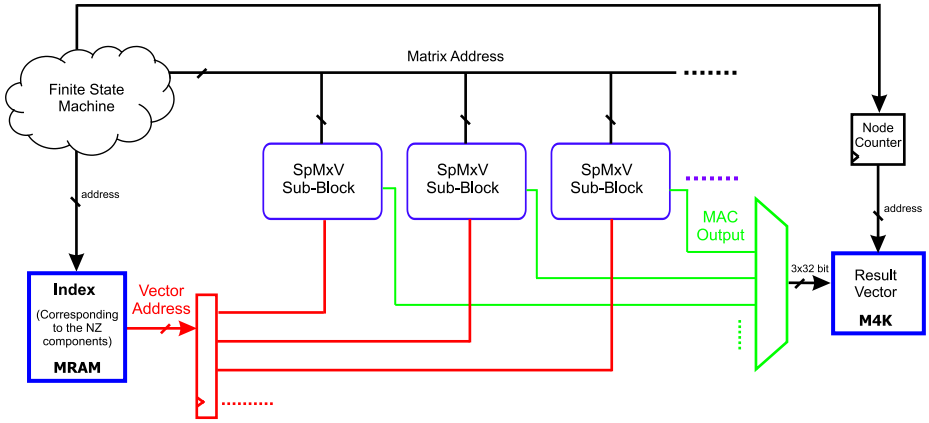


Figure 5.7: Third level of parallelization: top-view block diagram of the hardware-based sparse matrix by vector multiplication.

5.4 Memory Structure

The data structure discussed in Sec. 5.1 can be divided into two groups, namely the non-zero components of the sparse matrix, and the vectors used in the CG algorithm. In this section some details on the memory structure used for each data type are given. This discussion will help provide an insight into the operation of the proposed micro-architecture.

Table 5.1 summarizes different types of memory resources available on the

FPGA device used in work. All these memory blocks can be arranged in a dual-port configuration. In the current design, M4K blocks are used to store the NZ elements and vectors. MRAM blocks are utilized to store corresponding indices of the NZ components. M512 blocks are not currently employed in the micro-architecture.

Feature	EP2S60 FPGA Device
M512 RAM Blocks (512 bits + Parity)	329
M4K RAM Blocks (4 Kbits + Parity)	255
MRAM Blocks (512 Kbits + Parity)	2
Embedded Multipliers (18×18)	144

Table 5.1: Altera Stratix II EP2S60 Overview

The basic memory structure for matrix components as shown on Fig. 5.8 on Page 63 consists of three(3) M4K memory blocks, capable of storing 1 K 12-bit values. Multiple number of these basic blocks together construct the memory units a, b or c shown in Fig. 5.3. Using two(2) address ports, the available data bandwidth enables reading/writing of eight(8) 12-bit values at the same time. According to this bandwidth, $PL2$ as shown in Fig 5.5, is bounded to eight(8). SpMxV is processed as an internal multiplication of the vector by each row of the matrix and in this process, no data value from other rows should be accessed to avoid complexity in memory indexing logic. For this reason, the NZ components of each row in the matrix are bundled in packages of eight(8) while writing the data into memory blocks. This will result in storing some redundant values in the rows that their NZ components number is not a factor of eight(8). For a matrix of dimension N with

NNZ number of NZ components, the additional storage cost per row in average is four(4) 12-bit values, i.e.

$$Extra\ Memory\ Resource\ Usage\ Ratio\ = \frac{4 \times N}{NNZ} \tag{5.1}$$

where NNZ is proportional to N. In our numerical experiments for meshes with more than 400 nodes, $NNZ \simeq 36 \times N$ but this number could change using other meshing algorithms. According to Eq. (5.1), the additional memory usage cost is computed as 11% in average.

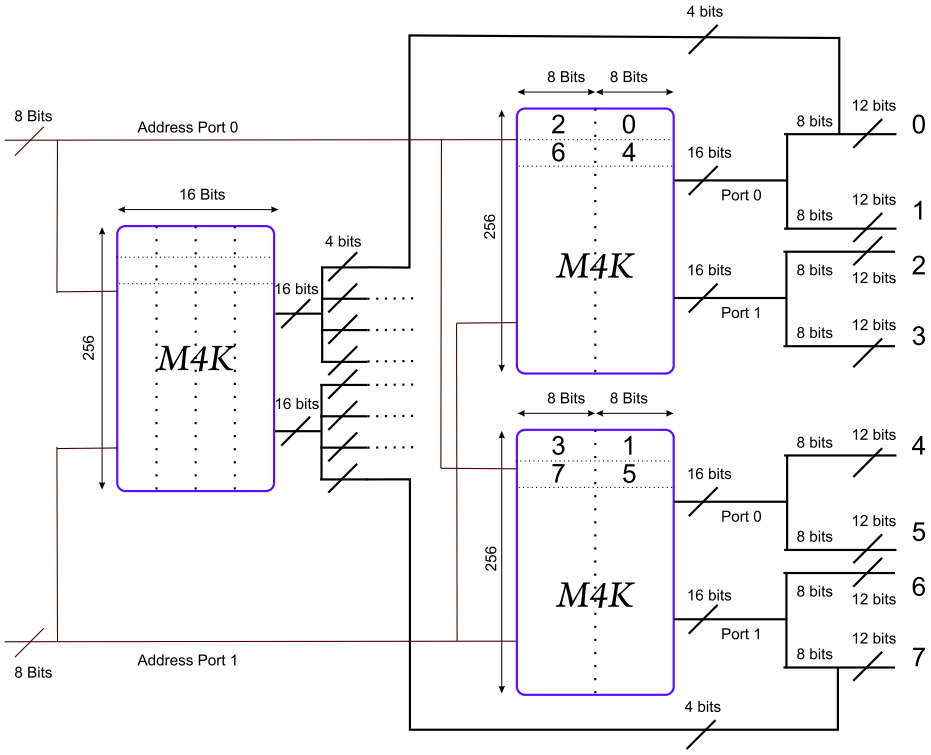


Figure 5.8: Basic memory structure for NZ components of matrix

The basic memory unit used for storing vector data is simply constructed by three similar blocks each storing either of $3i,\ 3i-1,\ 3i-2$ (for $i = 1, \cdots, n$) values

of the vector. Given the dual-port access in each of these memory blocks, the proposed structure provides a bandwidth of 6×16 bits per clock cycle. The proposed memory structure is depicted in Fig. 5.9. Using 3 or 4 M4Ks in each one of the memory blocks, the bandwidth can be easily increased to 6×24 or 6×32 per clock cycles. [3]

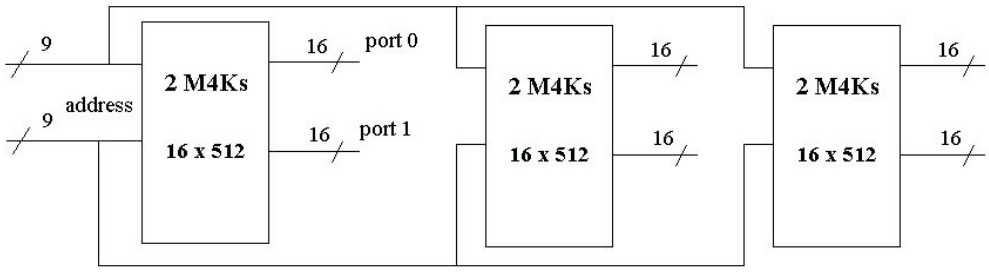


Figure 5.9: Basic memory structure for vectors in the CG algorithm

5.5 Critical Path

According to Quartus II timing analyzer, the maximum operating frequency (f_{max}) in the current design is limited to 100 MHz. As shown in Eq. (5.2), this frequency is determined based on the path with the longest propagation delay (Δt_{cr}) in the circuit. The path can be either connecting an external input to a register, between two registers or from a register to an external output. This path is referred to as the critical path.

$$f_{max} = \frac{1}{\Delta t_{cr}} \tag{5.2}$$

The critical path in our architecture is due to the vector-vector multiplication $\mathbf{d^T(Ad)}$ in the CG algorithm, discussed in Chapter 4. It should be noted that $\mathbf{d}$ and $\mathbf{Ad}$

[3] Using the parity bits, it is also possible to have 6×18, 6×27, 6×36 bandwidths.

vectors are 16 and 32 bit respectively. Therefore to enhance the utilization of the resources, 16×16 multipliers have been employed instead of 32×32 multipliers. Fig. 5.10 shows how this 32×16 multiplication is broken into two 16×16 multiplications. In this figure, LSB and MSB denote the least significant and most significant 16-bits of the 32-bit values.

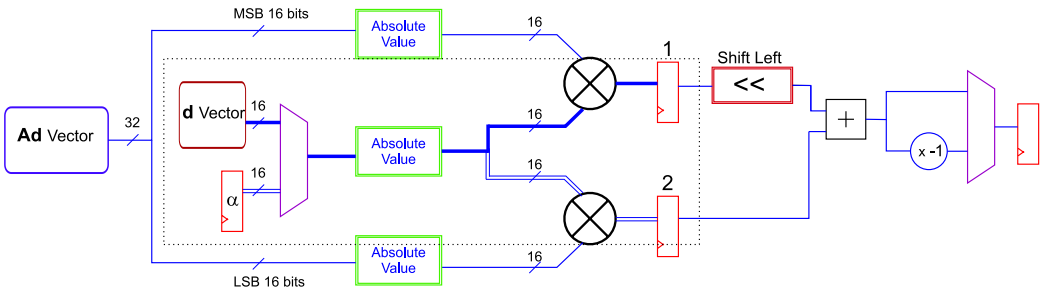


Figure 5.10: Critical path

The critical path shown in a dashed box in Fig. 5.10 consists of a 2x1 multiplexer, a 16-bit absolute value circuit and a 16×16 multiplier.

5.6 Scalability

The whole design can be extended to process FE meshes of larger sizes if an FPGA with more memory and multiplier units is employed. It is easiest to scale up *PL3*. This is due to the fact that the first and second levels of parallelization (*PL1* and *PL2*) are based on the sparse matrix structure and the available memory bandwidth, whereas the third level (*PL3*) is mainly restricted by the number of multipliers and memory blocks. For each additional matrix partition in Fig. 5.6 one extra "SpMxV sub-block" displayed in Fig. 5.7 is required. Obviously the NZ memory

block in "SpMxV sub-block" in Fig. 5.5 would only contain the portion of the non-zeros that are in the corresponding partition of the matrix in Fig.5.6. However the vector $\mathbf{d}$ as a memory block in "SpMxV sub-block" has to be replicated for each matrix partition. This underlines the significance of the choice of the word length for this vector in terms of memory usage. In summary, the third level of parallelization can be scaled up using an FPGA with larger number of multipliers and memory blocks. An extension of the proposed hardware architecture to a double-FPGA configuration is discussed below.

Chapter 6

Performance Analysis and Experimental Results

In the previous chapters some basic details of soft-body deformation modeling with an emphasis on FE formulation were reviewed. Fixed-point implementation of the iterative CG algorithm to solve a linear system of equations was discussed. A scalable FPGA-based hardware architecture for parallel implementation of the CG was proposed. In this chapter, the performance of the proposed architecture in accelerating the computations is studied. Moreover to demonstrate the effectiveness of the proposed approach, hardware-in-the-loop haptic simulations for interaction with deformable objects using linear FE models have been conducted. The experimental setup and the results are also reported in this chapter.

6.1 Hardware Accelerator Performance

Following the discussion on the scalability of the design in Section 5.6, an estimation of Giga Operations Per Second (GOPS) and the number of the multipliers used in SpMxV are provided in Table 6.1. The operations needed for SpMxV in the ideal case includes *NNZ* multiplications and *NNZ-N* additions, where *NNZ* is the number of the non-zero components of the sparse matrix, and N is the vector length. In this table, *PL3* denotes the number of the sub-partitions of the matrix in Fig. 5.6 in the third level of parallelization. $PL3$ is the easiest level of parallelization for scaling and hence is chosen as the determining factor for scalability in Table 6.1.

Number of Multipliers	$24 \times PL3$
GOPS	$(36 \times PL3)/clock\ cycle$

Table 6.1: Estimation of GOPS and number of the multipliers for SpMxV.

The proposed microarchitecture has been implemented on Altera Stratix II EP2S60 FPGA device. Table 6.2 summarizes the resource usage in the FPGA. In our design, given the hardware resource limitations $PL3$ is bounded to 5. In Table 6.1, the number of multipliers used for SpMxV multiplication is 120. In addition 21 other multipliers are used in parallel for vector-vector and scalar-vector multiplications in the implementation of the CG algorithm. These multipliers use the configuration of 18x18 bit sizes and are capable of running at 450 MHz.

The current FPGA implementation with $PL3 = 5$ leads to SpMxV kernel that operates at a rate of 18 GOPS. In Table 6.3, the performance of the proposed architecture has been compared with that of three different microprocessors used in the

Logic Utilization		Memory Blocks		Multipliers	Freq.
Combinational ALUTs	Dedicated Logic Reg.	M4K	MRAM		
8.4 k	10.5 k	251	2	141	100 MHz

Table 6.2: Resource utilization in FPGA and clock speed.

experiments conducted by Goumas et al. [64]. The last column provides a comparison between the average results in the fourth column to the speed of computation in our proposed architecture. It is evident that the massive low-level parallelism using fixed-point operations in our approach has yielded a significant speed up in the calculations over general-purpose floating point processors.

Processor	Clock Speed	L2 Cache	Average MFLOPS	xSlower
Woodcrest	2.6 GHz	4MB	495.53	x37
Netburst	2.8 GHz	1MB	297.88	x61
Operton	1.8 GHz	1MB	273.72	x66

Table 6.3: Comparative performance of SpMxV kernel on general-purpose CPUs [64] and the proposed FPGA-based hardware accelerator.

It should be noted in contrast to the mentioned processors, our proposed micro-architecture is using fixed-point operations rather than floating-point. Nonetheless given the test results presented in Sec. 6.2, it can be concluded that for the applications considered in this thesis, the proposed parallel computing platform can provide a huge speed up in calculations with a sufficient numerical accuracy.

In Table 6.4 approximate values of $PL3$ using different FPGA devices from the Altera Stratix II and III families are given. It is clear that due to their large on-chip memory and multiplier units, these FPGAs can easily enable real-time simulation of FE meshes in the order of several thousand nodes.

FPGA	EP2S60	EP2S180	EP3SL150	EP3SE110	EP3SE260
$PL3$	5	14	14	32	28

Table 6.4: Approximate value for $PL3$ in different FPGAs

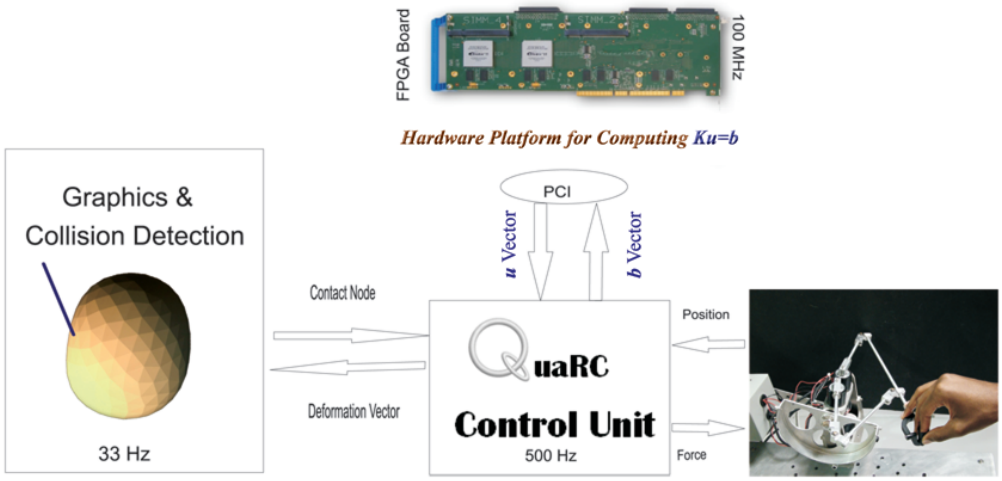


Figure 6.1: The block diagram of haptic-enabled simulator with hardware-based accelerator.

6.2 Experimental Platform

The proposed hardware-based accelerator for the CG algorithm has been implemented and integrated into an experimental platform for haptic interaction with deformable objects as shown in Fig. 6.1. The system consists of a custom-designed 3DOF haptic interface, a Quanser QPA linear current amplifier, a Quanser Q4 hardware-in-the-loop data acquisition board, a Gidel PROCStar II development board (see appendix) with two Altera Stratix II EP2S60 FPGA devices, and a 3.0 GHz Pentium R(D) with 2.0 GB RAM using a Matrox Millennium PCIe P650 graphics card. A brief description of different processes involved in the system follows.

6.2.1 Hardware-based Accelerator for the CG algorithm

The main computing engine of the system is the proposed FPGA hardware-based accelerator for the CG algorithm described earlier in the Chapter 5. At each time step, the FPGA board communicates with the PC through the PCI bus interface to receive the vector $\mathbf{b}$ for static or dynamic simulation. The matrix $\mathbf{K}$ in static (Eq. (3.16)) and also matrix $\hat{\mathbf{A}}$ (Eq. (3.27)) in dynamic simulations are assumed to be constant and are loaded to the FPGAs on-chip memory once at the beginning of the simulation process. However it should be emphasized that it is possible to partially update the on-board memory blocks filled with non-zero components of the sparse matrix. This can be done during the communication period in which the new $\mathbf{b}$ vector and the resulting node displacement vector are exchanged between the FPGA board and the host PC. As discussed in Sec. 3.7 the sparse matrix needs to be modified in real-time in order to accommodate a variable contact node. The changes to the matrix are applied directly on the FPGA devices according to the procedure outlined in 3.7. At the end of the CG iterations, the computed deformation vector $\mathbf{U}$ is returned to the PC through the PCI bus.

6.2.2 Haptic Control and Communication Process

The haptic control process is the communication hub among the haptic interface, the hardware-based accelerator, and the graphics and collision detection units, as depicted in Fig. 6.1. It runs at a rate of 500Hz under Quanser's Windows real-time extension QuaRC. During each time-step of haptic simulation, the Q4 board reads the haptic device position through the QPA current amplifier. Next, this data is sent to the collision detection unit to report the contact node number of

any collision occurs between the haptic pointer and the virtual deformable object. Based on the deformation at the contact node and the contact node number, the modified force vector $\mathbf{b}$ is calculated according to the procedure outlined in Sec. 3.7 and is transmitted to the FPGA board via the PCI bus. The deformation vector $\mathbf{U}$ is computed and sent back to the host PC. The deformation vector is employed to calculate the interaction force. At a lower rate, it is also sent to the graphics unit for display. The computed force is sent to haptic device via Q4 board. The remaining steps in the dynamic FE simulation given in Sec. 3.6, i.e. (3.24)-(3.26) are performed on the host PC. All these steps are depicted in the flow-chart diagram in Fig. 6.2 on page 73.

6.2.3 Collision Detection and Graphics

OpenGL is used for graphics rendering and collision detection. The process detects any possible collision between the haptic device and the virtual object and sends the contact node number to the control process. In turn, it receives the node displacement vector and renders the object deformation accordingly. The use of the Graphical Processing Unit on the video card for deformable object rendering and collision detection frees up CPU resources for other applications. The graphics and collision detection process is updated at a rate of 33Hz which should be sufficient considering the human visual response time and typical speed and frequency of hand motions.

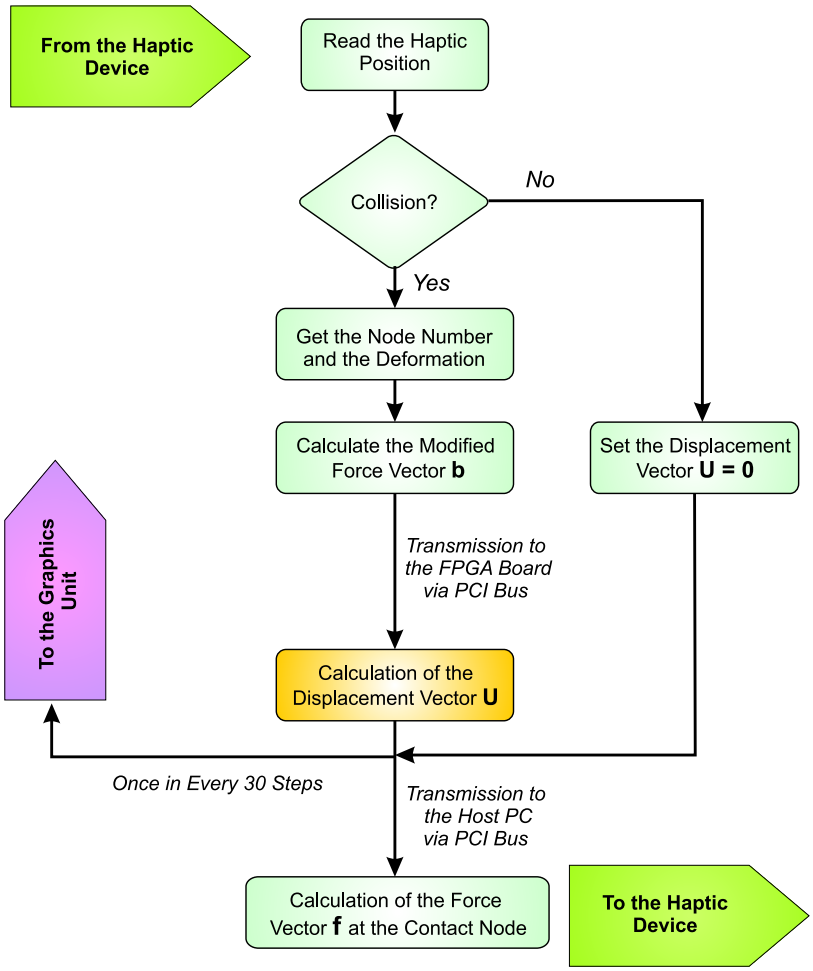


Figure 6.2: Haptic control and communication process

6.2.4 Experimental Results

Our hardware architecture which currently utilizes one Stratix II EP2S60 FPGA is capable of real-time FE simulation of deformation for a 3D mesh of 500 nodes at a rate of 500Hz (the length of vector **U** is 1500). Our preliminary haptic exploration experiments have been very encouraging. The proposed system, using linear static and dynamic elastic models, can provide users with a stable and realistic haptic and deformation feedback for objects with different mechanical properties. Fig. 6.3

shows the results of real-time FE dynamic simulation for haptic interaction with a deformable sphere. The object is constrained at several nodes in the lower left corner of the image. The user can make single point contacts with the sphere at any surface node.

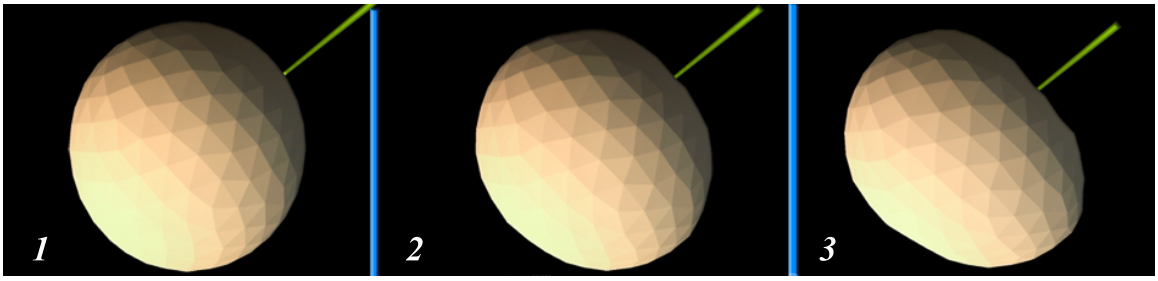


Figure 6.3: Three frames of object deformation in a dynamic simulation: (1) the user represented by small sphere is about to make contact with the large deformable sphere; (2) contact has been made; (3) the object is further pressed by the user.

Chapter 7

Conclusions and Future Work

Real-time simulation of haptic interaction with deformable objects is challenging due to a large amount of computations that must be completed within a very short period of time. In particular in FE modeling of deformation, the object is meshed and partitioned to tetrahedron elements and the partial differential equations arising from continuum mechanics based models are discretized in the spatial domain accordingly. Using static or dynamic linear elastic models results in a large linear system of equations that must be solved at each time step (namely $1 \sim 10$ msec for haptic simulation of soft tissue). The main goal of this thesis was to develop a hardware architecture to accelerate the computational kernel in such systems of equations meeting the tight real-time constrain of haptics applications. This objective was achieved through massive parallelization of the computations on an FPGA platform. Scalability of the solution and optimal usage of resources were the other main objectives in our hardware architecture design.

To solve a large and sparse linear system of equations, a fixed-point implementation of the iterative CG algorithm was proposed in Chapter 4. The numerical accuracy and reliability of the proposed fixed-point CG method were verified through numerical analysis. An FPGA-based parallel implementation of the fixed-point CG algorithm was proposed in Chapter 5. The ordered memory structure and concurrent usage of a large number of multipliers and adders provide sufficient bandwidth and computational power to meet the real-time computation requirement in haptics applications. The proposed micro-architecture employs fixed-point operations in order to maximize the parallelization of the computations. It is fairly independent of the FE mesh structure and can be easily scaled for deployment on FPGA devices with various capacities.

The hardware architecture was successfully employed for real-time haptic interaction with static and dynamic deformable objects. In our experiments, using one Altera Stratix II EP2S60 FPGA device and a 3.0 GHz Pentium R(D) with 2.0 GB RAM, for a 500-node 3D mesh with tetrahedral elements, a simulation update rate of 500 Hz was achieved. The proposed FPGA-based CG solver provides a powerful and portable computing platform that can be integrated into a desktop computer system and is capable of real-time simulation of 3D FE deformation models of several thousands nodes using commercially available FPGA devices.

Although the preliminary results presented in this thesis are very encouraging, there are still numerous possibilities for further research including:

- Extension of the work to allow multi-contact-point tool-object interaction
- Hardware-based real-time simulation of cutting and needle insertion
- Hardware-based acceleration of nonlinear FE models

- Extension of the proposed architecture to multi-FPGA configurations
- Comprehensive analysis of the trade-off involved in selecting the data word lengths, simulation accuracy, resources usage, and the achievable level of parallelism in a fixed-point implementation of the CG algorithm

Appendix A

PROCStar II Technical Specifications

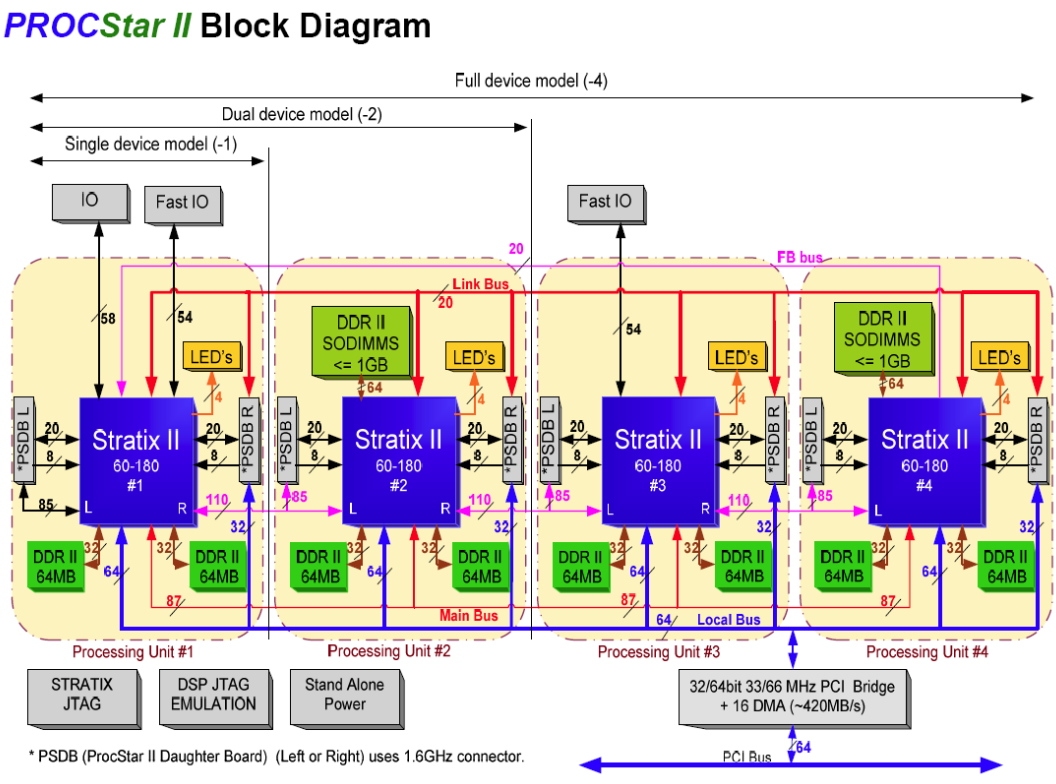


Figure A.1: PROCStar II Block Diagram

Bibliography

[1] R. T. Azuma, "A survey of augmented reality," *Presence -Cambridge Massachusetts*, vol. 6, pp. 355–385, Nov. 1997.

[2] M. Slater, A. Steed, and Y. Chrysanthou, *Computer Graphics and Virtual Environments: From Realism to Real-Time*. Addison Wesley, 2001.

[3] A. Liu, F. Tendick, K. Cleary, and C. Kaufmann, "A survey of surgical simulation: Applications, technology, and education," *Presence -Cambridge Massachusetts*, vol. 12, no. 6, pp. 599–614, 2003.

[4] D. Koller, P. Lindstrom, W. Ribarsky, L. F. Hodges, N. Faust, and G. Turner, "Virtual gis: A real-time 3d geographic information system," in *Proceedings of the 6th Conference on Visualization*, IEEE Computer Society, 1995.

[5] C. Basdogan and M. Srinivasan, "Haptic rendering in virtual environments," *K. Stanney (Ed): Virtual Environments HandBook, Lawrence Erlbaum Associates*, pp. 117–134, 2002.

[6] K. Salisbury, F. Conti, and F. Barbagli, "Haptic rendering: Introductory concepts," *IEEE Computer Graphics and Applications*, vol. 24, no. 2, pp. 24–32, 2004.

[7] S. D. Laycock and A. M. Day, "A survey of haptic rendering techniques," *Computer Graphics Forum*, vol. 26, no. 1, pp. 50–65, 2007.

[8] A. Gregory, A. Mascarenhas, S. Ehmann, M. Lin, and D. Manocha, "Six degree-of-freedom haptic display of polygonal models," in *Proceedings of Visualization*, pp. 139–146, 2000.

[9] S. P. DiMaio and S. E. Salcudean, "Needle insertion modelling for the interactive simulation of percutaneous procedures," *Medical Image Computing and Computer-Assisted Intervention*, pp. 253–260, July 2005.

[10] S. P. DiMaio and S. E. Salcudean, "Interactive simulation of needle insertion models," *IEEE Transactions on Biomedical Engineering*, vol. 52, no. 7, pp. 1167–1179, July 2005.

[11] V. Gourishankar, G. Srimathveeravalli, and T. Kesavadas, "Hapstick: A high fidelity haptic simulation for billiards," in *Second Joint EuroHaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (WHC'07)*, pp. 494–500, 2007.

[12] S. P. DiMaio and S. E. Salcudean, "Simulated interactive needle insertion," *Proceedings of the 10th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pp. 344–351, 2002.

[13] C. BRUYNS and K. MONTGOMERY, "Generalized interactions using virtual tools within the spring framework:probing, piecing, cauterizing and ablating," *J. Westwood (Ed): Medicine Meets Virtual Reality*, pp. 74–78, 2002.

[14] R. Lapeer, P. Gasson, J. Florens, S. Laycock, and V. Karri, "Introducing a novel haptic interface for the planning and simulation of open surgery," *Studies in Health Technology and Informatics*, pp. 197–199, 2004.

[15] K. Waters and D. Terzopoulos, "A physical model of facial tissue and muscle articulation," in *Proceedings of the First Conference on Visualization in Biomedical Computing*, pp. 77–82, 1990.

[16] H. Delingette, G. Subsol, S. Cotin, and J. Pignon, "A craniofacial surgery simulation testbed," *Visualization in Biomedical Computing*, vol. 2359, no. 1, pp. 607–618, 1994.

[17] E. Keeve, S. Girod, R. Kikinis, and B. Girod, "Craniofacial surgery simulation," in *Proceedings of the 4th International Conference on Visualization in Biomedical Computing*, pp. 541–546, 1996.

[18] D. Aulignac, C. Laugier, and M. Cavusoglu, "Modeling the dynamics of a human thigh for a realistic echographic simulator with force feed-back," in *Medical Image Computing and Computer Assisted Intervention*, pp. 1191–1198, 1999.

[19] M. Bro-Nielsen, "Finite element modeling in surgery simulation," *Proceedings of the IEEE*, vol. 86, pp. 490–503, March 1998.

[20] F. L. Stasa, *Applied Finite Element Analysis for Engineers*. Holt, Rinehart, and Winston, 1985.

[21] Y. Zhuang, *Real-time simulation of physically realistic global deformations*. PhD thesis, 2000. Chair-John Canny.

[22] K. J. Bathe, *Finite Element Procedures*. Prentice Hall, 1996.

[23] O. C. Zienkiewicz and R. L. Taylor, *The Finite Element Method, Vol. 1, 5th Edition*. Butterworth-Heinemann, 2000.

[24] J. Berkley, S. Weghorst, H. Gladstone, G. Raugi, D. Berg, and M. Ganter, "Banded matrix approach to finite element modelling for soft tissue simulation," *Virtual Reality*, vol. 4, pp. 203–212, September 1999.

[25] M. Bro-Nielsen and S. Cotin, "Real-time volumetric deformable models for surgery simulation using finite elements and condensation," *Computer Graphics Forum*, vol. 15, no. 3, pp. 57–66, 1996.

[26] Y. Zhuang and J. Canny, "Haptic interaction with global deformations," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2428–2433, 2000.

[27] M. Mahvash and V. Hayward, "Haptic simulation of a tool in contact with a nonlinear deformable body," in *Proceeding of the International Symp. on Surgical Simulation and Soft Tissue Deformation*, pp. 311–320, 2003.

[28] Z. A. Taylor, M. Cheng, and S. Ourselin, "Real-time nonlinear finite element analysis for surgical simulation using graphics processing units," *Medical Image Computing and Computer-Assisted Intervention*, vol. 4791, pp. 701–708, 2007.

[29] J. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," in *Technical report, School of Computer Science, Carnegie Mellon University*, 1994.

[30] L. Zhuo and V. K. Prasanna, "Sparse matrix-vector multiplication on fpgas," in *Proceedings of the 13th International Symposium on Field-Programmable Gate Arrays*, pp. 63–74, 2005.

[31] J. Jang, S. Choi, and V. K. Prasanna, "Area and time efficient implementation of matrix multiplication on fpgas," in *Proceedings of the First IEEE International Conference on Field Programmable Technology*, pp. 93–100, 2002.

[32] J. Sun, G. Peterson, and O. Storaasli, "Sparse matrix-vector multiplication design on fpgas," in *Proceedings of 15th Annual IEEE Symp. on Field-Programmable Custom Computing Machines*, pp. 349–352, 2007.

[33] U. Meier, O. Lpez, C. Monserrat, M. Juan, and M. Alcaiz, "Real-time deformable models for surgery simulation: A survey," *Computer Methods and Programs in Biomedicine*, vol. 77, no. 3, pp. 183–197, 2005.

[34] S. Coquillart and P. Jancéne, "Animated free-form deformation: An interactive animation technique," *ACM SIGGRAPH Computer Graphics*, vol. 25, no. 4, pp. 23–26, 1991.

[35] W. M. Hsu, J. F. Hughes, and H. Kaufman, "Direct manipulation of free-form deformations," in *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 177–184, 1992.

[36] M.-E. Algorri and F. Schmitt, "Deformable models for reconstructing unstructured 3d data," in *Proceedings of the First International Conference on Computer Vision, Virtual Reality and Robotics in Medicine*, pp. 420–426, 1995.

[37] T. McInerney and D. Terzopoulos, "Deformable models in medical image analysis: A survey," *Medical Image Analysis*, vol. 1, no. 2, pp. 91–108, 1996.

[38] H. Delingette, "Toward realistic soft-tissue modeling in medical simulation," *Proceedings of the IEEE*, vol. 86, no. 3, pp. 512–523, Mar 1998.

[39] P. Moore and D. Molloy, "A survey of computer-based deformable models," in *Machine Vision and Image Processing Conference,IMVIP 2007*, pp. 55–66, 2007.

[40] S. Gibson and B. Mirtich, "A survey of deformable modeling in computer graphics," in *Technical Report No. TR-97-19, Mitsubishi Electric Research Lab., Cambridge, MA*.

[41] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer, "Elastically deformable models," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 205–214, 1987.

[42] S. F. Gibson, "3d chainmail: A fast algorithm for deforming volumetric objects," in *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, pp. 149–154, 1997.

[43] A. Joukhadar and C. LAUGIER, "Dynamic simulation: Model, basic algorithms, and optimization," *J. Laumond and M. Overmars (Eds): Algorithms for Robotic Motion and Manipulation*, pp. 419–434, 1997.

[44] D. Baraff and A. Witkin, "Large steps in cloth simulation," in *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 43–54, 1998.

[45] Y. Lee, D. Terzopoulos, and K. Waters, "Constructing physics-based facial models of individuals," in *Proceedings of Graphics Interface '93*, pp. 1–8, 1993.

[46] Y. Lee, D. Terzopoulos, and K. Walters, "Realistic modeling for facial animation," in *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, pp. 55–62, 1995.

[47] D. Zerbato, S. Galvan, and P. Fiorini, "Calibration of mass spring models for organ simulations," in *Proceedings of the International Conference on Intelligent Robots and Systems*, pp. 370–375, 2007.

[48] A. V. Gelder, "Approximate simulation of elastic membranes by triangulated spring meshes," *Journal of Graphics Tools*, vol. 3, no. 2, pp. 21–41, 1998.

[49] G. Bianchi, B. Solenthaler, G. Szekely, and M. Harders, "Simultaneous topology and stiffness identification for mass-spring models based on fem reference deformations," *Medical Image Computing and Computer-Assisted Intervention MICCAI 2004*, pp. 293–301, 2004.

[50] S. De, J. Kim, and M. Srinivasan, "A meshless numerical technique for physically based real time medical simulation," in *Medicine Meets Virtual Reality*, pp. 113–118, 2001.

[51] I. Babuka, U. Banerjee, and J. Osborn, "Survey of meshless and generalized finite element methods: A unified approach," *A. Iserles (Ed): Acta Numerica*, vol. 12, pp. 1–122, 2003.

[52] S. Idelsohn, E. Onate, N. Calvo, and F. D. Pin, "Meshless finite element ideas," in *Proceedings of the Fifth World Congress on Computational Mechanics*, pp. 1–20, 2002.

[53] H. Delingette, S. Cotin, and N. Ayache, "A hybrid elastic model for real-time cutting, deformations, and force feedback for surgery training and simulation," *Visual Computer*, vol. 16, no. 8, pp. 437–452, 2000.

[54] X. Wu, M. Downes, T. Goktekin, and F. Tendick, "Adaptive nonlinear finite elements for deformable body simulation using dynamic progressive meshes," *Computer Graphics Forum*, vol. 20, no. 3, pp. 349–358, 2001.

[55] D. L. James and D. K. Pai, "Artdefo: accurate real time deformable objects," in *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 65–72, 1999.

[56] C. S. Krishnamoorthy, *Finite Element Analysis: Theory and Programming*. Tata McGraw-Hill, 1995.

[57] J. Shewchuk, "Delaunay refinement algorithms for triangular mesh generation," *Computational Geometry: Theory and Applications*.

[58] M. Bern, D. Eppstein, and J. Gilbert, "Provably good mesh generation," *Journal of Computer and System Sciences*, vol. 48, no. 3, pp. 384–409, 1994.

[59] G. Karniadakis and R. K. II, *Parallel Scientific Computing in C++ And Mpi: A Seamless Approach to Parallel Algorithms and Their Implementation*. Cambridge University Press, 2003.

[60] Y. Saad, *Iterative Methods for Sparse Linear Systems, 2nd Edition*. SIAM, 2003.

[61] K. H. Huebner, D. L. Dewhirst, D. E. Smith, and T. G. Byrom, *The Finite Element Method for Engineers, 4th Edition*. Wiley, 2001.

[62] P. O. Fjllstrm, "Algorithms for graph partitioning: A survey," *Computer and Information Science*, vol. 3, 1998.

[63] METIS - Family of Multilevel Partitioning Algorithms: http://glaros.dtc.umn.edu/gkhome/views/metis.

[64] G. Goumas, K. Kourtis, N. Anastopoulos, V. Karakasis, and N. Koziris, "Understanding the performance of sparse matrix-vector multiplication," in *Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008)*, pp. 283–292, 2008.