STUDIES ON THE VIABILITY OF THE BOUNDARY ELEMENT METHOD FOR THE REAL-TIME SIMULATION OF BIOLOGICAL ORGANS

A Thesis

Submitted for the degree of

DOCTOR OF PHILOSOPHY

IN THE FACULTY OF ENGINEERING

BY

Kirana Kumara P

Centre for Product Design and Manufacturing

Indian Institute of Science

BANGALORE-560012

AUGUST 2016

Acknowledgements

The contents of this page are not made available for privacy reasons.

Abstract

Realistic and real-time computational simulation of biological organs (e.g., human kidneys, human liver) is a necessity when one tries to build a quality surgical simulator that can simulate surgical procedures involving these organs. Currently deformable models, spring-mass models, or finite element models are widely used to achieve the realistic simulations and/or the real-time performance. It is widely agreed that continuum mechanics based numerical techniques are preferred over deformable models or spring-mass models, but those techniques are computationally expensive and hence the higher accuracy offered by those numerical techniques come at the expense of speed. Hence there is a need to study the speed of different numerical techniques, while keeping an eye on the accuracy offered by those numerical techniques. Such studies are available for the Finite Element Method (FEM) but rarely available for the Boundary Element Method (BEM). Hence the present work aims to conduct a study on the viability of BEM for the real-time simulation of biological organs, and the present study is justified by the fact that BEM is considered to be inherently efficient when compared to mesh based techniques like FEM. A significant portion of literature on the real-time simulation of biological organs suggests the use of BEM to achieve better simulations.

When one talks about the simulation of biological organs, one needs to have the geometry of a biological organ in hand. Geometry of biological organs of interest is not readily available many a times, and hence there is a need to extract the three dimensional (3D) geometry of biological organs from a stack of two dimensional (2D) scanned images. Software packages that can readily reconstruct 3D geometry of biological organs from 2D images are expensive. Hence, a novel procedure that requires only a few free software packages to obtain the geometry of biological organs from 2D image sequences is presented. The geometry of a pig liver is extracted from CT scan images for illustration purpose. Next, the three dimensional geometry of human kidney (left and right kidneys of male, and left and right kidneys of female) is obtained from the Visible Human Dataset (VHD). The novel procedure presented in this work can be used to obtain patient specific organ geometry from patient specific images, without requiring any of the many commercial software packages that can readily do the job.

To carry out studies on the speed and accuracy of BEM, a source code for BEM is needed. Since the BEM code for 3D elasticity is not readily available, a BEM code that can solve 3D linear elastostatic problems without accounting for body forces is developed from scratch. The code comes in three varieties: a MATLAB version, a Fortran version (sequential version), and a Fortran version (parallelized version). This is the first free and open source BEM code for 3D elasticity. The developed code is used to carry out studies on the viability of BEM for the real-time simulation of biological organs, and a few representative problems involving kidneys and liver are found to give accurate solutions. The present work demonstrates that it is possible to simulate linear elastostatic behaviour in real-time using BEM without resorting to any type of precomputations, on a computer cluster by fully parallelizing the simulations and by performing simulations on different number of processors and for different block sizes. Since it is possible to get a complete solution in real-time, there is no need to separately prove that every type of cutting, suturing etc. can be simulated in real-time.

Future work could involve incorporating nonlinearities into the simulations. Finally, a BEM based simulator may be built, after taking into account details like rendering.

Table of Contents

List of Figures

List of Tables

Notations and Abbreviations

1D	One dimensional
2D	Two dimensional
3D	Three dimensional
BE	Boundary Element
BEA	Boundary Element Analysis
BEM	Boundary Element Method
CPU	Central Processing Unit
CT	Computed Tomography
FE	Finite Element
FEA	Finite Element Analysis
FEM	Finite Element Method
GPU	Graphics Processing Unit
mm	millimeter
MRI	Magnetic Resonance Imaging
N	Newton
N/mm^2	Newton per millimeter squared
VHD	Visible Human Dataset
VHP	Visible Human Project

Chapter 1: Introduction

1.1 Motivation

A surgical simulator is very useful for training surgeons in difficult surgical procedures like laparoscopic surgery, brain surgery etc. Real-time simulation of biological organs is a necessity while building realistic surgical simulators (real-time performance greatly enhances realism). Currently, real-time simulation in most cases is achieved using deformable models, spring-mass models, or models based on the Finite Element Method (FEM). There is a large amount of literature on the use of each of these approaches for the real-time simulation of biological tissues and organs, including soft tissues (and organs that contain soft tissues, like liver). For example, [U. Meier, et al., 2005] presents a classification of the different deformable models available in literature, makes a comparison of the different deformable models, performs an evaluation of the state of the art, and discusses the future of deformable models. The reference [Herve Delingette, 1998] surveys existing models of deformation in medical simulation and analyzes the impediments to combining computer-graphics representations with biomechanical models. In this reference, different geometric representations of deformable tissue are compared in relation to the tasks of real-time deformation, tissue cutting, and force-feedback interaction. The reference [Herve Delingette and Nicholas Ayache, n.d.] gives some interesting facts about surgical simulators in the beginning. For example, the reference notes that the concept of surgery simulation, in the beginning, was in large part advocated by the American Department of Defense, as a key part of their vision of the future of emergency medicine; the reference notes that the development of commercial simulators has proved that there is a demand for products that help to optimize the learning curve of surgeons; the reference mentions that the emergence of medical robotics and more precisely of minimally invasive surgery robots, has reinforced the need for simulating surgical procedures, since these robots require a very specific hand-eye coordination; the reference also mentions that the modeling of living tissue, and their ability to deform under the contact of an instrument is one of the important aspect of simulators that should be improved. Next, the reference presents different algorithms for modeling soft tissue deformation in the context of surgery simulation. The reference [A. C. M. Dumay and G. J. Jense, 1995] discusses on the topic of endoscopic surgery simulation in a virtual environment. The paper tries to answer the question: “Can virtual environment technology assist surgeons in training and maintaining endoscopic surgery skills?” The paper opines that virtual environment technology has great potential for surgical training purposes. The reference [Jay T. Bishoff and Louis R. Kavoussi, n.d.] mentions that laparoscopic surgery is often a substitute for traditional open surgery because choosing laparoscopic surgery over an open surgery reduces trauma and shortens the recovery time for the patient, while [Issenberg SB, et al., 1999] mentions that since laparoscopic surgery needs highly skilled surgeons, it is preferable to use a surgical simulator for training and evaluating surgeons.

Sources in the literature (e.g., [U. Meier, et al., 2005]) widely agree on the fact that continuum mechanics based models are desirable if one is interested to achieve highly accurate simulations of biological organs. FEM is the most widely used numerical technique employed for the continuum mechanics based simulation of biological organs. One can note that *real-time* simulation of three dimensional nonlinear solids is a necessity for the continuum mechanics based *real-time* computational simulation of biological organs. One can also note that real-time graphics needs about *30* computations per second whereas real-time haptics needs about *1000* computations per second [SensAble technologies, 2005].

However, no one has been able to achieve real-time performance with realistic material behaviour using just nonlinear FEM in a straight forward way [Firat Dogan and M. Serdar Celebi, 2011]. Many have used just linear FEM with highly simplified material behaviour to achieve real-time performance, while some others have not bothered about real-time performance (they worry about the accuracy of the results only). Some have applied specialized techniques like static condensation [Olek C Zienkiewicz, et al., 2013] to solve only a specialized problem in hand. Many references have used specialized techniques like precomputations and static condensation even while using a linear elastostatic material behaviour. Still, research groups are hopeful that, with the ready availability of high performance computing (clusters, supercomputers, GPUs), it could be possible to achieve real-time performance, for a reasonably accurate material behaviour (e.g., [Salisbury, n.d.; Stephane Bordas, 2008]). The current status of these projects is not known. Hence how far these projects have been able to achieve the desired goals is not known. Just like these researchers are trying to use/evaluate FEM for surgical simulations, one can use/evaluate the Boundary Element Method (BEM) for the simulations; BEM too is a continuum mechanics based numerical technique, just like FEM.

The Boundary Element Method (BEM) is considered to be inherently efficient and hence there is a need to study the performance of the technique in the context of the real-time simulation of biological organs; but studies in this direction are rare, although review papers that deal with the topic of the real-time simulation of biological organs often mention that BEM could be better for the real-time simulations when compared to FEM.

The work presented in this thesis is motivated by the need to find answers to the following questions. The goal is to answer questions such as a) Is it possible to achieve realistic and real-time simulation of biological organs using the Boundary Element Method (BEM)?, b) What is the performance (speed and accuracy) of BEM in the context of the realistic and real-time simulation of biological organs?, c) Is BEM viable (and desirable) for the real-time simulation of biological organs?

The present work aims towards evaluating the suitability of BEM for the real-time and realistic simulation of biological organs by measuring speed and accuracy. A systematic study of this kind for BEM is not available in the literature. Speed is rarely mentioned in any publication (BEM or otherwise). Although speed depends on hardware, software, and expertise of the programmer too, reporting speed can at least give some indication on the usefulness of a computational technique for real-time applications. Although some researchers work on using continuum mechanics based models for the realistic and real-time simulation of biological organs, publications or theses that provide elaborate and quantitative discussion on the performance (speed, accuracy) of different numerical techniques (especially BEM) are rare (if not nonexistent).

1.2 Literature Review

There is a large amount of literature on the use of the Finite Element Method (FEM) for the real-time simulations; this literature is not discussed here. Although rare when compared to FEM, other continuum mechanics based numerical techniques like meshfree methods and the Boundary Element Method (BEM) have been tried out in the literature for the real-time simulations. For example, author's work [Kirana Kumara P and Ashitava Ghosal, 2012] uses the Finite Point Method (FPM) and precomputations to obtain real-time performance, whereas [Suvranu De, et al., 2005] uses the Method of Finite Spheres (MFS); both of these numerical techniques are meshfree methods. The reference [Suvranu De, et al., 2005] uses the point collocation-based method of finite spheres (PCMFS) for real time simulation of soft tissues during minimally invasive surgery (MIS). The reference also discusses the integration of the methodologies into a practical surgical simulation system. The reference [S. De, et al., 2003] reports the new developments and new applications of the method of finite spheres. In particular, it uses the method of finite spheres in a surgical simulator; the commercial finite element program 'ADINA' is also used in the work.

The review paper [U. Meier, et al., 2005] identifies BEM as a candidate which could be better than FEM when it comes to the realistic and real-time simulation of biological organs for surgical simulations. References [Ullrich Meier, et al., 2001; C. Monserrat, et al., 2001; P. Wang, et al., 2007] use BEM for simulations. Reference [Ullrich Meier, et al., 2001] uses linear elastostatic boundary elements together with Woodbury formula [Max A. Woodbury, 1950]; Woodbury formula is used to modify the characteristic matrix after the geometry is changed because of cutting. This reference simulates incisions on an organ in real-time. Reference [C. Monserrat, et al., 2001] uses linear elastostatic boundary elements together with precomputations to achieve real-time simulations. It also suggests parallel computing to obtain real-time performance but does not implement it. The reference tells that establishing deformable models with BEM is a reliable method to model objects in virtual reality environments for surgical simulations. In addition, the elasticity parameters are obtained experimentally through the use of a pig's liver. Reference [P. Wang, et al., 2007] precomputes linear elastostatic boundary element solutions to obtain real-time performance. To simulate cutting, [P. Wang, et al., 2007] uses interpolated pre-solutions which may not provide accurate solutions every time because cutting changes geometry and hence the solutions obtained from interpolated pre-solutions could be far from accurate at times. Reference [P. Wang, et al., 2007] also suggests parallel computing to obtain real-time performance but does not implement it. A few more references that use BEM for simulations are mentioned next.

Reference [Foster, et al., 2013] achieves real-time performance by using linear elastostatics and updating only part of the boundary element mesh where there is a change in the mesh, and updating the boundary element system of equations for that part of the geometry only. This method may not be suitable every time especially when there is considerable difference between the original mesh and the modified mesh. Reference [Robert G Aykroyd, et al., n.d.] deals with real-time boundary elements, but it is neither related to 3D linear elastostatics nor related to the simulation of biological organs. Reference [P Wang, et al., 2009] also uses linear elastostatics together with precomputations and interpolating precomputed solutions to achieve real-time performance. In that work, upon cutting, boundary element mesh is updated only where there is a change in the mesh, and the boundary element system of equations is updated for that part of geometry only. A look-up table is created in the precomputation step by applying unit displacement in three mutually perpendicular directions, on all the nodes in turn, and storing the solutions. Since the principle of superposition holds good for linear elastostatic problems, the solutions stored in the look-up table may be suitably superimposed to obtain the solutions during real-time simulations. In the concerned simulation, since a tumour is located completely within a soft tissue, [P Wang, et al., 2009] is an example where the boundary element has been used to solve a problem where a homogeneous and isotropic domain is enclosed within another homogeneous and isotropic region (i.e., a problem dealing with multiple regions or two regions). Mesh that represents the tumour and the mesh that represents the soft tissue together form nested surfaces here. Reference [P Wang, et al., 2009] also serves as an example where biological organs (soft tissue in particular) are modeled with the linear elastostatic constitutive law. Reference [Bo Zhu and Lixu Gu, 2012] deals with a hybrid deformable model and uses BEM together with a mass-spring model to simulate biological organs.

One can note that BEM is an established numerical technique that has already been used to solve a wide variety of problems including nonlinear problems. In the literature, one can find the application of BEM to problems which are more or less similar to the simulation of biological organs (e.g., [Martin Bayliss, 2003]). One can also note that those applications do not bother about real-time performance, and some of those bother about two dimensional (2D) applications only.

Studies on the applicability of FEM (not BEM) for the real-time simulation of biological organs are already available [e.g., Alex Rhomberg, 2001]. Such studies are not available for BEM. Since BEM has been identified in the literature as a good candidate for the simulation of biological organs, further studies on the usefulness of BEM for the simulation of biological organs is a necessity, and the present work is a small step towards fulfilling that necessity.

One can observe that reference [M. Kreienmeyer and E. Stein, 1995] deals with the parallel implementation of the boundary element method for linear elastic problems on a MIMD parallel computer, but it deals with two dimensional (2D) problems only. In addition, their work does not deal with real-time simulations, and it does not worry about the applicability of BEM to the simulation of biological organs.

One can note that nobody has tried to perform nonlinear simulations in real-time using boundary elements so far, may be because on the one hand codes for nonlinear boundary element analysis are not readily available and on the other hand 'common sense' may be telling that it would not be possible to get real-time performance with nonlinear boundary elements because of inherent limitations and delays that may exist in computer systems. It is important to note that when BEM is to be used to solve nonlinear problems, not only the boundary of the solution domain but also the volume of the solution domain needs to be discretized which means that one of the main advantages of using BEM over FEM is lost as soon as one tries to solve a nonlinear problem using BEM, and this may also be a reason why there has been no attempt to achieve the real-time performance while using nonlinear boundary elements.

1.3 Problem Definition

By carefully looking at the literature that deals with the application of BEM to the real-time simulation, one can observe that all of the works achieve the real-time performance either by following one of the following approaches or by following a combination of the following approaches: (i) Calculate the characteristic matrix for the system offline, and also take the inverse of the characteristic matrix offline (useful only in case of 3D linear elastostatics and also where the characteristic matrix does not change) (ii) Compiling a look-up table during the precomputation step by applying unit displacement in three mutually perpendicular directions, on all the nodes in turn, and storing the solutions; since the principle of superposition holds good for linear elastostatic problems, the solutions stored in the look-up table may be suitably superimposed to obtain the solutions during real-time simulations by interpolating the solutions from the look-up table (iii) Using Woodbury formula [Max A. Woodbury, 1950] to modify the characteristic matrix after the geometry is changed (because of cutting, say) (iv) Using a fine mesh only near the points where there is an interaction between the biological organs and the surgical tools (and also using a fine mesh where there is a contact, e.g., contact with the surrounding organs), and using a coarse mesh for the rest of the regions (v) Whenever there is a change in the geometry, boundary element mesh is updated only where there is a change in the mesh, and the boundary element system of equations are updated for that part of geometry only, and (vi) Using hybrid models, e.g., using BEM together with a spring-mass model.

One can note that the approaches (i) to (iii) explicitly require precomputations. As mentioned above, the approach (iv) requires a fine mesh only near the points where there is an interaction between a biological organ and a surgical tool and where there is contact, e.g., contact with the surrounding organs. Since one cannot know beforehand which part of the biological organ would be in contact with the surgical tool and which part of the biological organ would be in contact with the surrounding organs (since the boundary conditions can change during the simulations), the approach is not of general use. While following the approach (v), whenever there is a change in the geometry, boundary element mesh is updated only where there is a change in the mesh and the boundary element system of equations are updated for that part of geometry only. Since it is not possible to know beforehand where there is going be be a change in the geometry during the simulations, this approach would not be of general use either. The approach (vi) uses not just BEM but a hybrid model where BEM together with a spring-mass model is employed. In this case, one also needs to calculate the parameters for the spring-mass model beforehand. Hence the approaches (iv) to (vi), although may not require precomputations explicitly, may be thought to require some form/type/kind of precomputations. Since the approaches (i) to (iii) require precomputations explicitly, one can say that all of the references in the literature that make use of BEM for the real-time simulation of biological organs use some form/type/kind of precomputations.

One can see that solving a BEM problem in real-time without following any of the approaches (i) to (vi) above definitely has advantages. For example, simulating cutting reduces to solving the BEM problem for the changed geometry. Simulating prodding, suturing, dealing with multiple regions etc. do not require any special techniques or any special approaches. They always reduce to just solving a linear elastostatic problem with different loads and boundary conditions and for different meshes. Since the time required to solve a linear elastostatic problem can be estimated beforehand for a given number of total degrees of freedom and a given element type, one can always be sure that the computations are performed within the allowed time limit (i.e., in real-time). One can also note that many of the sources in literature recommend parallelization, but they mention it as 'future work'. This author has not come across any source in the literature that reports the parallelization without following any of the approaches (i) to (vi) above of a BEM solution with the intention of obtaining real-time performance, although this approach has been recommended in many places.

Hence this work aims to achieve the real-time performance by parallelizing the BEM but without following any of the approaches (i) to (vi) above. In this work, whole of the BEM model is parallelized, not just a part of the BEM model is parallelized. The parallelization is not limited to the part of the BEM model corresponding to the locations close to the points where loads and boundary conditions are applied. Further, whole of the BEM code that is used for the simulations is parallelized, not just some parts of the code are parallelized, e.g., the goal is not to parallelize just the part of the code that solves the system of simultaneous equations, but the part of the code that calculates the elements of the characteristic matrix needs to be parallelized also. As already indicated in the last paragraph, one of the natural advantages of the present approach is that there is no need to calculate the modified 'stiffness' matrix from the original 'stiffness' matrix when there is a change in the geometry because of a surgical cut. Hence one is guaranteed to get the real-time performance irrespective of whether there is a change in the geometry (because of cutting, say) and irrespective of whether the change in the geometry is small or large, once it is found that the simulation is real-time for a given problem (or a given problem size). It is important to note that this is not the case if one follows any of the approaches that are followed in any of the references in the literature that use BEM to achieve the real-time simulation of biological organs.

In the present work, simulations are carried out on a desktop computer equipped with a Graphics Processing Unit (GPU), and on a computer cluster (up to *256* processors). This work is a systematic study which demonstrates the applicability of BEM as applied to the real-time simulation of biological organs.

It is also worthwhile to mention that if it is found that only real-time graphics (about *30* computations per second) can be achieved but not real-time haptics (about *1000* computations per second), the simulation is useful still, since it can simulate surgical procedures like laparoscopic surgery where surgeons get little force feedback. If it is found that real-time haptics is also achieved, even open surgeries can be simulated effectively.

Of course, simulations may not need to be strictly real-time sometimes (i.e., a 'hard' real-time system may not be needed, but 'firm' or 'soft' real-time systems may also be suitable). For example, [P. Wang, et al., 2007] mentions that a neurosurgeon takes about *0.25* second to perform each incremental stage of surgical cutting in practice; hence it may be all right for the simulations to take *0.25* second to complete in this case. The reference [Valerie E. Taylor, et al., 1996] mentions that the allowable lag time is *0.1* seconds for inexperienced users and it is *1* second for experienced users; hence the simulations can take *0.1* or *1* second in this case. References [Sonya Allin, et al., 2002; H. Pongrac, et al., 2006; Hong Z. Tan, et al., 1994] mention that the just noticeable difference of the human sensory system for force perception at human fingertips is about *10%*. The reference [Ottensmeyer M and Salisbury K, 2001] mentions that humans cannot sense very small errors in real-time responses of solids. However, these references cannot serve as excuses for not obtaining real-time performance, since there would not have been a need for the various attempts in the literature to obtain a strictly real-time performance if one could always justify that a real-time performance is not a necessity when it comes to the realistic simulation of biological organs.

All the studies considered in the present work assume linear elastostatic behaviour first. The idea is to go for nonlinear behaviour only if it is possible to achieve the real-time performance (at least real-time graphics) with the linear elastostatic material model. Employing just the 3D linear elastostatic model (since it is continuum mechanics based) can provide better realism when compared to a model which is not physically based. The realism that may be lost because of the simplicity of the 3D linear elastostatic model may (at least partially) be compensated by the real-time performance that may be achievable by using the model. One can also note that many previous works reported in the literature are happy with the 3D linear elastostatics.

In the present work, biological organs are assumed to be isotropic and homogeneous. One can note that many biological organs (including soft biological organs like liver and kidney) are considered homogeneous and isotropic by researchers [Ullrich Meier, et al., 2001; C. Monserrat, et al., 2001; P. Wang, et al., 2007; Foster, et al., 2013], including many who are into realistic simulations of biological organs [Ullrich Meier, et al., 2001; C. Monserrat, et al., 2001; P. Wang, et al., 2007].

Rendering and the time needed for rendering is not considered in the present work. One can note that one can ignore the time needed for rendering since the time is likely to be small; one can also note that [Taylor V.E., et al., 1996] informs that out of simulation, tracking, rendering, network, and synchronization components of lag time, simulation time is the main cause for the lag when a simulation cannot be accomplished in real-time.

Approaches like “running simulation in hardware”, embedded systems, building and using custom hardware, custom-built processors, digital signal processors (DSPs), neurocomputers, analogue computers, real-time operating systems, custom operating systems with in-built simulation software etc. have not been tried out in the present thesis. New techniques which are not yet mature or practical like quantum computing, DNA computing etc. are also not considered in the present work.

This study uses only the standard comtemporary hardware (e.g., a desktop computer, a GPU, a computer cluster) together with the standard contemporary software (e.g., MATLAB, MATLAB Parallel Computing Toolbox, Fortran, MPI, BLACS, LAPACK, ScaLAPACK). No custom methods and communication protocols are used.

When one talks about the simulation of biological organs, one should have the geometry of some biological organ in hand. It is found that the geometry of biological organs of interest is not readily available many a times, and hence there is a need to extract the 3D geometry of biological organs from a stack of 2D scanned images in those cases. It is also found that the software packages that can readily reconstruct 3D geometry of biological organs from 2D images are expensive. Hence the need to obtain the geometry of biological organs from 2D scanned images by making use of only free and open source software packages is felt. Therefore, the present work also gives a novel procedure to reconstruct the 3D geometry of biological organs from 2D scanned images first, while making use of free software packages only. It is noteworthy to mention here that 2D scanned images for the entire representative human body is available for download without any cost from the dataset called Visible Human Dataset (VHD). Coming to the related literature, references [Dong Sun Shin, et al., 2011; Amy Elizabeth Kerdok, 2006; Li Lou, et al., 2009; Chen G, et al., 2010] have used images from sources other than VHD and have used commercial software packages to perform 3D reconstruction of biological organs, while the reference [Aimee Sergovich, et al., 2010] uses VHD together with commercial software packages. There are also authors who have used free software packages to extract geometry of biological organs [Hong-jian Gao, 2007].

In summary, this thesis tried to test the hypothesis: “Is the boundary element method viable as far as the real-time simulation of biological organs is concerned?” During the course of trying to answer the above question, a need to obtain the geometry of biological organs without making use of any paid software was felt, and hence some attempt was made in that direction too.

1.4 Contributions from the Present Work

The following are the main contributions of this work:

1. A novel procedure to obtain the geometry of biological organs, that uses only free and open source software packages, is presented. A pig liver is reconstructed from CT scan images that were available in the internet, by using the novel procedure. Human kidney (both left and right kidney, for both male and female) is reconstructed by using images from the Visible Human Dataset, by following the novel procedure. The novel procedure can also be employed to obtain patient specific geometry of biological

organs from patient specific CT scan images. The potential of a Linux flavor called CAELinux is recognized in the context of 3D reconstructions.

2. A BEM code that can solve 3D linear elastostatic problems is developed from scratch. The code is the first free and open source BEM code for 3D elasticity, in any of the programming languages. The code is available in three versions: MATLAB version, Fortran version (sequential version), Fortran version (parallelized version). The code can be of use to others who need an open access BEM source code for 3D linear elasticity especially because the present author could not find any suitable BEM source code for 3D linear elastostatics, either as free and open source software or as paid software.

3. This work is the first one to demonstrate that it is possible to simulate linear elastostatic behaviour in real-time using BEM, without resorting to any type of precomputations; all the earlier works use some or the other forms/types/kinds of precomputations to achieve the real-time performance, although some of the earlier works recommend the present approach as their future work. Since *all* computations can be performed in real-time in the present work, there is no need to separately prove that any type of cutting, suturing etc. can be simulated in real-time (this is *automatically proved* since it is possible to get a *complete* solution in real-time in the present approach).

4. This work is the first one to use a GPU for simulating biological organs while using BEM. In the present work, up to 14 times speed up has been achieved using a Graphics Processing Unit (GPU).

1.5 Overview of the Thesis

The present chapter (Chapter 1) has explained the motivation for undertaking the present research, carried out the literature review, defined the research problem, and listed out the contributions from the present work. Chapter 2 presents a novel procedure to extract three dimensional geometry of biological organs from two dimensional CT scan images. Chapter 3 first mentions about the development of the Boundary Element codes that are developed from scratch by the present author, and a study on the real-time simulation of biological organs using the Boundary Element

Method is presented afterwards. Chapter 4 is about conclusions, and mentioning about the scope for future work.

Chapter 2: Obtaining the Geometry of Biological Organs of Interest

To carry out any simulation on a biological organ (e.g., surgery planning, surgery simulation), the geometry of the biological organ is needed first. For example, a three dimensional digital model of the actual (or representative) human kidney is needed for a surgical simulator that is capable of simulating a laparoscopic surgery involving kidney. However, anatomical dimensions and geometry of body organs differ from patient to patient due to various factors such as age, body size, and weight and due to the presence of pathologies such as cysts and cancer. Using scanning procedures like Computed Tomography (CT), one can get patient specific 2D image sequences corresponding to different organs, and one can reconstruct 3D models of the patient specific biological organs from the corresponding 2D image sequences. In the same way, if one is happy with just a representative biological organ (not the patient specific biological organ), one can reconstruct the geometry by making use of the already available 2D images corresponding to the representative biological organ.

Currently, mainly two approaches are being practiced to obtain the geometry of a biological organ. The first approach is to buy a readily available 3D model of a representative biological organ from an online store (e.g., [TurboSquid, n.d.]). The second approach is to use commercial software packages (such as [3D-DOCTOR, n.d.; Mimics, n.d.; Amira, n.d.]) to reconstruct a 3D geometry of biological organs from a two dimensional (2D) image sequence. One can see that both of these approaches cost (sometimes significant amount of) money.

Hence in this chapter, a procedure is presented for the reconstruction of biological organs from image sequences obtained through CT-scan. Although commercial software packages which can accomplish this task are readily available, *the procedure presented here needs only free and open source software packages. The user is able to control the detail or the level of complexity of the solid constructed.* The free software packages used are: ImageJ, ITK-SNAP, and MeshLab. The procedure is presented through the illustration of the reconstruction of a pig liver from the scan data available in the literature. Next, a three dimensional surface model of human kidney is sucessfully reconstructed by making use of images from the Visible Human

Data Set (VHD); the images from the Visible Human Data Set do not cost anything. The 3D biological organs obtained this way can be used in the finite element analysis and this has been demonstrated by carrying out FE analyses on the reconstructed liver; the commercial software packages Rhinoceros [Rhinoceros, n.d.] and ANSYS [ANSYS, n.d.] are used for this purpose. Finally, it is noted that a Linux flavor called CAELinux has all the necessary software packages (that can do 3D reconstruction of biological organs) in-built; hence using this operating system can avoid the necessity of having to install all the necessary software one-by-one.

2.1 The 3D Reconstruction of a Pig Liver

2.1.1 Software Packages used

The software packages that are needed to carry out the procedure are: ImageJ, ITK-SNAP, MeshLab. Very concise information on these software packages is given below and the features that are most useful for the present work are presented.

ImageJ

ImageJ [ImageJ, n.d.; Rasband W.S., n.d.; Abramoff M.D., et al., 2004] is a free and open source software package used for image processing. The software package is in the public domain. The software is written in Java, and hence it can run on any computer with Java installed.

Present work uses ImageJ 1.42q (together with Java 1.6.0). Its role is to import an image stack from the file, getting image information, selecting the image portions with a rectangle and processing the selected portion only, removing selected portion of the image, subtracting a particular intensity value from each pixel in the image, multiplying each pixel intensity value by a particular number, applying the image processing procedures to a single image or to the whole image stack, reversing the order of images in an image stack, concatenating image stacks, saving the processed image stack in different file formats etc.

ITK-SNAP

ITK-SNAP [ITK-SNAP, n.d.; Paul A. Yushkevich, et al., 2006] is a free and open source software package that may be used to carry out the segmentation (According to a popular definition, “Image segmentation is the process of assigning a label to every pixel in an image in such a way that pixels with the same label share certain characteristics”) for structures in 3D images. The software supports image navigation, semi-automatic segmentation, and manual segmentation, and it can also export a segmented image sequence (or segmented volume) as a 3D surface mesh. ITK-SNAP is a free and open source software package that is made available under the General Public License [GNU, n.d.]. ITK-SNAP 2.0.0 is used in the present work for 3D automatic (also called semi-automatic) segmentation.

Selecting the region of interest is the first step in the automatic segmentation. The same region of interest applies to all the images in the image stack. The next step is ‘snake initialization’. The term ‘snake’ here refers to a closed curve (or a collection of closed curves) in 2D, or a closed surface (or a collection of closed surfaces) in 3D. ‘Snake’ is initialized as a circular bubble. In fact, the ‘snake’ is intended to be an approximation of the final segmented volume. In the next step, the ‘snake’ evolves, or the initial ‘snake’ represented by the spherical bubble changes its shape towards the shape of the final segmented volume that represents the surface of the biological organ to be extracted from the image stack. The final step in using ITK-SNAP is to export the (3D) segmented volume as a (3D) surface.

MeshLab

MeshLab [MeshLab, n.d.] is a free and open source software package that can process unstructured 3D triangular meshes. The software can edit, clean, heal, inspect, render and convert models made up of 3D triangular meshes (meshes describing the surfaces of the 3D models). The software package is licensed under the GNU General Public License. MeshLab v1.2.2 has been used in the present work. Some of the numerous useful features available in the software are explained next.

Fill Hole is used to check whether the 3D surface has any holes (the mesh to be edited should not have any holes).

Quadric Edge Collapse Decimation filter is used to convert the geometry described by very large number of triangle faces that is usually the case, to a geometry that is a very good approximation to the original one but described with lesser number of faces.

Triangle Quality filter can display the quality measures for each of the triangular faces of the 3D mesh.

MLS Projection filter is used to improve the quality of the triangular faces that describe the 3D mesh. The filter achieves the goal by projecting the whole mesh onto the MLS surface defined by it, as explained in the reference [Oztireli, et al., 2009].

Laplacian Smooth filter is another filter that may be used for improving the triangle quality. The smoothing is achieved here by calculating the average position for each vertex with its nearest vertex.

Taubin Smooth filter is also found to be very useful for improving the triangle quality. For each iteration, it makes two steps of smoothing, forth and back.

2.1.2 A Note about the Operating Systems

A point to be noted is that it is better to run image processing software packages like ImageJ, ITK-SNAP and MeshLab under Linux because of its stability while running memory intensive tasks. Both Linux and Windows versions of all the three free and open source software packages explained above are available for download, and the present author has tried running these software packages under both Linux (Fedora 9, 64 bit) and Windows XP Professional SP2 (32 bit). This author has found that, although installation might be somewhat tedious sometimes because of the need for manual compilation of the source code and/or the need to resolve and install for dependencies, it is worth spending some time installing these software packages under Linux, since it was found to make a robust environment for processing large image sequences. However, because of ease of installation, Windows versions can be used and they work fine as long as the problem size is not too large or the processing task does not require too much memory. As far as the present work is concerned, both ImageJ and MeshLab could be used both under Linux and Windows without any observable difference in the performance. The tasks requiring ITK-SNAP could be completed only under Linux.

The hardware configuration of the computer used for this work is: AMD Athlon 64 X2 Dual Core Processor 5200+, 2.61 GHz, 8 GB RAM. However, although Linux could use the entire available RAM, Windows could detect only 2.75 GB of RAM.

This might be the reason why ITK-SNAP had trouble while executing a memory intensive task like semi-automatic segmentation under Windows.

2.1.3 General Procedure for Extracting 3D Organs from 2D Image Slices

The following three steps make up the procedure to obtain 3D organs from 2D image sequences. These steps make use of the software packages (all free) explained in the previous subsection.

1. Open ImageJ, import the image file (e.g., *.tif), process the image stack with the goal of brightening the pixels which belong to the organ of interest but with the goal of making the intensity of the pixels which do not belong to the organ of interest zero. This makes the next step (segmentation) much easier. Many a times, to reduce size, image files give only the half of the information (the other half of the organ may be considered to be symmetric to the first half). In that case, one can reverse the order of images in the image stack and then concatenate this image stack with the original image stack to get the full image stack. One should save the processed image stack in a file format that can be read by ITK-SNAP (e.g., *.raw).

2. Open ITK-SNAP and open the file saved in the previous step. If the image file format does not include the image header information (like the total number of pixels in each direction, absolute distance between pixels in both the directions, total number of such images in the image stack considered, distance between the consecutive images in the stack, voxel representation scheme etc.), ITK-SNAP requests the user to enter this information. In this case, the information has to be obtained from the source that supplied the image file. Then, ITK-SNAP displays the image stack. Next, one has to complete segmentation (manual or automatic). Then, the segmented volume has to be exported as a 3D surface mesh using the menu item *Save as Mesh…* in a file format supported by MeshLab (e.g., *.stl).

3. Open MeshLab and open the file saved in the previous step. Next, *Quadric Edge Collapse Decimation* filter is used to reduce the number of triangle faces to the target number of faces. *Triangle Quality* is used to colour code the triangles to identify ill shaped triangles and the filters *MLS projection*,

Laplacian Smooth and *Taubin Smooth* are used to make those triangles well shaped. Finally, the processed mesh is saved. This mesh represents the 3D organ of interest.

2.1.4 An Illustration of the 3D Reconstruction of a Pig Liver

Obtaining the Image File and its Details

The image file is obtained from the website of the Harvard Biorobotics Laboratory [Biorobotics, n.d.]. The image file is a zipped .tif image stack which contains a pig liver volume in the undeformed state and it contains 147 image slices. The image stack has the name: 'LiverNoDeformation.tif' and has a size of around 37 MB. The image stack was obtained using CT-scan and one can refer to [Amy Elizabeth Kerdok, 2006] for the details on the procurement and processing of the liver used for the scan, the scanning equipments used, scanning procedure employed, and further details on the image stack compiled etc. The details which are of interest to us are: 512 X 512 pixels make up each image in the stack, the distance between pixels (in both directions) = 0.29 mm, distance between consecutive images in the stack = 1 mm, each pixel needs 8 bits of memory to describe it and each pixel is described by an 8 bit unsigned integer, the images are gray scale images, since the images are gray scale images and since each pixel is represented by an 8 bit unsigned integer each pixel can have an intensity between 0 to 255 (totally 256 intensity levels).

Use ImageJ

Use *File* → *Import* → *Raw…*, enter image information, and the image stack 'LiverNoDeformation.tif' will be displayed. The 50th and the 100th image are shown in Figure 2.1 and Figure 2.2 respectively.

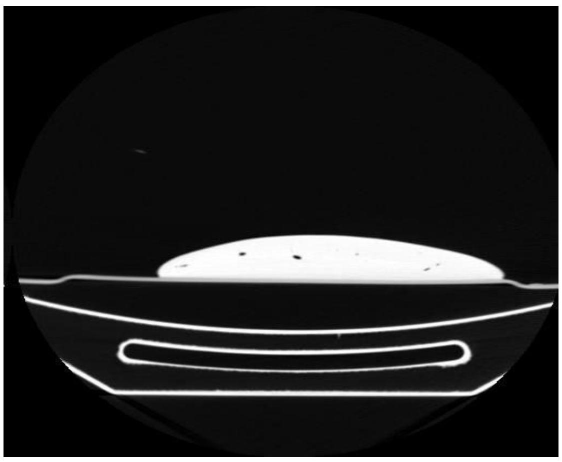

Figure 2.1 The 50th Image

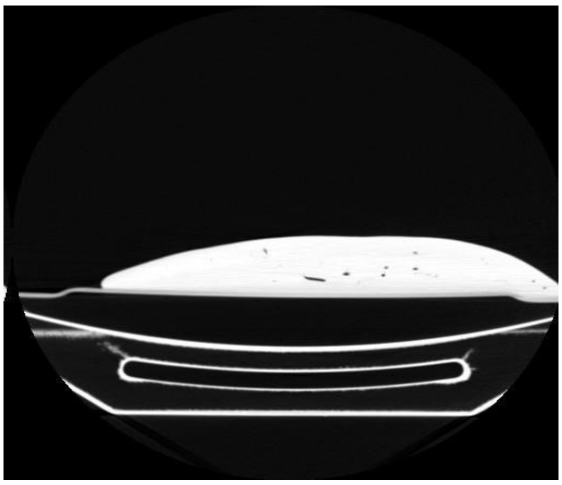

Figure 2.2 The 100th Image

It can be observed that the liver is located around the center of the images and the bottom portion of the images shows the part of the experimental set up supporting the liver. Our goal here is to make the pixels belonging to the liver very bright and making the intensity values of other pixels (which do not belong to liver) zero; we should do this for all the images in the image stack. This makes the next step (segmentation) that uses ITK-SNAP very trivial, since only the pixels that belong to the liver volume have some positive intensity values. This is accomplished as follows.

Use *Process* → *Math* → *Subtract…*, enter *Value = 200*, select *Ye*s to accept the processing of all 147 images. Select the bottom portion which does not belong to the liver in a yellow rectangular box and use *Process* → *Math* → *Multiply…*, then enter *Value = 0*, and process all 147 images. This will set the intensity values of the pixels selected in the rectangular box to zero. Follow the same procedure to eliminate the left side portion of the image stack that does not belong to the liver. Only those pixels that belong to the liver volume have some positive intensity value now. Use *Process* → *Math* → *Multiply…*, then enter *Value = 3* to multiply these intensity values by a factor of three. This brightens up the image of the liver. The 50th and 100th image, after undergoing processing with ImageJ are shown in Figure 2.3 and Figure 2.4.

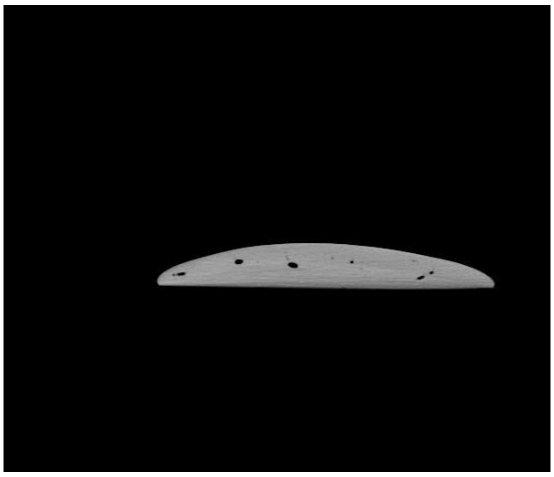

Figure 2.3 The 50th Image (After using ImageJ)

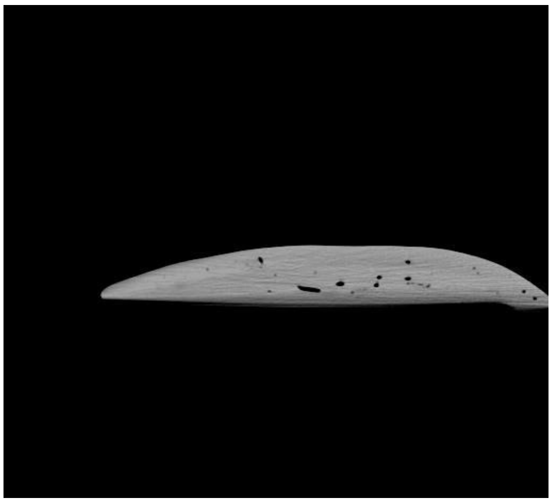

Figure 2.4 The 100th Image (After using ImageJ)

Now, *File → Save As → Raw Data…* has been used to save the image as a.raw. The original image stack represents only half of the complete image stack. Hence, use *Stack Reverser* to reverse the order of images in the image stack, save this new stack as b.raw, *Concatenate* a.raw and b.raw to get the full stack (with 294 images) 'LiverNoDeformation.raw'.

Use ITK-SNAP

Use *File → Open Grayscale Image…* to browse to the location of 'LiverNoDeformation.raw'. *Raw Binary File* should be selected as the image file format. Supply the image details (from 4.1) when asked. Image stack will be displayed. Select *SNAP Tool* in the *IRIS Toolbox* now. Make sure that the region selected for segmentation (through red dotted lines) includes the whole liver, and then press *Segment 3D*. Select the *Intensity Regions* radio button now, and press *Preprocess Image* and *Intensity Region Filter* window pops up. Select the proper settings here, proceed to the next step and *Add Bubble*, and *Iterate Continuously*. We can see now how bubble slowly changes its shape to the shape of the image on the screen. When shape of the bubble assumes the shape of the image on the screen (in all the windows), segmentation is complete and we should press *Finish*. All the images in the image stack have been segmented by now. Use *Segmentation → Save As Mesh…*

to save the segmented volume that represents the liver as a surface mesh made of triangle faces. The surface mesh file is named as ‘liver.stl’. Size of the file is 86.0 MB.

Use MeshLab

Open ‘liver.stl’ in MeshLab. The liver as seen in MeshLab is shown in Figure 2.5. The liver is represented here by 288190 vertices or 576376 triangle faces. We use *Quadric Edge Collapse Decimation* filter to reduce the total number of faces representing the organ. The same liver represented by 1002 vertices (2000 faces) now is shown in Figure 2.6 and represented by 102 vertices (200 faces) is shown in Figure 2.7. We use *MLS Projection*, *Laplacian Smooth* and *Taubin Smooth* filters to improve triangle quality. The liver represented by 102 vertices (or 200 faces) only but with well-shaped triangles is shown in Figure 2.8. We will save this model (liver represented by 102 vertices or 200 faces, all triangles being well shaped) as ‘liver102vprocessed.stl’. The size of the file is around 60 KB.

Now we have obtained the 3D model of the liver (shown in Figure 2.5 to Figure 2.8).

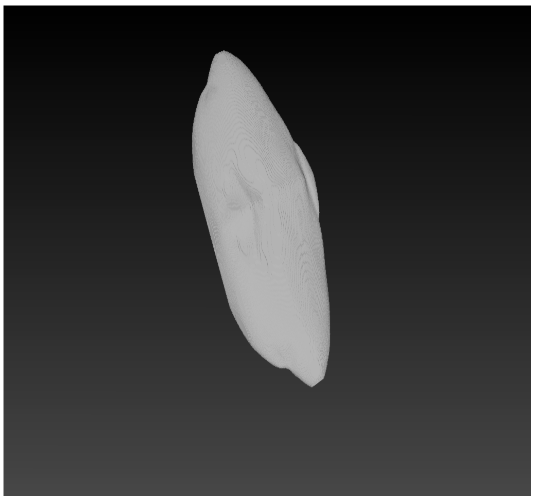

Figure 2.5 The Liver Displayed in MeshLab (288190 Vertices or 576376 Faces)

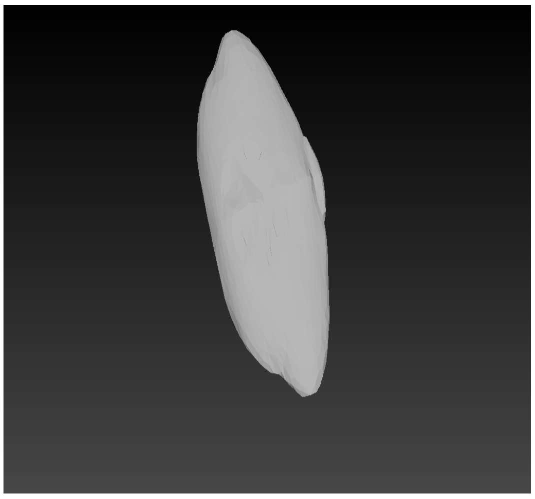

Figure 2.6 The Liver Displayed in MeshLab (1002 Vertices or 2000 Faces)

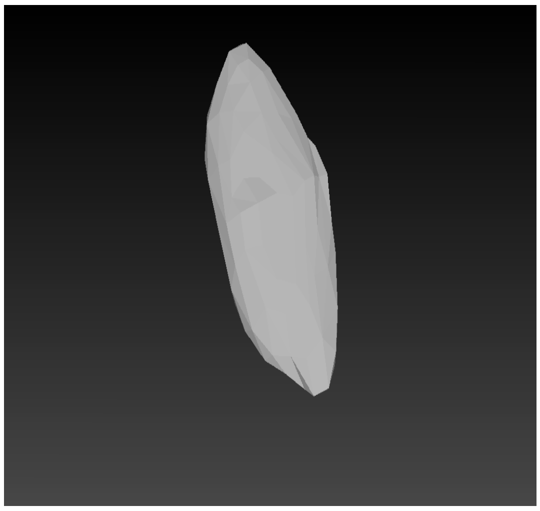

Figure 2.7 The Liver Displayed in MeshLab (102 Vertices or 200 Faces)

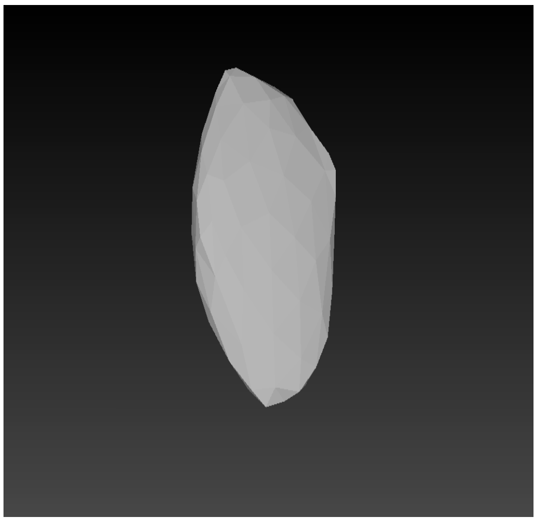

Figure 2.8 The Liver Displayed in MeshLab (Well Shaped 200 Faces)

2.2 The 3D Reconstruction of Human Kidney using Images from the Visible Human Dataset

This section follows similar procedure as the previous section to obtain the geometry of human kidney, but instead of using a readily available image stack (patient specific or representative) present section utilizes the Visible Human Dataset (VHD) [VHD, n.d.] to obtain the geometry of human kidney. Visible Human Data Set (also known as The Visible Human Project Image Data Set or The Visible Human Project Data Sets) is an anatomical data set developed under a contract from the National Library of Medicine (NLM) [NLM, n.d.] by the Departments of Cellular and Structural Biology, and Radiology, University of Colorado School of Medicine. One can note that images from VHD may be downloaded for free, after obtaining a free license from the National Library of Medicine (NLM) which is a part of the National Institutes of Health (NIH) [NIH, n.d.]; VHD is a part of the more ambitious Visible Human Project (VHP) [VHP, n.d.].

VHD contains CT, MRI and cryosection images. In the present section, only normal CT-scan images of visible human male and female are used. Images in the ‘png’ format are used since this is the format recommended by VHP. File size of CT-scan images is small, and the images are good enough for reconstructing a 3D model of whole kidney (i.e., inner (or finer) details of kidney are not present; reconstructed 3D model represents just the outer surface of kidney). One can also easily identify human kidney in the CT-scan images of VHD.

The next three subsections illustrate the process of obtaining the geometry of human kidney using VHD and some free software packages (same as the ones used in the previous section).

2.2.1 Using ImageJ to Form an Image Stack

CT-scan images for visible human male and female are available from head to toe. Out of these images, one has to identify the images which belong to kidney. Upon viewing individual images in ImageJ, and upon consulting references like [VOXEL-MAN, n.d.; Henry Gray, 1918], one can conclude that for visible human male, both left are right kidneys are contained between the images 'cvm1551f.png' and 'cvm1692f.png' (47 images in total). Similarly, for visible human female, both left and right kidneys are contained between the images 'cvf1564f.png' and 'cvf1693f.png' (130 images in total). These 47 images for male, and 130 images for female, have to be copied into two separate empty folders. To form an image stack for male, select the menu item 'File -> Import -> Image Sequence…', browse to the location of the folder containing 47 images and select the first image in the folder and follow the prompts (with default options); all 47 images are now displayed in ImageJ as an image stack; now, select the menu item 'File -> Save As -> Raw Data…' to save the image stack in the '*.raw' format (where * is the name given). Similar procedure may be followed to obtain an image stack for the female.

2.2.2 Using ITK-SNAP to Perform Segmentation and 3D Reconstruction

ITK-SNAP does the segmentation and 3D reconstruction to the correct scale. Hence header information for the images in the image stack is essential. VHD contains header information for each of the images in its database. Upon going through the header files of each of the 47 images of male, one can note that the following header information is identical for all the 47 images: Image matrix size – X = 512, Image matrix size – Y = 512, Image dimension – X = 460 mm, Image dimension –Y = 460 mm, Image pixel size – X = 0.898438, Image pixel size – Y = 0.898438, Screen

Format = 16 bit, Spacing between scans = 3 mm. Similarly, the following header information is identical for all the 130 female images: Image matrix size – X = 512, Image matrix size – Y = 512, Image dimension – X = 480 mm, Image dimension –Y = 480 mm, Image pixel size – X = 0.9375, Image pixel size – Y = 0.9375, Screen Format = 16 bit, Spacing between scans = 1 mm.

Now, the method of reconstructing the left kidney of the male is explained in detail, with illustrations. The same method may be employed to reconstruct the right kidney of the male, and the left and right kidneys of the female.

Select the menu item 'File -> Open Greyscale Image…', browse to the location of the image stack for male, follow the prompts and supply the header information. As noted in the first paragraph of this subsection, the 'missing header information' to be supplied for the image stack (for male) is: Image dimensions: X: 512, Y: 512, Z: 47, Voxel spacing: X = 0.898438, Y = 0.898438, Z = 3, Voxel representation: 16 bit, unsigned. Once the header information is supplied, image stack is displayed in ITK-SNAP. One can browse through all the 47 images in the image stack. For illustration purposes, 17th image and 33rd image in the image stack (i.e., 'cvm1602f.png' and 'cvm1650f.png' in the VHD) are shown in Figure 2.9 and Figure 2.10 respectively; also, the left and right kidneys are identified in Figure 2.9 and Figure 2.10, by making use of illustrations from [VOXEL-MAN, n.d.; Henry Gray, 1918].

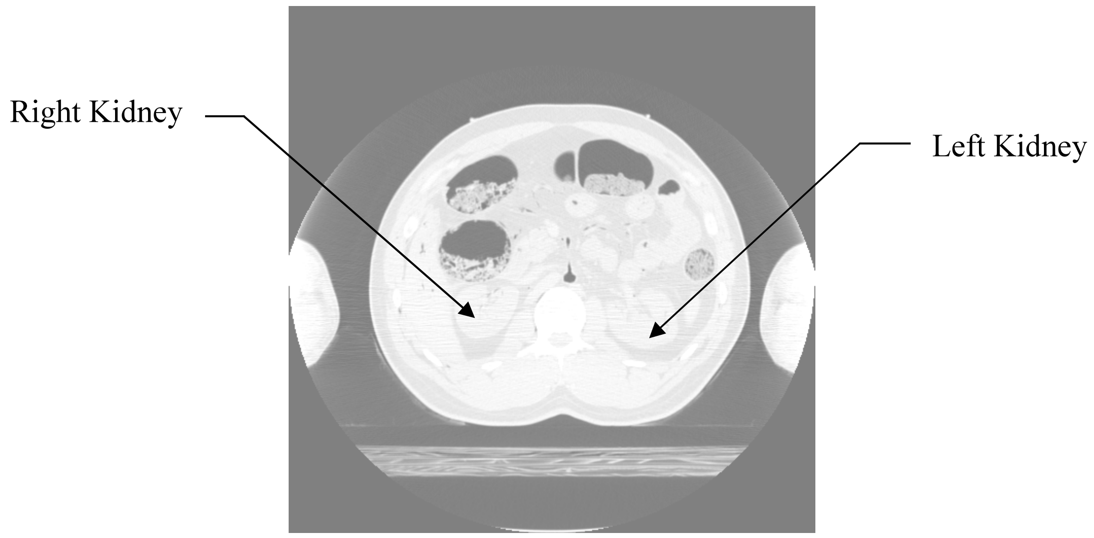

Figure 2.9 The 17th Image in the Image Stack

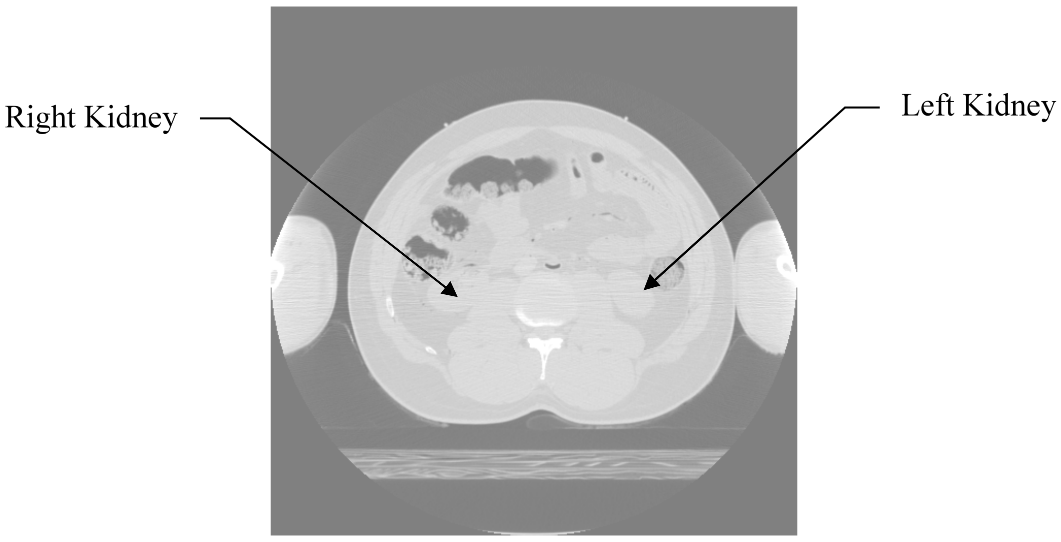

Figure 2.10 The 33rd Image in the Image Stack

Now the task is to do the segmentation. Select 'Polygon tool' from the 'IRIS Toolbox', for slice-by-slice manual segmentation. Select 'continuous' radio button under 'Polygon Tool'. Click and drag the mouse cursor along the edge of the left kidney (as seen in the axial view (window)), carefully. This draws the contour of the edge of the left kidney. Right click on the image, and select the 'accept' button to create the segmentation for the image on display. This process has to be repeated for all images in the image stack, which contain pixels that belong to the left kidney. For illustration purposes, 17th image and 33rd image in the image stack, after segmentation, are shown in Figure 2.11 and Figure 2.12 respectively.

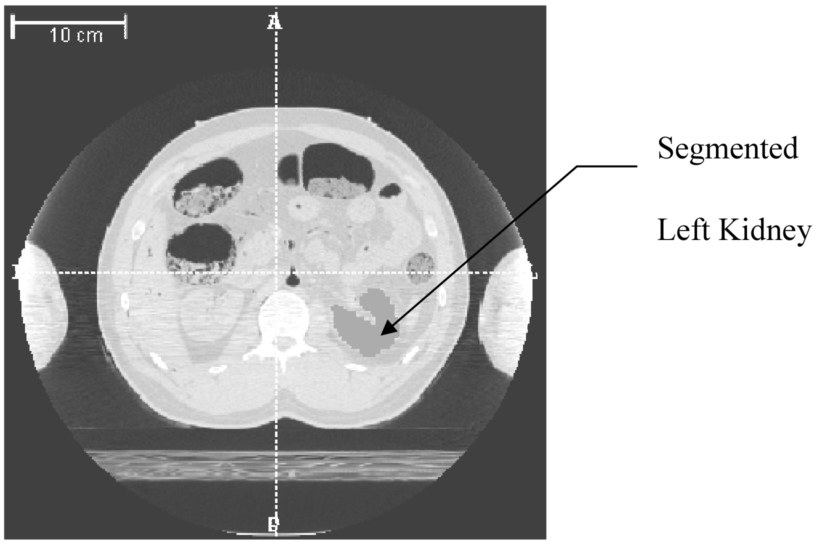

Figure 2.11 The 17th Image in the Image Stack (After Segmentation)

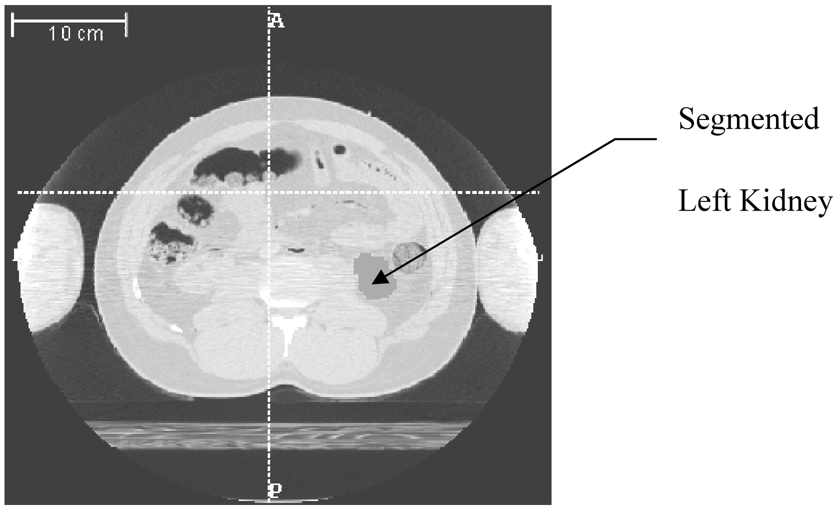

Figure 2.12 The 33rd Image in the Image Stack (After Segmentation)

Once the segmentation is over, 3D reconstruction is to be carried out. This is accomplished by the menu item 'Segmentation -> Save As Mesh…', following prompts, browsing to the location where the reconstructed model is to be stored, and giving a name in the format 'path*.stl' for the file that represents the reconstructed

3D model (where 'path' is the complete path, e.g., C:\Users\PC\Desktop\*.stl, and * is any file name).

Now, a 3D reconstruction of the left kidney for the visible human male is over. Similar process may be followed to reconstruct the right kidney of the visible human male, and the left and right kidneys of the visible human female.

2.2.3 Using MeshLab to Reduce the Total Number of Faces Describing the 3D Model

The 3D model of kidney obtained through the use of ITK-SNAP typically is of very large size and typically is described by a very large number of surface triangles. MeshLab could be very helpful in reducing the total number of surface triangles that are needed to describe the 3D model satisfactorily. It also serves as a tool to smoothen the reconstructed 3D geometry; after using smoothing features provided by MeshLab, it may be necessary to scale the reconstructed 3D models to the correct dimensions, if the original dimensions are to be strictly retained. MeshLab can also improve the triangle quality of surface triangles of the 3D model. It can also reduce the file size.

The 3D models of kidney, after undergoing processing with MeshLab, are shown in the next subsection.

2.2.4 Reconstructed Geometry

Reconstructed left kidney of the male, after undergoing processing through MeshLab, is shown in Figure 2.13. Similarly, reconstructed right kidney of the male, after undergoing processing through MeshLab, is shown in Figure 2.14. Reconstructed left kidney of the female is shown in Figure 2.15. Reconstructed right kidney of the female is shown in Figure 2.16. All the four 3D models are made up of 1000 surface triangles. While obtaining these four models, job of MeshLab is to smoothen the 3D models reconstructed through ITK-SNAP and to reduce the total number of surface triangles to 1000.

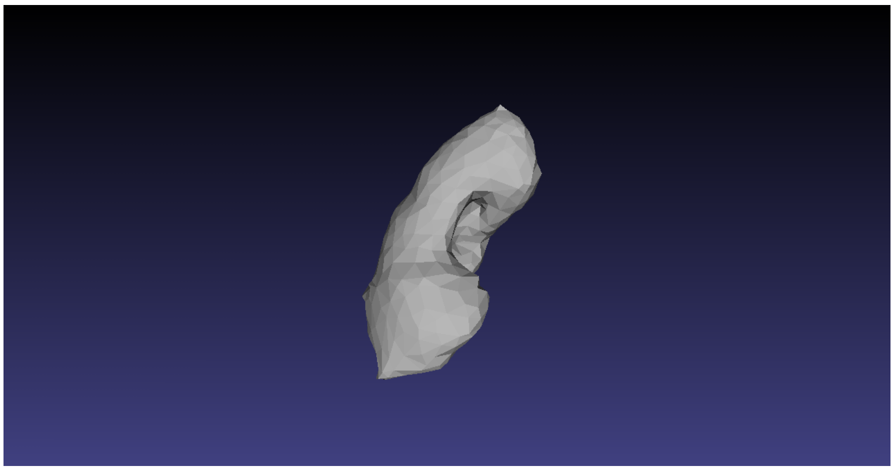

Figure 2.13 Reconstructed Left Kidney of Male

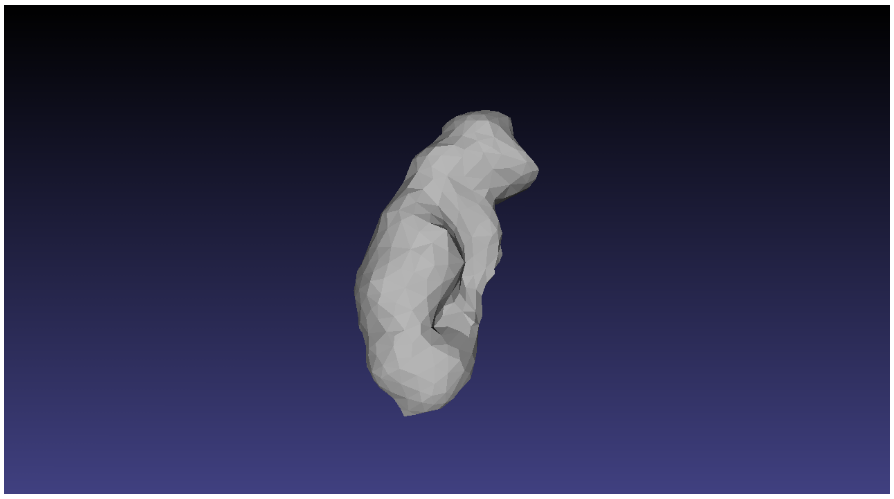

Figure 2.14 Reconstructed Right Kidney of Male

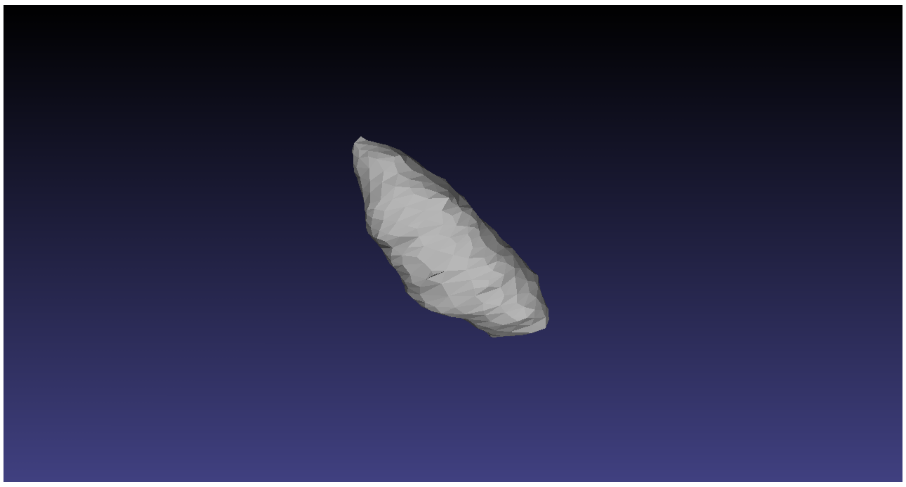

Figure 2.15 Reconstructed Left Kidney of Female

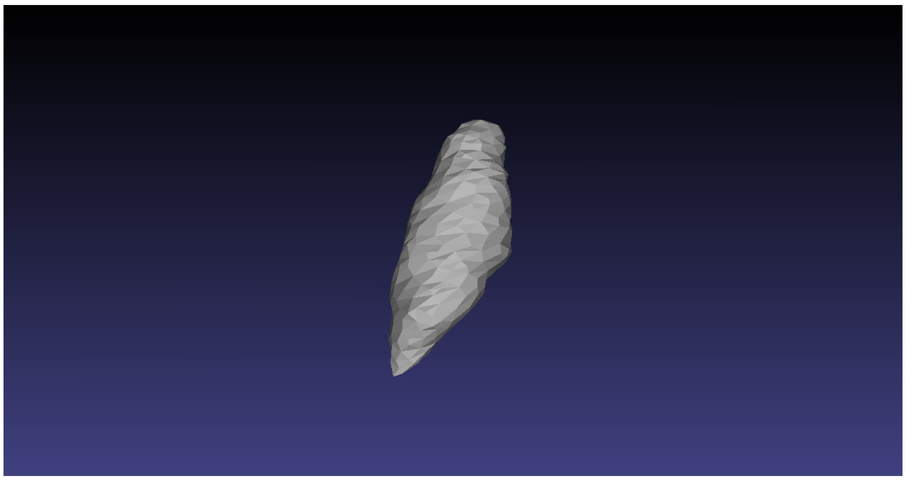

Figure 2.16 Reconstructed Right Kidney of Female

2.2.5 Discussion

In this subsection, 3D model of human kidney is extracted from CT-scan images from the VHD, using free software packages. The free software packages used are: 1) ImageJ 2) ITK-SNAP 3) MeshLab. The organs reconstructed are: 1) left kidney of visible human male 2) right kidney of visible human male 3) left kidney of visible human female 4) right kidney of visible human female. All the four models are in 'stl' format.

Use of free software packages together with images that may be obtained for free, as has been done in the present work, makes it possible to obtain the geometry of a representative human kidney, completely for free. Buying a 3D model of a human kidney, or using a commercial software package to extract 3D models from image sequences, cost (sometimes significant amount of) money. In the present approach, user can control how finely the geometry should be described (using the free software package MeshLab). Since MeshLab can improve the quality of the surface mesh that describes a reconstructed 3D model, the reconstructed 3D model that has undergone processing with MeshLab can be used in a finite element analysis after converting the surface model to a solid model using software packages like Rhinoceros. The method used to extract the geometry of a kidney, as illustrated in the present work, may be used to extract other whole biological organs from VHD.

It may be noted that the method given here to obtain the 3D models of human kidney need not be followed rigidly. It is good to read the documentation for the software packages used here, and one can experiment with the various options provided by the software packages instead of rigidly following the method illustrated in this work. For example, instead of tracing the boundary of the kidney in each of the images through the mouse pointer, the 'Paintbrush tool' provided by ITK-SNAP can be tried out to carry out the segmentation; ITK-SNAP also provides a tool that can do semi-automatic segmentation.

As to the limitations, present work uses only CT-scan images. Although these are found to be sufficient to obtain the geometry of a whole kidney, whenever the reconstructed geometry should include the finer details of the kidney, or whenever some other organ is to be extracted from VHD, there is a possibility that other types of images (e.g., MRI images, cryosection images) are more suited in some cases. One more limitation with the present work is that multiple software packages need to be downloaded, installed and used here.

2.3 Making use of CAELinux

The previous two subsections (i.e., section 2.1 and section 2.2) use the free software packages ImageJ, ITK-SNAP, and MeshLab to obtain the geometry of biological organs. To avoid installing these three software packages one-by-one, instead of Windows or the normal variety of Linux, one can use an operating system called CAELinux [CAELinux, n.d.] which includes the three software packages.

CAELinux is a Linux distribution which is bundled with free software packages related to Computer Aided Engineering (CAE). The free software packages include software that can build a three dimensional solid model, programs that can mesh a geometry, software for carrying out Finite Element Analysis (FEA), programs that can carry out image processing etc. One can also note that CAELinux is a free and open source operating system and all software packages that are included in the operating system are also free. Hence one can see that CAELinux could be a very useful tool in application areas like surgical simulation which require three dimensional reconstructions of biological organs (since CAELinux includes software packages required for these tasks).

2.4 Using the Reconstructed Geometry in Finite Element Analyses

This section performs two FE analyses on the liver constructed in the section 2.1. While the analyses do not represent typical biomedical engineering analyses, they clearly show that the surface mesh (which represents the reconstructed liver) obtained by following the procedure presented in the section 2.1 is of good quality. The software packages Rhinoceros [Rhinoceros, n.d.] and ANSYS [ANSYS, n.d.] are used in the present section. Two analyses are carried out in the present section: one linear analysis, and one nonlinear analysis.

3D models of biological organs obtained in the previous sections are surface meshes made up of triangles. Hence, in this section, the commercial software Rhinoceros is used to convert the 3D surface (surface model of the liver obtained in section 2.1) into a 3D solid so that the 3D solid model can be used in the FE analyses. This conversion from the 3D surface to 3D solid may be achieved as follows: Open 'liver102vprocessed.stl' in Rhinoceros. Use the command ***MeshToNurb***, and the resulting solid may be saved in ACIS (.sat) format using ***Save As…***. The file is named as 'liver102vprocessed.sat'. Size of the file is around 500 KB.

2.4.1 Linear Analysis

The 3D model of the liver 'liver102vprocessed.sat' is imported into ANSYS and used in a finite element analysis. The purpose of carrying out the FE analysis is just to show that 3D models of the organs obtained through the procedure presented in this paper do not pose serious problems during meshing. A test load, boundary condition and material are assumed. Figure 2.17 shows the deformed shape of the liver superimposed over the undeformed liver, as seen in ANSYS.

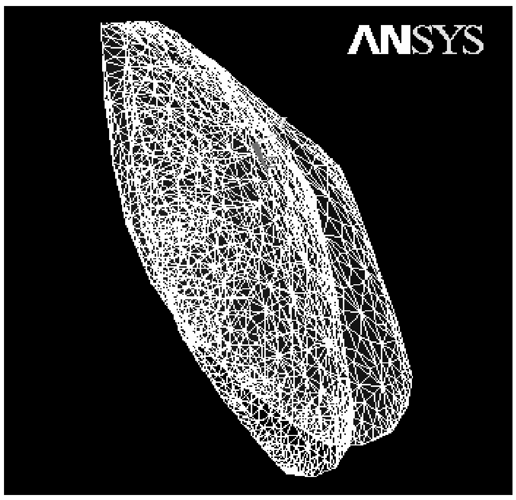

Figure 2.17 The Liver (Deformed + Undeformed)

2.4.2 Nonlinear Analysis

The 3D model of the liver (the 3D solid 'liver102vprocessed.sat') is ***imported*** into ANSYS and used in a finite element analysis. This is a dummy analysis which demonstrates that the geometry obtained by following the procedure illustrated in the previous sections could be used in a nonlinear finite element analysis. A box shaped imaginary cyst is created inside the liver. Cyst is placed approximately at the center of the liver. Figure 2.18 shows the liver and the cyst.

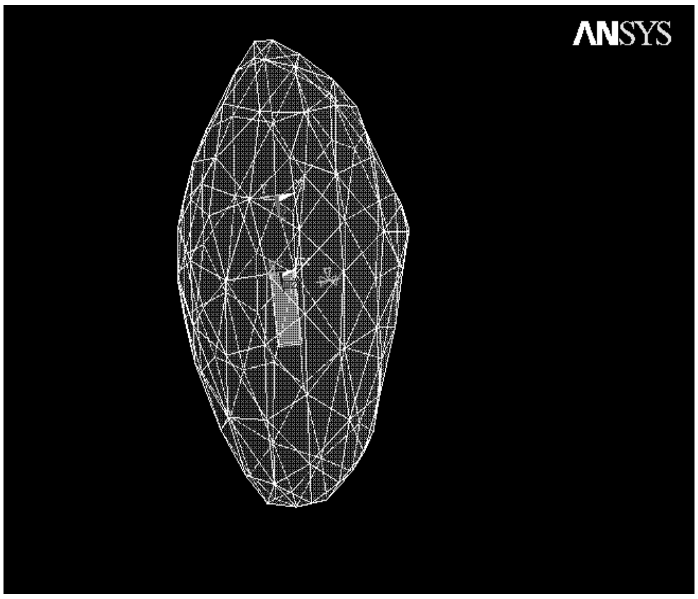

Figure 2.18 The Liver and the Cyst Displayed in ANSYS

Both liver and cyst are assumed linear elastic but they both can undergo large deformation, hence making the analysis nonlinear. The Young's modulus for the liver is taken as 200000 N/mm^2 and its Poisson's ratio is taken as 0.33 (this is equivalent to assuming that liver is made out of steel!). For the cyst, Young's modulus is taken as 100000 N/mm^2 and Poisson's ratio is taken as 0.30. Of course, these values are no way near to the real values. Next, one of the triangles forming the bottom surface of the liver is 'fixed' or it is assumed that the displacements in any direction over all of this triangle surface is zero. A 3 mm displacement is applied at a vertex located on the top surface. The displacement is applied in the Y-direction which is roughly along the thickness of the liver. The element used is: ***Solid Tet 10node 187***. The element is capable of handling large deformation. Element shape is ***Tetrahedron*** and ***Free*** meshing has been used; this is to ensure that the complex geometry is properly meshed. ***Solution → Analysis Type → Sol'n Controls → Basic → Analysis Options*** is set to ***Large Displacement Static***. Analysis is carried out and the displacement along Y-direction, at a vertex which is close to the vertex where the point load is applied, is found to be 0.19387 mm. Next, the same analysis is carried out but without any cyst inside the liver and the displacement at the same vertex along Y-direction is noted down again. It is found to be 0.18848 mm. One can clearly see that the difference in the displacements obtained at the same point for the two cases is the result of the presence or the absence of the cyst inside the liver.

2.5 Summary

A procedure for obtaining models of biological organs from image sequences that are obtained through CT-scanning has been presented in this chapter. The procedure uses free software packages only; the free software packages needed are: ImageJ, ITK-SNAP, and MeshLab. While using this procedure, user can control how fine the organ description should be. The procedure has been illustrated by successfully reconstructing a pig liver. The same procedure has been used to obtain the 3D surface model of a representative human kidney from CT-scan images from the Visible Human Dataset (VHD). The practice of extracting the geometry of biological organs completely for free, as illustrated in the present work, could be a free alternative to the use of expensive commercial software packages or to the purchase of a digital model. The procedure can be of use to obtain patient specific organ geometry, or to obtain representative organ geometry. It is also demonstrated that finite element simulations can be performed on geometry obtained using the procedure illustrated in the present chapter; hence the procedure could be of use in surgery planning and surgery simulation since both of these extensively use finite elements for numerical simulations.

Chapter 3: Real-time Simulation of Biological Organs using the Boundary Element Method

In this chapter, the possibility of simulating biological organs in real-time using the Boundary Element Method (BEM) is investigated. Liver and kidney are the biological organs considered. A Graphics Processing Unit (GPU), and a computer cluster are employed to speed up the computations.

This chapter utilizes constant boundary elements of triangular shape. Next three paragraphs mention the advantages of using BEM, advantages of using (boundary) elements of triangular shape, and advantages of using constant (boundary) elements respectively.

The BEM needs a meshing of only the boundary of any geometry (at least for linear elastostatic problems). Hence lesser number of elements can describe geometry. Literature mentions that BEM shows very good scalability when parallelized to execute on a computer containing multiple processors; parallelizing the BEM would help in reducing solution times. BEM is generally thought to be an efficient numerical technique. In BEM, some of the unknowns can be displacements while at the same time the remaining unknowns can be tractions; hence computation of tractions from displacements is not needed (during postprocessing); during surgical simulations, often the goal is to obtain reaction forces corresponding to prescribed values of displacements (i.e., goal is not to obtain displacements that correspond to known values of forces), and one can observe that BEM could be better than FEM for such problems. Often, one needs to know the solutions only on the boundary of an organ (or only at the nodes of boundary elements), and with BEM, there is no need to calculate solutions at internal points to get the solutions at the boundary, while FEM unnecessarily calculates solutions at internal nodes also. It is widely mentioned in the literature that collision detection and rendering are easy with the type of object representation used in BEM. The system of equations resulting from BEM is dense while FEM produces sparse ‘characteristic’ matrices. Since the algorithms and routines that take advantage of the sparseness of characteristic matrices are rare, BEM has an upper hand in this case (also since BEM needs meshing of only the boundary of the geometry and hence lesser number of elements can describe a geometry).

There are advantages in using boundary elements of triangular shape. Any complicated geometry can easily be represented by surface triangles. One can construct a geometry using any standard CAD software package and then use the 'stl export' option available in the package to obtain the surface mesh as an STL file (exporting a mesh as a VRML file may also serve the purpose). Mesh processing tools and remeshing tools (e.g., MeshLab [MeshLab, n.d.], ReMESH [ReMESH, n.d.]) are widely available for surface meshes made up of triangles. Tools that can modify, edit, heal, and improve the quality of surface meshes that are made up of triangles (i.e., tools that can convert a mesh made up of ill shaped triangles to a mesh made up of well shaped triangles) are also widely available. One can note that it is easy to obtain 3D models described by surface triangles from 3D scan. One can also note that many of the software packages that can do 3D reconstruction of biological organs from 2D image sequences have the 'stl export' option. Literature tells that it is easier to perform rendering and collision detection with meshes made up of surface triangles.

Now, advantages of using constant boundary elements are explained. Constant boundary elements are fast in the sense that they do not need complicated shape functions that are computationally expensive. Constant elements are easy to use and program. No connectivity information is needed if constant elements are used. This makes it easy to parallelize the code, and the resulting code is highly scalable. Constant elements are suitable for handling multiple regions since nodes are located completely within the elements (not on the edges or corners). Of course, accuracy may be poor for a given number of elements when compared to linear and quadratic elements. For applications like the real-time simulation of biological organs, accuracy may not be too important when compared to speed.

To use BEM for the real-time simulation of biological organs, one needs a source code for BEM. Although many BEM libraries are available, present author could not find any suitable library that is of help for solving the problem in hand. For example, Helsinki BEM Library [Helsinki BEM, n.d.] is a MATLAB source code library for solving problems that obey the Laplace or Poisson equation. The web source [http://www.boundary-element-method.com/, n.d.] contains codes that are specifically useful for solving acoustics problems. The source also contains codes for solving Laplace problems and Helmholtz problems. Book [Ang W.T., 2007] gives FORTRAN codes for Laplace's equation and Helmholtz equation, in two and three dimensions. The codes can be freely downloaded from the website for the book. A BEM code for two dimensional (2D) pulsating cylinders is available from the MATLAB Central (from MathWorks) [MATLAB Central, n.d.]. One can note that none of the above sources provide source codes for 3D elasticity. Fast Multipole Boundary Element Method (FastBEM) software is available from [Yijun Liu, n.d.]; it provides software for 3D elasticity also; source codes are not available still. The website for the book [Beer G., et al., 2008] contains many programs in Fortran. It contains programs for 3D elasticity also. Author has downloaded the programs, but has found that the program that solves the 3D elasticity problem does not give accurate solutions (the program gives the same output for different inputs); the concerned authors do not provide any support for the programs. Hence the present author had to write his own BEM codes, from scratch.

3.1 Description of the BEM Code Developed

Since it was inevitable to develop one's own code for BEM, present author first wrote a MATLAB code that can be used to solve any three dimensional linear elastostatic problem (without considering body forces) using constant boundary elements. Next, a Fortran translation of the MATLAB code was developed. Finally, a parallelized version of the Fortran code was developed. All these codes are available for download as free and open source software from the source [Kirana Kumara P, 2014a], under the very permissive MIT License.

All theory behind the code, including many of the formulae, is taken from [Beer G., et al., 2008; Ang W.T., 2007; C.A. Brebbia and J. Dominguez, n.d.; Brebbia CA, 1978; Youssef F. Rashed, n.d.]. Author also used many other resources while writing the code. A few of the many noteworthy ones are [Alastair McKinstry, et al., n.d.; Blaise Barney, n.d.; "Intel Math Kernel Library Reference Manual", n.d.; "ScaLAPACK Example Programs", n.d.; "ScaLAPACK - Scalable Linear Algebra PACKage", n.d.; "Parallel ESSL Guide and Reference", n.d.]. This author used the information contained in many of the online forums also, especially while running/compiling/developing the code.

3.1.1 Theory behind the Code

From the Appendix of [Beer G., et al., 2008], for static elasticity, in indicial notation, the displacement u_i at an internal point *P*, in the absence of initial stresses and strains, is given by

$$u_i(p) = \int_S U_{ij}(P,Q)t_j(Q)dS - \int_S T_{ij}(P,Q)u_j(Q)dS \tag{3.1}$$

where u_i, t_i (or u_j, t_j) are the displacements and tractions

$U_{ij}(P,Q), T_{ij}(P,Q)$ are called the fundamental solutions

P is called the source point and Q is called the field point

S is the surface (for 3D problems) which represents the geometry

For 3D problems, the fundamental solutions are given by

$$U_{ij}(P,Q) = \frac{C}{r}(C_1\delta_{ij} + r_{,i}r_{,j}) \tag{3.2}$$

$$T_{ij}(P,Q) = \frac{-C_2}{r^n}\begin{bmatrix} \left(C_3\delta_{ij} + (n+1)r_{,i}r_{,j}\right)\cos\theta \\ -C_3\left(1-\delta_{ij}\right)\left(n_j r_{,i} - n_i r_{,j}\right) \end{bmatrix} \tag{3.3}$$

In (3.2) and (3.3), r is the distance between P and Q, and n_i and n_j are the outward normals. The derivative of r with respect to the Cartesian axis i is denoted as $r_{,i}$ and the derivative of r with respect to the Cartesian axis j is denoted as $r_{,j}$. The term $\cos\theta$ is given by

$$\cos\theta = \frac{1}{r}\vec{r} \bullet \vec{n} \tag{3.4}$$

δ_{ij} is given by

$$\delta_{ij} = \begin{Bmatrix} 1 for(i = j) \\ 0 for(i \neq j) \end{Bmatrix} \tag{3.5}$$

The values of the constants are given by

$$n = 2 \qquad C = \frac{1}{16\pi G(1-\nu)} \qquad C_1 = 3 - 4\nu \qquad C_2 = \frac{1}{8\pi(1-\nu)} \qquad C_3 = 1 - 2\nu \tag{3.6}$$

where ν is the Poisson's ratio.

The shear modulus G is given by

$$G = \frac{E}{2(1+\nu)} \tag{3.7}$$

where E is the modulus of elasticity.

One may note that the equations (3.1) to (3.7) above have been copied from the book [Beer G., et al., 2008].

In the present work, a 3D solid is represented by 3D boundary triangles, i.e., 3D triangular surface mesh. T is the total number of triangles which together represent the 3D solid; hence, the total number of elements is equal to T. Let S^m be the surface of the element with element number m. Since constant elements are used, over each of the elements, displacements and tractions are assumed constant. For each of the elements, either displacement or traction is known, the other being an unknown that has to be calculated. In this work, solution is sought only on the boundary. For a point P on the boundary of a solid, if P is located inside a smooth region of the boundary, (3.1) can be reduced to the following three equations, i.e., (3.8), (3.9) and (3.10).

$$\frac{1}{2}u_x(P_e) = \sum_{m=1}^{T}\left[A1 + A2 + A3 - B1 - B2 - B3\right] \tag{3.8}$$

$$\frac{1}{2}u_y(P_e) = \sum_{m=1}^{T}\left[A4 + A5 + A6 - B4 - B5 - B6\right] \tag{3.9}$$

$$\frac{1}{2}u_z(P_e) = \sum_{m=1}^{T}\left[A7 + A8 + A9 - B7 - B8 - B9\right] \tag{3.10}$$

where $A1 = t_x(Q_m)\int_{S^m} U_{xx}(P_e, Q_m)dS^m \qquad A2 = t_y(Q_m)\int_{S^m} U_{xy}(P_e, Q_m)dS^m$

$$A3 = t_z(Q_m)\int_{S^m} U_{xz}(P_e, Q_m)dS^m$$

$$B1 = u_x(Q_m)\int_{S^m} T_{xx}(P_e, Q_m)dS^m \qquad B2 = u_y(Q_m)\int_{S^m} T_{xy}(P_e, Q_m)dS^m$$

$$B3 = u_z(Q_m)\int_{S^m} T_{xz}(P_e, Q_m)dS^m$$

$$A4 = t_x(Q_m)\int_{S^m} U_{yx}(P_e, Q_m)dS^m \qquad A5 = t_y(Q_m)\int_{S^m} U_{yy}(P_e, Q_m)dS^m$$

$$A6 = t_z(Q_m)\int_{S^m} U_{yz}(P_e, Q_m)dS^m$$

$$B4 = u_x(Q_m)\int_{S^m} T_{yx}(P_e, Q_m)dS^m \qquad B5 = u_y(Q_m)\int_{S^m} T_{yy}(P_e, Q_m)dS^m$$

$$B6 = u_z(Q_m)\int_{S^m} T_{yz}(P_e, Q_m)dS^m$$

$$A7 = t_x(Q_m)\int_{S^m} U_{zx}(P_e, Q_m)dS^m \qquad A8 = t_y(Q_m)\int_{S^m} U_{zy}(P_e, Q_m)dS^m$$

$$A9 = t_z(Q_m)\int_{S^m} U_{zz}(P_e, Q_m)dS^m$$

$$B7 = u_x(Q_m)\int_{S^m} T_{zx}(P_e, Q_m)dS^m \qquad B8 = u_y(Q_m)\int_{S^m} T_{zy}(P_e, Q_m)dS^m$$

$$B9 = u_z(Q_m)\int_{S^m} T_{zz}(P_e, Q_m)dS^m$$

Equations (3.8) - (3.10) may also be written in the expanded form given by the following three equations, i.e., (3.11), (3.12) and (3.13).

$$\frac{1}{2}u_x(P_e) = \left[A11 + A21 + A31 - B11 - B21 - B31\right]$$

$$+\left[A12 + A22 + A32 - B12 - B22 - B32\right] + \ldots$$

$$\ldots + \left[A1m + A2m + A3m - B1m - B2m - B3m\right] + \ldots$$

$$\ldots + \left[A1T + A2T + A3T - B1T - B2T - B3T\right]$$

(3.11)

$$\frac{1}{2}u_y(P_e) = \left[A41 + A51 + A61 - B41 - B51 - B61\right]$$

$$+\left[A42 + A52 + A62 - B42 - B52 - B62\right] + \ldots$$

$$\ldots + \left[A4m + A5m + A6m - B4m - B5m - B6m\right] + \ldots$$

$$\ldots + \left[A4T + A5T + A6T - B4T - B5T - B6T\right]$$

(3.12)

$$\frac{1}{2}u_z(P_e) = \left[A71 + A81 + A91 - B71 - B81 - B91\right]$$

$$+\left[A72 + A82 + A92 - B72 - B82 - B92\right] + \ldots$$

$$\ldots + \left[A7m + A8m + A9m - B7m - B8m - B9m\right] + \ldots$$

$$\ldots + \left[A7T + A8T + A9T - B7T - B8T - B9T\right]$$

(3.13)

where $A1m = t_x(Q_m)\int_{S^m} U_{xx}(P_e, Q_m)\,dS^m$ $\qquad A2m = t_y(Q_m)\int_{S^m} U_{xy}(P_e, Q_m)\,dS^m$

$$A3m = t_z(Q_m)\int_{S^m} U_{xz}(P_e, Q_m)\,dS^m$$

$$B1m = u_x(Q_m)\int_{S^m} T_{xx}(P_e,Q_m)dS^m \qquad B2m = u_y(Q_m)\int_{S^m} T_{xy}(P_e,Q_m)dS^m$$

$$B3m = u_z(Q_m)\int_{S^m} T_{xz}(P_e,Q_m)dS^m$$

$$A4m = t_x(Q_m)\int_{S^m} U_{yx}(P_e,Q_m)dS^m \qquad A5m = t_y(Q_m)\int_{S^m} U_{yy}(P_e,Q_m)dS^m$$

$$A6m = t_z(Q_m)\int_{S^m} U_{yz}(P_e,Q_m)dS^m$$

$$B4m = u_x(Q_m)\int_{S^m} T_{yx}(P_e,Q_m)dS^m \qquad B5m = u_y(Q_m)\int_{S^m} T_{yy}(P_e,Q_m)dS^m$$

$$B6m = u_z(Q_m)\int_{S^m} T_{yz}(P_e,Q_m)dS^m$$

$$A7m = t_x(Q_m)\int_{S^m} U_{zx}(P_e,Q_m)dS^m \qquad A8m = t_y(Q_m)\int_{S^m} U_{zy}(P_e,Q_m)dS^m$$

$$A9m = t_z(Q_m)\int_{S^m} U_{zz}(P_e,Q_m)dS^m$$

$$B7m = u_x(Q_m)\int_{S^m} T_{zx}(P_e,Q_m)dS^m \qquad B8m = u_y(Q_m)\int_{S^m} T_{zy}(P_e,Q_m)dS^m$$

$$B9m = u_z(Q_m)\int_{S^m} T_{zz}(P_e,Q_m)dS^m$$

where m takes values from 1 to T .

Equations (3.8) - (3.10) (or equations (3.11) - (3.13)) are the basic equations upon which the present code is developed. Since displacements and tractions are constants over each of the elements, for each of the elements, displacements and tractions are considered only for just one chosen point inside each element. P_e and Q_m refer to these points; here, the subscripts e or m in P_e or Q_m refer to the element number. The subscript e in P_e varies from 1 to T which is the total number of elements. Further, $m = e$ implies that $P_e = Q_m$. Hence, if a solid is discretized by T boundary elements, equation (3.8) - (3.10) (or equation (3.11)-(3.13)) give rise to a set of coupled $3T$

linear algebraic equations in $3T$ unknowns. Unknowns are either displacements (u_x, u_y or u_z at P_e or Q_m) or tractions (t_x, t_y or t_z at Q_m (or P_e when $e = m$)). For elements with prescribed displacements (u_x, u_y and u_z), the tractions (t_x, t_y and t_z) are the unknowns. On the other hand, for elements with prescribed tractions (t_x, t_y and t_z), the displacements (u_x, u_y and u_z) are the unknowns. The set of $3T$ algebraic equations may be written in the form

$$[K]_{3T\times 3T}\{U\}_{3T\times 1} = \{F\}_{3T\times 1} \tag{3.14}$$

where $\{U\}$ denotes the vector of unknowns, which consists of unknown displacements and unknown tractions. The matrix $[K]$ is fully populated, in general. Solving (3.14) for $\{U\}$, one can straight away obtain the values of the unknowns, be it unknown displacements or unknown tractions.

Now, the method used to find the integrals of the fundamental solutions over the elements is explained, i.e., the goal now is to evaluate the integrals

$$\int_{S^m} U_{xx}(P_e,Q_m)dS^m,\ \int_{S^m} T_{xx}(P_e,Q_m)dS^m,\ \int_{S^m} U_{xy}(P_e,Q_m)dS^m,\ \int_{S^m} T_{xy}(P_e,Q_m)dS^m,$$

$$\int_{S^m} U_{xz}(P_e,Q_m)dS^m,\ \int_{S^m} T_{xz}(P_e,Q_m)dS^m,\ \int_{S^m} U_{yx}(P_e,Q_m)dS^m,\ \int_{S^m} T_{yx}(P_e,Q_m)dS^m,$$

$$\int_{S^m} U_{yy}(P_e,Q_m)dS^m,\ \int_{S^m} T_{yy}(P_e,Q_m)dS^m,\ \int_{S^m} U_{yz}(P_e,Q_m)dS^m,\ \int_{S^m} T_{yz}(P_e,Q_m)dS^m,$$

$$\int_{S^m} U_{zx}(P_e,Q_m)dS^m,\ \int_{S^m} T_{zx}(P_e,Q_m)dS^m,\ \int_{S^m} U_{zy}(P_e,Q_m)dS^m,\ \int_{S^m} T_{zy}(P_e,Q_m)dS^m,$$

$$\int_{S^m} U_{zz}(P_e,Q_m)dS^m,\ \int_{S^m} T_{zz}(P_e,Q_m)dS^m$$

These integrals are evaluated by following the procedure explained in Chapter 6 of the book [Ang W.T., 2007]. One may note that the equations (3.15) to (3.26) below have been adapted/copied from the book [Ang W.T., 2007]. The integrals above are evaluated by using the formula:

$$\int_{S^m} f(x,y,z)dS^m = \int_0^1 \int_0^{1-v} f(x,y,z)J^m dudv$$

$$= \int_0^1 \int_0^1 f(x,y,z)(1-v)J^m dtdv$$

$$\cong \frac{1}{16}\sum_{k=1}^{16} f(t_k, v_k)$$

$$= \frac{1}{16}\sum_{k=1}^{16} f(x_k, y_k, z_k)(1-v_k)J^m \qquad (3.15)$$

Equation (3.15) may be written as

$$\int_{S^m} f(x,y,z)dS^m = \frac{1}{16}\sum_{k=1}^{16} f(x_k, y_k, z_k)(1-v_k)J^m \qquad (3.16)$$

In equation (3.16), $f(x,y,z)$ is the fundamental solution (i.e., U_{xx}, T_{xx}, U_{xy}, T_{xy}, U_{xz}, T_{xz}, U_{yx}, T_{yx}, U_{yy}, T_{yy}, U_{yz}, T_{yz}, U_{zx}, T_{zx}, U_{zy}, T_{zy}, U_{zz}, T_{zz}) which needs to be integrated over the element that has the element number m.

Let (x_a, y_a, z_a), (x_b, y_b, z_b) and (x_c, y_c, z_c) be the coordinates of the vertices which define the triangular boundary element m. Of course, the vertices always have to be properly ordered such that the normal vector to S^m points out of the 3D solid under consideration. Then J^m in (3.16) is given by

$$J^m = 2\sqrt{\sigma^m(\sigma^m - \alpha^m)(\sigma^m - \beta^m)(\sigma^m - \gamma^m)} \qquad (3.17)$$

$$\text{where } \sigma^m = \frac{\alpha^m + \beta^m + \gamma^m}{2}$$

$$\alpha^m = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2 + (z_a - z_b)^2}$$

$$\beta^m = \sqrt{(x_b - x_c)^2 + (y_b - y_c)^2 + (z_b - z_c)^2}$$

$$\gamma^m = \sqrt{(x_c - x_a)^2 + (y_c - y_a)^2 + (z_c - z_a)^2}$$

For the purpose of evaluating x_k, y_k and z_k in (3.16), the values for (t_k, v_k) are noted down as follows (*4 by 4* numerical integration is assumed here for the sake of simplicity of explanations).

$$(t_1,v_1)=\left(\frac{1}{4}+\frac{1}{4\sqrt{3}},\frac{1}{4}+\frac{1}{4\sqrt{3}}\right) \quad (t_2,v_2)=\left(\frac{1}{4}+\frac{1}{4\sqrt{3}},\frac{1}{4}-\frac{1}{4\sqrt{3}}\right)$$

$$(t_3,v_3)=\left(\frac{1}{4}-\frac{1}{4\sqrt{3}},\frac{1}{4}+\frac{1}{4\sqrt{3}}\right) \quad (t_4,v_4)=\left(\frac{1}{4}-\frac{1}{4\sqrt{3}},\frac{1}{4}-\frac{1}{4\sqrt{3}}\right)$$

$$(t_5,v_5)=\left(\frac{3}{4}+\frac{1}{4\sqrt{3}},\frac{1}{4}+\frac{1}{4\sqrt{3}}\right) \quad (t_6,v_6)=\left(\frac{3}{4}+\frac{1}{4\sqrt{3}},\frac{1}{4}-\frac{1}{4\sqrt{3}}\right)$$

$$(t_7,v_7)=\left(\frac{3}{4}-\frac{1}{4\sqrt{3}},\frac{1}{4}+\frac{1}{4\sqrt{3}}\right) \quad (t_8,v_8)=\left(\frac{3}{4}-\frac{1}{4\sqrt{3}},\frac{1}{4}-\frac{1}{4\sqrt{3}}\right)$$

$$(t_9,v_9)=\left(\frac{3}{4}+\frac{1}{4\sqrt{3}},\frac{3}{4}+\frac{1}{4\sqrt{3}}\right) \quad (t_{10},v_{10})=\left(\frac{3}{4}+\frac{1}{4\sqrt{3}},\frac{3}{4}-\frac{1}{4\sqrt{3}}\right)$$

$$(t_{11},v_{11})=\left(\frac{3}{4}-\frac{1}{4\sqrt{3}},\frac{3}{4}+\frac{1}{4\sqrt{3}}\right) \quad (t_{12},v_{12})=\left(\frac{3}{4}-\frac{1}{4\sqrt{3}},\frac{3}{4}-\frac{1}{4\sqrt{3}}\right)$$

$$(t_{13},v_{13})=\left(\frac{1}{4}+\frac{1}{4\sqrt{3}},\frac{3}{4}+\frac{1}{4\sqrt{3}}\right) \quad (t_{14},v_{14})=\left(\frac{1}{4}+\frac{1}{4\sqrt{3}},\frac{3}{4}-\frac{1}{4\sqrt{3}}\right)$$

$$(t_{15},v_{15})=\left(\frac{1}{4}-\frac{1}{4\sqrt{3}},\frac{3}{4}+\frac{1}{4\sqrt{3}}\right) \quad (t_{16},v_{16})=\left(\frac{1}{4}-\frac{1}{4\sqrt{3}},\frac{3}{4}-\frac{1}{4\sqrt{3}}\right)$$

(3.18)

The value of u_k is calculated by using the formula

$$u_k = t_k(1 - v_k) \tag{3.19}$$

Next, to calculate x_k, y_k and z_k in (3.16), one needs to also calculate the components of the unit normal vector to the element surface S^m. Again, assuming that (x_a, y_a, z_a), (x_b, y_b, z_b) and (x_c, y_c, z_c) are the coordinates of the vertices of the triangular boundary element m, the components of the unit normal vector in the x, y and z direction are given by

$$n_x^m = \frac{(y_b - y_a)(z_c - z_a) - (z_b - z_a)(y_c - y_a)}{d}$$

$$n_y^m = \frac{(z_b - z_a)(x_c - x_a) - (x_b - x_a)(z_c - z_a)}{d}$$

$$n_z^m = \frac{(x_b - x_a)(y_c - y_a) - (y_b - y_a)(x_c - x_a)}{d}$$

(3.20)

where $d = [\,((y_b - y_a)(z_c - z_a) - (z_b - z_a)(y_c - y_a))^2$

$$+((z_b - z_a)(x_c - x_a) - (x_b - x_a)(z_c - z_a))^2$$

$$+((x_b - x_a)(y_c - y_a) - (y_b - y_a)(x_c - x_a))^2\,]^{1/2}$$

Depending on the values of $\left|n_z^m\right|$ and $\left|n_y^m\right|$, x_k, y_k and z_k in (3.16) can be calculated by using either of the following equations (3.21) - (3.23).

If $\left|n_z^m\right| \geq \frac{1}{\sqrt{3}}$

$$x_k = (x_b - x_a)u_k + (x_c - x_a)v_k + x_a$$

$$y_k = (y_b - y_a)u_k + (y_c - y_a)v_k + y_a$$

$$z_k = -(n_z^m)^{-1}\left[n_x^m(x_k - x_a) + n_y^m(y_k - y_a)\right] + z_a$$

(3.21)

Else, if $\left|n_z^m\right| < \frac{1}{\sqrt{3}}$ and $\left|n_y^m\right| \geq \frac{1}{\sqrt{3}}$

$$x_k = (x_b - x_a)u_k + (x_c - x_a)v_k + x_a$$

$$z_k = (z_b - z_a)u_k + (z_c - z_a)v_k + z_a$$

$$y_k = -\left(n_y^m\right)^{-1}\left[n_x^m(x_k - x_a) + n_z^m(z_k - z_a)\right] + y_a$$

(3.22)

Else, if $\left|n_z^m\right| < \frac{1}{\sqrt{3}}$ and $\left|n_y^m\right| < \frac{1}{\sqrt{3}}$

$$y_k = (y_b - y_a)u_k + (y_c - y_a)v_k + y_a$$

$$z_k = (z_b - z_a)u_k + (z_c - z_a)v_k + z_a$$

$$x_k = -\left(n_x^m\right)^{-1}\left[n_y^m(y_k - y_a) + n_z^m(z_k - z_a)\right] + x_a$$

(3.23)

Equations (3.21) - (3.23) are also used to evaluate the Cartesian coordinates of P_e or Q_m, which may be denoted as $\left(x^m, y^m, z^m\right)$, by setting $u_k = \frac{1}{4}$ and $v_k = \frac{1}{2}$. Hence, for element m, $\left(x^m, y^m, z^m\right)$ which is the chosen point inside the element m and which is the only point on the element where displacement or traction is considered (the other points on the element having the same value of displacement or traction as that of this point), is given by (3.24) - (3.26).

If $\left|n_z^m\right| \geq \frac{1}{\sqrt{3}}$

$$x^m = (x_b - x_a)\frac{1}{4} + (x_c - x_a)\frac{1}{2} + x_a$$

$$y^m = (y_b - y_a)\frac{1}{4} + (y_c - y_a)\frac{1}{2} + y_a$$

$$z^m = -\left(n_z^m\right)^{-1}\left[n_x^m\left(x^m - x_a\right) + n_y^m\left(y^m - y_a\right)\right] + z_a$$

(3.24)

Else, if $\left|n_z^m\right| < \frac{1}{\sqrt{3}}$ and $\left|n_y^m\right| \geq \frac{1}{\sqrt{3}}$

$$x^m = (x_b - x_a)\frac{1}{4} + (x_c - x_a)\frac{1}{2} + x_a$$

$$z^m = (z_b - z_a)\frac{1}{4} + (z_c - z_a)\frac{1}{2} + z_a$$

$$y^m = -\left(n_y^m\right)^{-1}\left[n_x^m\left(x^m - x_a\right) + n_z^m\left(z^m - z_a\right)\right] + y_a$$

(3.25)

Else, if $\left|n_z^m\right| < \frac{1}{\sqrt{3}}$ and $\left|n_y^m\right| < \frac{1}{\sqrt{3}}$

$$y^m = (y_b - y_a)\frac{1}{4} + (y_c - y_a)\frac{1}{2} + y_a$$

$$z^m = (z_b - z_a)\frac{1}{4} + (z_c - z_a)\frac{1}{2} + z_a$$

$$x^m = -\left(n_x^m\right)^{-1}\left[n_y^m\left(y^m - y_a\right) + n_z^m\left(z^m - z_a\right)\right] + x_a$$

(3.26)

One can see that (3.16) can now be evaluated if one knows the expressions for the fundamental solutions (i.e., U_{xx}, T_{xx}, U_{xy}, T_{xy}, U_{xz}, T_{xz}, U_{yx}, T_{yx}, U_{yy}, T_{yy}, U_{yz}, T_{yz}, U_{zx}, T_{zx}, U_{zy}, T_{zy}, U_{zz}, T_{zz}). Using (3.2) and (3.3), expressions for the fundamental solutions may be written in the expanded form as given by (3.27) below. In these equations, (x_1, y_1, z_1) denotes the coordinates of the point P_e while (x_2, y_2, z_2) denotes the coordinates of the point (x_k, y_k, z_k).

$$U_{xx} = \frac{C}{r}\left[C_1 + \left(\frac{dr}{dx}\right)^2\right] \qquad U_{yy} = \frac{C}{r}\left[C_1 + \left(\frac{dr}{dy}\right)^2\right]$$

$$U_{zz} = \frac{C}{r}\left[C_1 + \left(\frac{dr}{dz}\right)^2\right]$$

$$U_{xy} = U_{yx} = \left(\frac{C}{r}\right)\left(\frac{dr}{dx}\right)\left(\frac{dr}{dy}\right) \qquad U_{yz} = U_{zy} = \left(\frac{C}{r}\right)\left(\frac{dr}{dy}\right)\left(\frac{dr}{dz}\right)$$

$$U_{zx} = U_{xz} = \left(\frac{C}{r}\right)\left(\frac{dr}{dz}\right)\left(\frac{dr}{dx}\right)$$

$$T_{xx} = \frac{-C_2}{r^2}\left[\left\{C_3 + 3\left(\frac{dr}{dx}\right)^2\right\}\cos\theta\right] \qquad T_{yy} = \frac{-C_2}{r^2}\left[\left\{C_3 + 3\left(\frac{dr}{dy}\right)^2\right\}\cos\theta\right]$$

$$T_{zz} = \frac{-C_2}{r^2}\left[\left\{C_3 + 3\left(\frac{dr}{dz}\right)^2\right\}\cos\theta\right]$$

$$T_{xy} = \frac{-C_2}{r^2}\left[3\left(\frac{dr}{dx}\right)\left(\frac{dr}{dy}\right)\cos\theta - C_3\left(n_y^m \frac{dr}{dx} - n_x^m \frac{dr}{dy}\right)\right]$$

$$T_{yx} = \frac{-C_2}{r^2}\left[3\left(\frac{dr}{dy}\right)\left(\frac{dr}{dx}\right)\cos\theta - C_3\left(n_x^m \frac{dr}{dy} - n_y^m \frac{dr}{dx}\right)\right]$$

$$T_{yz} = \frac{-C_2}{r^2}\left[3\left(\frac{dr}{dy}\right)\left(\frac{dr}{dz}\right)\cos\theta - C_3\left(n_z^m \frac{dr}{dy} - n_y^m \frac{dr}{dz}\right)\right]$$

$$T_{zy} = \frac{-C_2}{r^2}\left[3\left(\frac{dr}{dz}\right)\left(\frac{dr}{dy}\right)\cos\theta - C_3\left(n_y^m\frac{dr}{dz} - n_z^m\frac{dr}{dy}\right)\right]$$

$$T_{zx} = \frac{-C_2}{r^2}\left[3\left(\frac{dr}{dz}\right)\left(\frac{dr}{dx}\right)\cos\theta - C_3\left(n_x^m\frac{dr}{dz} - n_z^m\frac{dr}{dx}\right)\right]$$

$$T_{xz} = \frac{-C_2}{r^2}\left[3\left(\frac{dr}{dx}\right)\left(\frac{dr}{dz}\right)\cos\theta - C_3\left(n_z^m\frac{dr}{dx} - n_x^m\frac{dr}{dz}\right)\right]$$

(3.27)

where $r = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$

$$\frac{dr}{dx} = \frac{(x_2 - x_1)}{r} \qquad \frac{dr}{dy} = \frac{(y_2 - y_1)}{r}$$

$$\frac{dr}{dz} = \frac{(z_2 - z_1)}{r}$$

$$\cos\theta = \frac{1}{r}\left[(x_2 - x_1)n_x^m + (y_2 - y_1)n_y^m + (z_2 - z_1)n_z^m\right]$$

(From equation (3.4))

Other notations have the same meanings as earlier

n_x^m, n_y^m and n_z^m are constant over an element m

n_x^m, n_y^m and n_z^m are different for different elements, in general

One can note that since there are sixteen (x_k, y_k, z_k) for every element m, when one integrates a fundamental solution over an element surface S^m (which contains the point Q_m), for every (x_1, y_1, z_1), there are sixteen different (x_2, y_2, z_2). Further, when the whole code is considered, since the total number of elements equals T, for every (x_1, y_1, z_1), there are $16T$ different (x_2, y_2, z_2); and there are T different (x_1, y_1, z_1) in total.

The code addresses strong singularity by using ‘rigid body modes’ and weak singularity is taken care of by utilizing higher number of integration points over each of the boundary elements. The next paragraph tells a few words about ‘rigid body modes’.

When an integral as encountered in the 3D linear elastostatics without body forces (with constant elements) is strongly singular, the value of the definite integral exists only in the sense of Cauchy Principal Value (CPV). Cauchy Principal Values in this case may be found either by direct evaluation [M. Guiggiani and A. Gigante, 1990; M. Guiggiani, n.d.] or by using ‘rigid body modes’ explained in [Gernot Beer, et al., 2008; C.A. Brebbia and J. Dominguez, n.d.]. The code uses ‘rigid body modes’ or ‘rigid body considerations’ as explained in [Gernot Beer, et al., 2008; C.A. Brebbia and J. Dominguez, n.d.] to evaluate strongly singular integrals.

When good accuracy is required, the code can use *16 by 16* numerical integration (i.e., *256* integration points over each element). Higher number of integration points ensures accurate evaluation of weakly singular integrals. Of course, the code has a provision to use *4 by 4*, *8 by 8*, or *32 by 32* integration points also; location of Gauss points and the corresponding weights for these cases are listed in the code itself; the location of Gauss points and weights are obtained from the web source [http://www.efunda.com, n.d.]. Test runs showed that using *4 by 4* integration does not give very accurate results. Test runs using *8 by 8* integration gave accurate results but the code uses *16 by 16* integration by default just to ensure that one does not get erroneous results just because of inaccurate integration. It was also observed from test runs that there is not much improvement in accuracy by using more than *16 by 16* integration points. Of course, using *8 by 8* integration instead of the default *16 by 16* integration would definitely make the code execute faster.

3.1.2 Illustration and Verification

In the present subsection, the code is tested by using the code to solve four simple problems with known analytical solutions.

All the four problems considered here use the same geometry, although four different boundary conditions give rise to four different problems named as: Problem A, Problem B, Problem C, and Problem D. Each of the four problems is solved for three mesh resolutions named as: Mesh 1, Mesh 2, and Mesh 3. Mesh 1 always consists of

100 boundary elements whereas Mesh 2 always consists of 300 boundary elements and Mesh 3 always consists of 500 boundary elements.

The geometry is a block of 1 mm by 1 mm cross section and 3 mm length. This is the geometry that is used for each of the simulations here, although different boundary conditions together with different mesh resolutions give rise to many simulations. The geometry is constructed in Rhinoceros, and then the geometry is saved as a .stl file, the geometry being described by less than 100 triangles in total. Then one can obtain Mesh 1, Mesh 2, and Mesh 3 by using MeshLab to increase the total number of triangles that describe the geometry to 100, 300, and 500 respectively.

In this subsection, for all the simulations, modulus of elasticity is assumed as 200000 N/mm^2, and the Poisson's ratio is assumed to be equal to 0.33.

Problem A is the problem where one end of a bar is fixed while a load of specified magnitude is applied at the other end, the load being applied as the uniformly distributed load over the whole face which represents the loaded end. The solution (displacement over the whole face, along the direction of the applied load) is sought at the loaded end. The magnitude of the applied load is such that the load produces a displacement of 1 mm at the loaded end, as calculated by the analytical formula.

Problem B is to take a cantilever beam, fix one end of the beam, and apply a load (of specified magnitude) at the other end in the lateral direction, the load being applied as the uniformly distributed load over the whole face which represents the loaded end. The solution (displacement over the whole face, along the direction of the applied load) is sought at the loaded end. The magnitude of the applied load is such that the load produces a displacement of 1 mm (along the direction of the load) at the loaded end, as calculated by the analytical formula.

Problem C is to take a bar, fix one end of the bar, and apply a specified non-zero displacement at the other end. The specified non-zero displacement is uniformly enforced over the whole face. The solution (traction over the whole face, along the direction of the specified non-zero displacement) is sought at the end of the bar where the non-zero displacement is applied. The magnitude of the specified non-zero displacement is equal to 1 mm.

Problem D is to take a cantilever beam, fix one end of the beam, and apply a specified non-zero displacement at the other end in the lateral direction. The specified non-zero displacement is uniformly enforced over the whole face. The solution (traction over the whole face, along the direction of the specified non-zero displacement) is sought at the end of the beam where the non-zero displacement is applied. The magnitude of the specified non-zero displacement is equal to 1 mm.

Whenever Mesh 1 is used, element numbers 6, 7, 12, 16, 25, 26, 35, 36, 37, 38, 40, 52, 63, 64, 96, 97, 98, 99, and 100 are subjected to the boundary condition of zero displacement along all the three axes. Whenever Mesh 1 is used to solve Problem A or Problem B, element numbers 8, 39, and 53 are subjected to non-zero traction. Whenever Mesh 1 is used to solve Problem C or Problem D, element numbers 8, 39, and 53 are subjected to non-zero displacement. Whenever Mesh 1 is used, the rest of the elements are subjected to zero traction.

Whenever Mesh 2 is used, element numbers 18, 19, 23, 38, 39, 46, 71, 72, 80, 123, 124, 125, 126, 127, 128, 129, 141, 168, 169, 170, 171, 172, 173, 174, 175, 176, 200, 201, 202, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, and 293 are subjected to the boundary condition of zero displacement along all the three axes. Whenever Mesh 2 is used to solve Problem A or Problem B, element numbers 2, 20, 21, 22, 24, 47, 48, 130, 131, 132, 133, 177, 178, 179, 180, 181, 203, 204, 294, 295, 296, 297, 298, 299, and 300 are subjected to non-zero traction. Whenever Mesh 2 is used to solve Problem C or Problem D, element numbers 2, 20, 21, 22, 24, 47, 48, 130, 131, 132, 133, 177, 178, 179, 180, 181, 203, 204, 294, 295, 296, 297, 298, 299, and 300 are subjected to non-zero displacement. Whenever Mesh 2 is used, the rest of the elements are subjected to zero traction.

Whenever Mesh 3 is used, element numbers 29, 30, 31, 32, 33, 39, 63, 64, 65, 66, 80, 81, 122, 123, 142, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 248, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 343, 344, 345, 346, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, and 490 are subjected to the boundary condition of zero displacement along all the three axes. Whenever Mesh 3 is used to solve Problem A or Problem B, element numbers 3, 8, 34, 35, 36, 37, 40, 41, 67, 68, 82, 83, 84, 124, 158, 228, 229, 230, 231, 232, 233, 304, 305, 306, 307, 308, 309, 310, 311, 347, 348, 349, 350, 491, 492, 493, 494, 495, 496, 497, 498, 499, and

500 are subjected to non-zero traction. Whenever Mesh 3 is used to solve Problem C or Problem D, element numbers 3, 8, 34, 35, 36, 37, 40, 41, 67, 68, 82, 83, 84, 124, 158, 228, 229, 230, 231, 232, 233, 304, 305, 306, 307, 308, 309, 310, 311, 347, 348, 349, 350, 491, 492, 493, 494, 495, 496, 497, 498, 499, and 500 are subjected to non-zero displacement. Whenever Mesh 3 is used, the rest of the elements are subjected to zero traction.

The analytical solutions for bar and beam are obtained by using formulae from [Warren C. Young and Richard G. Budynas, 2002]. In the case of beams, the said reference informs that the analytical formula is reasonably accurate for 'length to diameter ratio' equal to or more than 3. Hence one can assume that the analytical formulae can provide reasonably accurate results here. The analytical formulae used here are the same as the ones that can be found in any book on the 'mechanics of materials' (also known as 'strength of materials' or 'engineering mechanics') or any design data handbook.

Traction and force are related by $Traction = \frac{Force}{Area}$

All the tractions in all the tables below refer to the 'applied tractions' (not 'reaction tractions'; the sign of the 'reaction tractions' is opposite to the corresponding 'applied tractions').

For all the tables below, 'A' is the analytical solution, 'B' is the solution using this author's code, and 'C' is the solution obtained by using the software from [Yijun Liu, n.d.].

Table 3.1 Solution of Problem A with Mesh 1

Serial Number	Element Number	A *(mm)*	B *(mm)*	C *(mm)*
1	8	1.000	0.911	0.976
2	39	1.000	0.887	0.969
3	53	1.000	0.929	0.975

Table 3.2 Solution of Problem A with Mesh 2

Serial Number	Element Number	A *(mm)*	B *(mm)*	C *(mm)*
1	2	1.000	0.933	0.976
2	20	1.000	0.934	0.971
3	21	1.000	0.946	0.987
4	22	1.000	0.976	0.994
5	24	1.000	0.943	0.988
6	47	1.000	0.937	0.969
7	48	1.000	0.974	0.991
8	130	1.000	0.940	0.988
9	131	1.000	0.954	0.988
10	132	1.000	0.975	0.991
11	133	1.000	0.965	0.989
12	177	1.000	0.936	0.985
13	178	1.000	0.957	0.967
14	179	1.000	0.971	0.988
15	180	1.000	0.975	0.989
16	181	1.000	0.975	0.983
17	203	1.000	0.961	0.989
18	204	1.000	0.975	0.993
19	294	1.000	0.979	0.981
20	295	1.000	0.964	0.986
21	296	1.000	0.962	0.990
22	297	1.000	0.972	0.980
23	298	1.000	0.969	0.990
24	299	1.000	0.976	0.987
25	300	1.000	0.982	0.982

Table 3.3 Solution of Problem A with Mesh 3

Serial Number	Element Number	A *(mm)*	B *(mm)*	C *(mm)*
1	3	1.000	0.961	0.987
2	8	1.000	0.973	0.991
3	34	1.000	0.959	0.981
4	35	1.000	0.964	0.990
5	36	1.000	0.964	0.993
6	37	1.000	0.978	0.994
7	40	1.000	0.957	0.990
8	41	1.000	0.970	0.992
9	67	1.000	0.957	0.979
10	68	1.000	0.975	0.993
11	82	1.000	0.951	0.971
12	83	1.000	0.967	0.990
13	84	1.000	0.969	0.991
14	124	1.000	0.961	0.979
15	158	1.000	0.972	0.993
16	228	1.000	0.953	0.988
17	229	1.000	0.963	0.991
18	230	1.000	0.963	0.991
19	231	1.000	0.978	0.992
20	232	1.000	0.976	0.994
21	233	1.000	0.964	0.988
22	304	1.000	0.962	0.985
23	305	1.000	0.955	0.988
24	306	1.000	0.965	0.989
25	307	1.000	0.961	0.972
26	308	1.000	0.978	0.992
27	309	1.000	0.974	0.991
28	310	1.000	0.962	0.986
29	311	1.000	0.960	0.979
30	347	1.000	0.961	0.990
31	348	1.000	0.965	0.989
32	349	1.000	0.971	0.993
33	350	1.000	0.969	0.990
34	491	1.000	0.964	0.980
35	492	1.000	0.957	0.989
36	493	1.000	0.969	0.988
37	494	1.000	0.966	0.992
38	495	1.000	0.972	0.983
39	496	1.000	0.976	0.994
40	497	1.000	0.971	0.992
41	498	1.000	0.966	0.989
42	499	1.000	0.963	0.987
43	500	1.000	0.964	0.978

Table 3.4 Solution of Problem B with Mesh 1

Serial Number	Element Number	A *(mm)*	B *(mm)*	C *(mm)*
1	8	1.000	0.123	0.164
2	39	1.000	0.125	0.170
3	53	1.000	0.125	0.167

Table 3.5 Solution of Problem B with Mesh 2

Serial Number	Element Number	A *(mm)*	B *(mm)*	C *(mm)*
1	2	1.000	0.220	0.301
2	20	1.000	0.221	0.302
3	21	1.000	0.222	0.303
4	22	1.000	0.239	0.311
5	24	1.000	0.225	0.305
6	47	1.000	0.224	0.299
7	48	1.000	0.241	0.312
8	130	1.000	0.223	0.303
9	131	1.000	0.228	0.306
10	132	1.000	0.237	0.308
11	133	1.000	0.231	0.307
12	177	1.000	0.220	0.304
13	178	1.000	0.231	0.299
14	179	1.000	0.234	0.304
15	180	1.000	0.239	0.312
16	181	1.000	0.240	0.311
17	203	1.000	0.229	0.305
18	204	1.000	0.239	0.311
19	294	1.000	0.239	0.312
20	295	1.000	0.232	0.304
21	296	1.000	0.232	0.308
22	297	1.000	0.236	0.303
23	298	1.000	0.237	0.310
24	299	1.000	0.237	0.309
25	300	1.000	0.240	0.313

Table 3.6 Solution of Problem B with Mesh 3

Serial Number	Element Number	A *(mm)*	B *(mm)*	C *(mm)*
1	3	1.000	0.280	0.363
2	8	1.000	0.295	0.372
3	34	1.000	0.289	0.368
4	35	1.000	0.281	0.368
5	36	1.000	0.282	0.365
6	37	1.000	0.294	0.371
7	40	1.000	0.289	0.370
8	41	1.000	0.292	0.371
9	67	1.000	0.291	0.371
10	68	1.000	0.291	0.367
11	82	1.000	0.279	0.356
12	83	1.000	0.296	0.373
13	84	1.000	0.291	0.370
14	124	1.000	0.293	0.372
15	158	1.000	0.292	0.371
16	228	1.000	0.282	0.363
17	229	1.000	0.283	0.363
18	230	1.000	0.288	0.367
19	231	1.000	0.292	0.367
20	232	1.000	0.295	0.372
21	233	1.000	0.293	0.373
22	304	1.000	0.291	0.372
23	305	1.000	0.280	0.365
24	306	1.000	0.286	0.365
25	307	1.000	0.283	0.356
26	308	1.000	0.292	0.367
27	309	1.000	0.288	0.362
28	310	1.000	0.294	0.373
29	311	1.000	0.295	0.372
30	347	1.000	0.287	0.366
31	348	1.000	0.290	0.370
32	349	1.000	0.295	0.372
33	350	1.000	0.295	0.373
34	491	1.000	0.295	0.373
35	492	1.000	0.279	0.367
36	493	1.000	0.286	0.361
37	494	1.000	0.289	0.368
38	495	1.000	0.288	0.361
39	496	1.000	0.293	0.370
40	497	1.000	0.293	0.369
41	498	1.000	0.295	0.373
42	499	1.000	0.294	0.373
43	500	1.000	0.296	0.374

Table 3.7 Solution of Problem C with Mesh 1

Serial Number	Element Number	A *(N/mm^2)*	B *(N/mm^2)*	C *(N/mm^2)*
1	8	66666.667	69455.491	67709.806
2	39	66666.667	87269.862	67597.361
3	53	66666.667	68126.836	69598.728

Table 3.8 Solution of Problem C with Mesh 2

Serial Number	Element Number	A *(N/mm^2)*	B *(N/mm^2)*	C *(N/mm^2)*
1	2	66666.667	81486.037	72147.765
2	20	66666.667	84995.345	96075.469
3	21	66666.667	60477.860	58340.465
4	22	66666.667	63451.442	65324.353
5	24	66666.667	61538.614	61009.855
6	47	66666.667	89685.834	86463.197
7	48	66666.667	72730.978	75117.094
8	130	66666.667	81725.052	71178.087
9	131	66666.667	60001.908	57705.564
10	132	66666.667	56818.792	58661.374
11	133	66666.667	70193.468	60635.449
12	177	66666.667	77898.163	70736.586
13	178	66666.667	95974.703	102684.039
14	179	66666.667	54412.395	58161.175
15	180	66666.667	70418.616	63709.805
16	181	66666.667	70688.789	84493.907
17	203	66666.667	60214.349	60584.703
18	204	66666.667	59849.910	58948.963
19	294	66666.667	74441.487	85417.435
20	295	66666.667	57029.932	58600.649
21	296	66666.667	58294.055	58062.791
22	297	66666.667	94369.778	89092.484
23	298	66666.667	55930.383	57797.973
24	299	66666.667	59414.068	62179.143
25	300	66666.667	105867.355	88768.029

Table 3.9 Solution of Problem C with Mesh 3

Serial Number	Element Number	A *(N/mm^2)*	B *(N/mm^2)*	C *(N/mm^2)*
1	3	66666.667	80476.356	73159.342
2	8	66666.667	55361.324	58157.622
3	34	66666.667	81488.617	72990.680
4	35	66666.667	88061.514	83577.623
5	36	66666.667	84322.201	58739.816
6	37	66666.667	55216.998	62156.941
7	40	66666.667	100695.230	59235.072
8	41	66666.667	59496.037	57981.005
9	67	66666.667	105300.871	83651.848
10	68	66666.667	56521.454	55767.696
11	82	66666.667	101867.166	98513.931
12	83	66666.667	73946.175	74829.386
13	84	66666.667	60380.866	58330.960
14	124	66666.667	73281.612	79994.163
15	158	66666.667	62786.751	57938.524
16	228	66666.667	81912.796	72839.110
17	229	66666.667	58204.600	59675.048
18	230	66666.667	61334.618	58250.261
19	231	66666.667	51198.805	58036.568
20	232	66666.667	59142.952	61901.167
21	233	66666.667	59131.074	59150.439
22	304	66666.667	63908.099	61983.984
23	305	66666.667	-24162.010	72759.266
24	306	66666.667	59526.511	57062.372
25	307	66666.667	97691.950	101571.154
26	308	66666.667	55190.489	62833.250
27	309	66666.667	57028.667	59152.382
28	310	66666.667	74329.468	64840.899
29	311	66666.667	79957.222	85299.495
30	347	66666.667	61955.035	61226.324
31	348	66666.667	49676.034	58915.288
32	349	66666.667	64664.811	61325.679
33	350	66666.667	57082.719	58612.255
34	491	66666.667	74295.764	85650.272
35	492	66666.667	157395.807	109298.246
36	493	66666.667	59576.169	60274.758
37	494	66666.667	62087.698	58549.440
38	495	66666.667	93969.612	87382.461
39	496	66666.667	53555.652	57365.196
40	497	66666.667	136219.200	98542.285
41	498	66666.667	61211.673	59074.620
42	499	66666.667	64546.506	59417.968
43	500	66666.667	117247.806	89561.493

Table 3.10 Solution of Problem D with Mesh 1

Serial Number	Element Number	A *(N/mm^2)*	B *(N/mm^2)*	C *(N/mm^2)*
1	8	1851.852	16745.249	14038.863
2	39	1851.852	14955.246	8343.157
3	53	1851.852	14647.078	11391.793

Table 3.11 Solution of Problem D with Mesh 2

Serial Number	Element Number	A *(N/mm^2)*	B *(N/mm^2)*	C *(N/mm^2)*
1	2	1851.852	18244.275	13142.154
2	20	1851.852	5455.474	9299.625
3	21	1851.852	11861.911	8381.703
4	22	1851.852	5704.807	4044.563
5	24	1851.852	9784.264	6129.254
6	47	1851.852	17826.030	16801.162
7	48	1851.852	3336.521	2120.893
8	130	1851.852	16568.507	9112.139
9	131	1851.852	10339.188	7333.374
10	132	1851.852	5277.272	6285.796
11	133	1851.852	12325.845	7992.502
12	177	1851.852	2711.764	5034.345
13	178	1851.852	23381.354	19417.762
14	179	1851.852	9049.983	10662.799
15	180	1851.852	1990.279	3893.684
16	181	1851.852	-2544.444	4636.885
17	203	1851.852	9552.691	8184.371
18	204	1851.852	4154.832	4725.087
19	294	1851.852	14116.890	11219.491
20	295	1851.852	10504.130	10891.369
21	296	1851.852	6297.779	5542.622
22	297	1851.852	7255.440	9279.251
23	298	1851.852	3294.441	4792.101
24	299	1851.852	5117.793	7314.132
25	300	1851.852	6862.872	9318.893

Table 3.12 Solution of Problem D with Mesh 3

Serial Number	Element Number	A *(N/mm^2)*	B *(N/mm^2)*	C *(N/mm^2)*
1	3	1851.852	15222.115	11018.318
2	8	1851.852	5130.232	4799.061
3	34	1851.852	19880.347	12566.685
4	35	1851.852	4517.181	9191.471
5	36	1851.852	13686.393	6232.052
6	37	1851.852	3222.669	3373.365
7	40	1851.852	18120.214	4286.694
8	41	1851.852	6405.182	5306.015
9	67	1851.852	32838.165	18274.278
10	68	1851.852	7280.242	6738.656
11	82	1851.852	22682.031	19879.622
12	83	1851.852	3799.621	1851.409
13	84	1851.852	6727.599	5156.966
14	124	1851.852	15597.445	11754.625
15	158	1851.852	7680.305	5398.902
16	228	1851.852	14570.145	7367.858
17	229	1851.852	11075.544	11323.280
18	230	1851.852	6265.970	6833.082
19	231	1851.852	4928.453	6599.296
20	232	1851.852	3452.823	3940.951
21	233	1851.852	5767.174	4582.045
22	304	1851.852	8684.409	7425.243
23	305	1851.852	-32724.628	3707.784
24	306	1851.852	9587.585	8351.468
25	307	1851.852	23976.227	19357.735
26	308	1851.852	3739.101	4301.970
27	309	1851.852	11322.100	11805.035
28	310	1851.852	4796.674	3331.125
29	311	1851.852	6857.137	3845.101
30	347	1851.852	9412.559	7351.534
31	348	1851.852	4891.820	5378.983
32	349	1851.852	4670.369	3630.269
33	350	1851.852	5501.112	4487.105
34	491	1851.852	16602.155	12500.110
35	492	1851.852	29371.086	11074.415
36	493	1851.852	11947.036	12320.136
37	494	1851.852	7431.994	5579.314
38	495	1851.852	10873.372	9543.230
39	496	1851.852	5126.721	6020.652
40	497	1851.852	5006.136	1302.501
41	498	1851.852	5701.605	4648.435
42	499	1851.852	5770.083	4548.775
43	500	1851.852	-1107.964	8370.779

From the tables above, one can see that as the mesh is refined, general trend is that the solutions obtained by using this author's code approach the corresponding analytical solutions. Further, the solutions obtained by using this author's code are in good agreement with the corresponding solutions obtained by using the software from [Yijun Liu, n.d.]. Thus one can infer that the present code (i.e., this author's code) has performed satisfactorily.

3.2 Description of Hardware and Software Utilized

In the present chapter, simulations are carried out on two hardware platforms. The first hardware platform is a desktop computer, and the second hardware platform is a computer cluster.

Coming to hardware and software that are used when simulations are carried out on a desktop, MATLAB codes (with or without GPU) are run on a desktop computer (*Intel(R) Xeon(R) CPU E5405 @ 2.00 GHz* (*8* cores), *8 GB* RAM, Mainboard: *Intel D5400XS*, Chipset: *Intel 5400B*, *Windows XP Professional x64 Edition*, SSD: *Corsair CSSD-F60GB2*, *MATLAB2011b* (*32* bit version), GPU: *NVIDIA Quadro 4000 (Driver Version 6.14.12.9573)*). Apart from the solid-state drive (SSD), the simulations were tried out using the ordinary hard disk also. The *64* bit version of *MATLAB2011b* was also tried out. Since it was found from the simulation results that the SSD is about *1.5* times faster when compared to the conventional hard disk and the *32* bit MATLAB is about *10* times faster when compared to the *64* bit MATLAB, it is decided to use the SSD and the *32* bit MATLAB.

Coming to hardware that is used when simulations are carried out on a cluster, a computer cluster consisting of *17* nodes is used. The cluster consists of *9* nodes with *32* cores each (*2.4 GHz AMD Opteron 6136* processor, *64 GB* RAM) and *8* nodes with *64* cores each (*2.2 GHz AMD Opteron 6274* processor, *128 GB* RAM). A *500 GB SATA HDD (3 Gbps)* is used for the operating system and system software. An Infiniband Card *(MHQH19B-XTR)* is used for Message Passing Interface (MPI) communication, and Dual-port Gigabit Ethernet Connectivity is used for enabling logins and Network File System (NFS). Coming to software, *Intel Composer XE (Version: 2011.5.220)* which includes *Intel Fortran compiler* together with *Intel Math Kernel Library (Intel MKL)* is used with *MVAPICH2 (Version: 1.8-r5423)*. *CentOS 6.2* (*Linux x86_64* Platform) is the operating system, and the batch scheduler software

Torque is used for job scheduling and load balancing. Whenever *Torque* is required, *mpiexec (Release 0.84)* from Ohio Supercomputer Center (OSC) is used instead of the usual command *'mpirun'*, in the job script. Although the cluster has *800* processors in total, only *256* cores are used in the present work. This is because the author's organization does not allow any individual to use more than *256* processors at any given point of time in the concerned computer cluster; this is to avoid a single user utilizing all the available computing resources which may cause problems to other users of the computer cluster. From the results to be presented later in this chapter, one can also see that there may not be a need to go for more number of processors since that is not likely to lead to any speed up.

Whenever a BEM code is needed, present chapter makes use of the BEM code explained in the last section, which is freely downloadable from [Kirana Kumara P, 2014a]. The source [Kirana Kumara P, 2014a] provides the code in three versions: (i) A MATLAB code for solving three dimensional linear elastostatic problems using constant boundary elements while ignoring body forces (ii) A Fortran translation of the MATLAB code (iii) A parallelized version of the Fortran code. In the present chapter, whenever a simulation needs to be run on a desktop computer (with or without using a GPU), a MATLAB version of the code is used. Whenever a simulation needs to be run on a computer cluster, a Fortran version of the code is used.

3.3 Studies on the Speed of the Boundary Element Method

The necessity and the novelty and the significance of conducting a study on the speed of BEM, as applied to the real-time computational simulation of biological organs, is already noted down in the first chapter of the present thesis. This section conducts a study of speed of BEM by actually running a BEM code on a computer or a computer cluster. Although tools (e.g., Warwick Performance Prediction (WARPP) simulator) that can predict the time required to run a particular computer program without actually running the program are available, there is advantage in using these tools only if the target hardware is not accessible to the user or if the user cannot run all the desired simulations on the targeted hardware for some reason (e.g., a single execution taking too much time). The execution times predicted by these tools are just estimates which may or may not always be reliable. Further, when it comes to the real-time simulation of biological organs, all the sources in the literature get an estimate of the time needed to complete a simulation by actually running the simulations on a computer system. One can also note that by studying references like [Al Aho and Jeff Ullman, 1992; Hammond Simon D., et al., 2009; Stephen A. Jarvis, et al., 2006], one can come to the conclusion that it is not possible to obtain a reliable estimate of the execution time of a given program without actually running the program at least once (for a specified input) and/or running benchmark tests that specifically aim to characterize the given hardware.

Many of the subsections in this section utilize a sample problem. The sample problem is about taking up a *4 mm by 4 mm by 4 mm* cube, and completely fixing one face of the cube, and applying traction of *4 N/mm*^{2} in the y-direction over the whole of the opposite face. Each face of the cube is discretized into sixteen boundary elements. Young's modulus is assumed to be equal to *200000 N/mm*^{2}, and Poisson's ratio is assumed to be equal to *0.33*; aim of the simulation is to obtain the displacements for the elements that are subjected to known tractions and also to obtain the tractions for the elements that are subjected to known displacements, using constant boundary elements, and using linear elastostatic assumption and ignoring body forces. To demonstrate the speed that can be achieved using BEM, one needs to solve a problem using BEM, and the sample problem is of help here. Whenever the phrase 'sample problem' appears in the present chapter, the phrase refers to this problem only.

The present section uses just the sample problem (explained above) to demonstrate the speed of BEM, and does not mention anything about the speed of BEM when simulations are carried out on complicated geometry like biological organs. One can note that the speed of a simulation here depends solely on the total number of elements and boundary conditions, and the speed does not have any relevance to the actual geometry; hence there is no need to carry out simulations on different geometry just to measure how fast (or slow) the simulations are; in fact, trying out simulations on different geometry could be of help when one is concerned about accuracy but one can also note that the next section of the present chapter uses a complicated geometry (i.e., a liver), not just a cantilever beam, to demonstrate the accuracy of BEM.

The following subsections give information about speeds that can be achieved using different hardware and software; results indicate whether or not a particular hardware-software combination can offer real-time performance for simulations considered in a particular subsection.

3.3.1 Solving the Sample Problem (without Manual Parallelization)

In this subsection, no GPU is used. No manual parallelization is attempted. Of course, the automatic parallelization available in MATLAB is utilized.

Time needed to solve the sample problem is found using the MATLAB commands ‘tic’ and ‘toc’. It was found that it took *2.103 s*, *2.146 s*, and *2.092 s* respectively, during three trials, to solve the sample problem using the default option of using all *8* cores (i.e., using the default fully automatic parallelization available in MATLAB) in the desktop. Next, the problem is solved using a single core in the desktop, and the time needed to solve the problem in this case was found to be *2.101 s*, *2.090 s*, and *2.101 s* respectively, during three successive trials. One can see that it takes more time to solve the problem on *8* cores, when compared to the time needed to solve the problem on a single core. Hence, the automatic parallelization offered by MATLAB is not of use for this problem (in fact, in this case, performance offered by the automatically parallelized code is worse when compared to the performance offered by the sequential code), and one can also observe that the code takes about the same time to run on the *8* cores as it takes to run on a single core. One can also see that the simulation is far from being real-time.

A small note on the speed of MATLAB in general, now. There is a general opinion that MATLAB is slower when compared to ‘lower level’ programming languages like Fortran, C, C++; in fact, there have been many discussions in online forums on topics like “Speed of MATLAB versus speed of Fortran” etc. Upon going through these forums, one comes across varying opinions like “Present day compilers for high level languages such as FORTRAN are so good that the codes written in high level languages are almost as fast as the same codes written in an assembly language, at least when the programmer is not extremely skilled”, “MATLAB used to be about *100* times slower when compared to languages like FORTRAN; but once MATLAB started using the modern JIT compiler, MATLAB is about *10* times slower”, “MATLAB has many built-in functions for scientific applications and the functions are so optimized for speed that it is difficult to write the same functions oneself, in languages like FORTRAN, to achieve the same speed”, “If the programmer is not skilled, languages like FORTRAN could be slower when compared to MATLAB", or "Whether Fortran is faster when compared to MATLAB is highly dependent on the problem in hand as well as the skill of the programmer". In the present work, instead of assuming that the MATLAB is faster when compared to Fortran or otherwise, codes are written both in MATLAB and Fortran and whether Fortran is faster is decided based on the results from the actual runs; in fact, from the results presented (or to be presented) in this section, one can conclude that Fortran is significantly faster when compared to MATLAB; further, a Fortran code can be parallelized and run on a computer cluster whereas it is not easy to find a MATLAB version that can run on a cluster, and even if a MATLAB version that can run on a cluster is found, that version has its own limitations. Of course, people make their MATLAB codes faster by making use of 'System Objects', 'MATLAB Coder', 'MEX functions' etc. (e.g., ["Simulation Acceleration using System Objects, MATLAB Coder and Parallel Computing Toolbox", n.d.]), but the present work aims to achieve the real-time performance without making use of these specialized approaches; also, all simulations cannot be translated to these approaches, and one can note that the GPU implementations of these specialized features are not available often, and further, it may not be possible to achieve the real-time performance with MATLAB even after employing these specialized techniques.

3.3.2 Solving the Sample Problem (with Manual Parallelization)

Since it is found from the previous subsection that the real-time performance cannot be obtained through either a sequential or an automatically parallelized MATLAB code that solves the sample problem, attempt is made in this subsection to manually parallelize the code.

Now, only a portion of the MATLAB code is parallelized; idea is that if real-time performance can be obtained for this portion of the code, manual parallelization can be attempted for a larger portion of the code; and of course, there is no need to attempt to parallelize the whole MATLAB code if one cannot obtain the real-time performance even for a portion of the whole code. Manual parallelization is attempted only for the 'for' loop that calculates the components of the unit normal to the element faces; this is because the concerned 'for' loop is "embarrassingly parallel", and also because manual parallelization is as easy as just replacing 'for' with 'parfor'.

One can note here that “embarrassingly parallel” is a terminology used by the High Performance Computing (HPC) community to indicate that the program to be parallelized is highly scalable. Ideally, the time required to solve an “embarrassingly parallel” problem reduces linearly as the number of processors increase (ignoring the time needed for inter-processor communications, and the linear scalability is possible only up to certain number of processors).

The time taken to execute the loop after just replacing ‘for’ with ‘parfor’ (without initializing ‘matlabpool’) is *0.102 s*, *0.102 s*, and *0.102 s*, for the three trials considered; here, the code runs on the ‘client’ only, not on the ‘MATLAB workers’.

Next, ‘matlabpool’ is used to initialize ‘matlabpool’, and ‘matlabpool close’ is used to close ‘matlabpool’. Again, ‘for’ is replaced with ‘parfor’. The time taken to execute the concerned ‘for’ loop is *13.680 s*, *13.667 s*, and *11.533 s*, for the three trials considered; times include the time taken to initialize and close ‘matlabpool’. If one ignores the time taken to initialize and close ‘matlabpool’, the time taken to execute the concerned ‘for’ loop is *0.324 s*, *0.322 s*, and *0.330 s*, for the three trials. Hence one can see that initializing and closing ‘matlabpool’ takes a lot of time. All the simulations mentioned in this paragraph use the default *8* ‘workers’ (since there are *8* cores in the desktop); ‘workers’ are also known as ‘labs’.

Now, simulations exactly similar to the ones in the last paragraph but only with *1* ‘worker’ are carried out. The time taken to execute the same ‘for’ loop, excluding the time taken for executing ‘matlabpool’ and ‘matlabpool close’, is *0.263 s*, *0.264 s*, and *0.265 s*, for three trials.

Now, the time taken for the same simulation as the one in the last paragraph but with *2* ‘workers’ is found to be *0.277 s*, *0.274 s*, and *0.275 s*, for three trials. The time taken for the same simulation as the one in the last paragraph but with *4* ‘workers’ is found to be *0.290 s*, *0.290 s*, and *0.289 s*, for three trials.

One can observe that as the number of ‘labs’ increase, simulation becomes slower in this case. One can also observe that none of the simulations mentioned in the present subsection could be completed in real-time. One can also observe that if one does not substitute ‘parfor’ for ‘for’, and uses the default automatic parallelization of MATLAB, the time taken to execute the ‘for’ loop is just *0.032 s*, *0.032 s*, and

0.032 s, for the three trials, which makes the simulation a real-time one; the default option of using all the available *8* cores is used here. Instead of using the default option of using all the available cores, if ‘maxNumCompThread’ is used to restrict the number of cores to be used, following is the time needed to execute the portion of the MATLAB code within and including the concerned ‘for’ loop, without substituting ‘parfor’ for ‘for’: *0.032 s*, *0.032 s*, and *0.032 s* for the three trials if only one core is used; *0.032 s*, *0.032 s*, and *0.032 s* for the three trials if two cores are used; *0.033 s*, *0.032 s*, and *0.032 s* for the three trials if four cores are used; *0.032 s*, *0.032 s*, and *0.032 s* for the three trials if eight cores are used.

From the results presented in this subsection, at least for the problem considered in this subsection, there is no use in manually parallelizing the MATLAB code using ‘parfor’. One should also note that not all statements that can be put inside a ‘for’ loop can be put inside a ‘parfor’ loop. Of course, one cannot rule out the possibility of a future version of MATLAB providing a better implementation of ‘parfor’.

3.3.3 Solving the Sample Problem on a GPU

Since the previous two subsections are not successful in obtaining real-time performance, there is a need to run the sample problem on the GPU to see whether one can achieve real-time performance.

GPU computing features available in MATLAB depend on the features available in CUDA and GPU drivers, which in turn depend on the features supported by the GPU hardware. One has to note that only a small subset of MATLAB functions can be run on GPUs using the MATLAB Parallel Computing Toolbox. One can observe that newer versions of MATLAB (and the Parallel Computing Toolbox) have better support for GPUs, and one can definitely hope to see the future versions of MATLAB enabling more and more MATLAB functions to be run on GPUs using Parallel Computing Toolbox. As of now, a lot more needs to be done from the software developers to make many MATLAB functions to readily run on GPUs. Not only functions, but some MATLAB scripts cannot readily be ported to GPUs. Of course, programs written in lower level languages like Fortran and C may also be modified to run on GPUs; however, this task may not be easy always, and sometimes, porting a code to a GPU could itself be a research problem. Whenever a GPU is used, one needs to transfer the variables from CPU to GPU, and after performing computations on the GPU, results have to be transferred from GPU to CPU; and these data transfers are time consuming. Although author is aware of these limitations of GPU computing, present subsection makes an attempt to run the MATLAB code (that solves the sample problem) on a GPU with the intention of obtaining real-time performance, using the MATLAB Parallel Computing Toolbox.

Just like in the last subsection (which tried to parallelize the MATLAB code (that solves the sample problem) using ‘parfor’), only a portion of the MATLAB code that solves the sample problem is parallelized in this subsection first; idea is that if real-time performance cannot be obtained for even a portion (or a part) of the code, there is no need to attempt to parallelize the whole MATLAB code. Hence the parallelization is now attempted only for the ‘for’ loop that calculates the components of the unit normal to the element faces. Whenever the parallelized code is run on the GPU, there is a need to transfer the variables from CPU to GPU.

With prior initialization of the GPU arrays using ‘parallel.gpu.GPUArray.zeros’, time needed to execute the MATLAB program from the beginning of the program to the end of the concerned ‘for’ loop is found to be *1.249 s*, *1.247 s*, and *1.251 s*, for the three trials. If GPU arrays are not initialized using ‘parallel.gpu.GPUArray.zeros’, time needed to execute the MATLAB program from the beginning of the program to the end of the concerned ‘for’ loop is found to be *1.789 s*, *1.807 s*, and *1.805 s* for three trials.

The simulation that is exactly same as the one carried out in the last paragraph but carried out on the CPU alone (without using the GPU at all) takes *0.172 s*, *0.173 s*, and *0.173 s* (for three trials) to complete, if run on a single core of the desktop; but if all the 8 cores of the desktop are utilized, the same simulation takes *0.173 s*, *0.173 s*, and *0.173 s*, for three trials.

Now, with prior initialization of the GPU arrays using ‘parallel.gpu.GPUArray.zeros’, time needed to execute on the GPU the concerned ‘for’ loop alone is *1.252 s*, *1.251 s*, and *1.253 s* for three trials. If the time needed to execute ‘parallel.gpu.GPUArray.zeros’ is also taken into account, time needed to execute on the GPU the concerned ‘for’ loop alone plus the time needed to execute ‘parallel.gpu.GPUArray.zeros’ is *1.271 s*, *1.258 s*, and *1.266 s* for three trials. If GPU arrays are not initialized using ‘parallel.gpu.GPUArray.zeros’, time needed to execute on the GPU the concerned 'for' loop alone is *1.728 s*, *1.708 s*, and *1.729 s* for three trials.

One can see that none of the simulations that are tried out in the present subsection turned out to be real-time. As far as the problems considered in this subsection are concerned, there is no advantage in using a GPU instead of a CPU. Since it is found that not even a portion of the sample problem could be executed in real-time on a GPU, there is no point in trying to run the whole of the sample problem on a GPU in real-time.

3.3.4 Using the GPU to Solve the System of Equations only

Results presented in the previous subsections of the present section show that it is not possible to solve the sample problem in real-time by making use of a desktop computer loaded with MATLAB, even if a GPU together with the MATLAB Parallel Computing Toolbox is made use of. One can note that the sample problem is a small-sized problem and if it is not possible to solve this small-sized problem in real-time, it would not be possible to solve a larger sized problem in real-time. It is of use sometimes, even if one manages to solve only a part of the sample problem in real-time. For example, if one is happy with linear elastostatics and if there is no change in the geometry during a simulation, the 'characteristic matrix' and its inverse can be precomputed, and the problem then reduces to just a matrix multiplication; and in this case, if one can manage to complete the matrix multiplication in real-time, it could be as good as solving the whole problem in real-time. Hence there is a need to see whether particular portions of the sample problem can be executed in real-time, either by making use of a GPU or not.

Hence, in this subsection, attempt is made to obtain the real-time performance while solving a system of linear simultaneous algebraic equations; also, in the next subsection, attempt is made to obtain the real-time performance while multiplying a matrix by a vector. One can note that solving a system of equations is a part of solving the sample problem. The task that is carried out in the present subsection is taken up just out of curiosity (or academic interest) since, unlike the task that is concerned with just multiplying a matrix by a vector, present author cannot think of any use in achieving the real-time performance just for the portion of the MATLAB code that just solves the system of simultaneous equations.

One can see that the time needed to multiply a matrix by a vector depends on the size of the matrix, not on the actual values of the elements of the matrix. Similarly, at least when not using an iterative solver, the time needed to solve a system of equations is mainly dependent on the size of the system of equations only. Hence, in the present work, instead of solving the actual system of equations obtained through BEM, a dummy system of equations is generated and solved. Using dummy system of equations is useful here because, while the sample problem always generates a system of equations that has *288* simultaneous equations, different problem sizes can easily be tried out if dummy system of equations are utilized. Results show that when the size of the system of equations is equal to *500*, the system of equations can be solved in real-time, either by making use of the GPU or otherwise; here, solving the system of equations on the GPU, and solving the system of equations on the CPU, both take almost the same amount of time. When the size of the system of equations is equal to around *1000*, the system of equations can be solved in real-time only if the GPU is made use of. When the size of the system of equations is equal to *1500*, the system of equations cannot be solved in real-time whether a GPU is used or not.

3.3.5 Using the GPU to Multiply a Matrix by a Vector

Motivation for the present attempt to obtain the real-time performance while multiplying a matrix by a vector is explained in the previous subsection. The same arguments used in the last subsection to make use of dummy problems are applicable to the present subsection too.

From the results, when the size of the vector is *16000*, the simulation on the CPU takes *0.418 s* while the simulation that uses the GPU takes just *0.030 s* (i.e., *14* times faster). The thirty computations per second desired by real-time graphics amounts to an allowable time of up to *0.033 s* per computation, and one can note that this targeted speed for this simulation cannot be achieved if the GPU is not made use of here.

As already mentioned in the previous subsection, if one is happy with linear elastostatics and if there is no change in the geometry during a simulation, the ‘characteristic matrix’ and its inverse can be precomputed and hence the problem reduces to just that of multiplying a matrix with a vector. Hence in this case, if one can manage to multiply a matrix with a vector in real-time, it is as good as solving the whole problem in real-time. One cannot find any source in the literature that uses a

GPU to achieve the real-time performance while utilizing the BEM in this manner. Further, as already noted, it is not possible to achieve the real-time performance for a model with as high a number of degrees of freedom as *16000*, if a GPU is not utilized for multiplying a matrix with a vector.

3.3.6 Running the Sample Problem on a Computer Cluster

This subsection is a very important part of the present chapter. Motivation for trying to obtain the real-time performance while solving the whole of the sample problem on a cluster has already been explained in the beginning of the present chapter; also, the last section (i.e., 'Summary') mentions some points related to the present subsection.

In this subsection, whole of the sample problem (not a part) is run on a cluster. The sample problem is solved using *1*, *4*, *16*, *64*, and *256* processors, and the time needed to solve the sample problem is noted down for each of the cases. One can note that the code first calculates the 'characteristic matrix' and the 'right hand side', and then solves the system of equations; the 'time' needed to solve the problem (as noted down in the tables included in this subsection) always includes both the time needed to calculate the 'characteristic matrix' and the 'right hand side' and the time needed to solve the system of equations. Part of the code that calculates the 'characteristic matrix' and the 'right hand side' is separated from the part of the code that solves the system of equations by using the BLACS routine 'blacs_barrier'; hence, in the code, the task of solving the system of equations begins only just after the whole of both the 'characteristic matrix' and the 'right hand side' are assembled (i.e., solution of the system of equations can begin only after each of the processes in the process grid complete their part of the work in calculating the 'characteristic matrix' and the 'right hand side'). Parallelization of the part of the code that calculates the 'characteristic matrix' and the 'right hand side' uses the 'Block Distribution' whereas the part of the code that solves the system of equations uses the 'Block-Cyclic Distribution'. When *4* processors are used, a *2 by 2* process grid is used; when *16* processors are used, a *4 by 4* process grid is used; when *64* processors are used, an *8 by 8* process grid is used; and when *256* processors are used, a *16 by 16* process grid is used. Parallel version of the Fortran code that solves the sample problem on the cluster uses ScaLAPACK to solve the system of equations while the sequential version of the Fortran code that solves the sample problem on a single core of the cluster uses LAPACK while solving the system of equations. The parallel version of the Fortran code uses BLACS and MPI also.

One can see that the present simulation uses a 'characteristic matrix' of *288 by 288* size. One can note that when the parallelized Fortran code is run on a single processor, the 'characteristic matrix' is Block-Cyclically distributed by using a block size of *288 by 288*. When the parallelized Fortran code is run on *4* processors, block sizes of *144*, *128*, *64*, *32*, and *1* are tried out. Similarly, when the code is run on *16* processors, block sizes of *64*, *32*, and *1* are tried out; when the code is run on *64* processors, block sizes of *32* and *1* are tried out; and when the code is run on *256* processors, a block size of *16* is used. Idea behind choosing these block sizes is that, from literature, codes execute slower if the block sizes are too small (like *1*), and again, if the block sizes are too large, some of the processors may not get any data to process and hence the very use of higher number of processors to achieve better parallelism may lose its purpose; also, some references recommend using a block size of *32*, *64*, or even *128*, for good performance. Hence the block sizes for different cases that use different number of processors are chosen such that all the processors get some data to process, and different block sizes are tried out for the same cases to find out how block sizes affect the speed.

Now, the time taken for the code to execute on different number of processors, for different block sizes, is listed in Table 3.13; results are presented for four trials, and the averages of the four trials are also listed. Table 3.14 gives the average time taken by the code to execute itself on different total number of processors, taking into account all the block sizes considered for the case, and taking into account all the trials of a particular simulation also.

Table 3.13 Solution Time in Seconds

	First Run	Second Run	Third Run	Fourth Run	Average
Serial (with '-mkl=sequential')	0.170	0.246	0.165	0.134	0.179
Threaded (with '-mkl')	0.364	0.335	0.330	0.246	0.319
Parallel (1 Process, Block Size=288)	0.340	0.230	0.149	0.182	0.225
Parallel (4 Processes, Block Size=144)	0.226	0.069	0.136	0.053	0.121
Parallel (4 Processes, Block Size=128)	0.219	0.116	0.099	0.106	0.135
Parallel (4 Processes, Block Size=64)	0.406	0.497	0.212	0.460	0.394
Parallel (4 Processes, Block Size=32)	0.221	0.187	0.255	0.239	0.225
Parallel (4 Processes, Block Size=1)	0.532	0.441	0.344	0.527	0.461
Parallel (16 Processes, Block Size=64)	0.055	0.067	0.063	0.070	0.064
Parallel (16 Processes, Block Size=32)	0.065	0.057	0.050	0.042	0.054
Parallel (16 Processes, Block Size=1)	0.053	0.049	0.058	0.064	0.056
Parallel (64 Processes, Block Size=32)	1.008	1.229	0.750	0.078	0.766
Parallel (64 Processes, Block Size=1)	1.740	2.041	2.400	1.171	1.838
Parallel(256 Processes,Block Size=16)	2.067	1.167	1.321	1.460	1.504

Table 3.14 Average Solution Time in Seconds, Considering all the Runs and all the Block Sizes that are Considered

Serial (with '-mkl=sequential')	0.179
Threaded (with '-mkl')	0.319
Parallel (1 process)	0.225
Parallel (4 processes)	0.267
Parallel (16 processes)	0.058
Parallel (64 processes)	1.302
Parallel (256 processes)	1.504

From the results presented in Table 3.13 and Table 3.14, for the sample problem (i.e., the problem considered here), block sizes do not play any significant and meaningful role in the overall sense. On an average, the speediest performance is obtained when *16* processors together with a block size of *32* are used; for this case, the simulation took *0.054 s* to complete (this corresponds to about *19* computations per second). One can also observe that the fastest performance recorded in the tables corresponds to the ‘Fourth Run’ corresponding to the case when *16* processors together with a block size of *32* are used; for this instance, the simulation took *0.042 s* to complete (and this corresponds to about *24* computations per second). One can observe that if the average is taken for all the runs and all the block sizes together, on an average, speediest performance is obtained when *16* processors are used (solution time = *0.058 s*; this corresponds to about *17* computations per second).

One might need to keep a few points in mind while studying the results presented in this subsection. The cluster used here is made up of heterogeneous nodes. The nodes with *32* cores have *2.4 GHz* processors whereas the nodes with *64* cores have *2.2 GHz* processors. Inter-node communications could be slower when compared to intra-node communications. In the present subsection, simulations using *1*, *4*, and *16* processors are usually run on a node having *32* cores, and the simulation that uses *64* processors is run on a node having *64* cores; the simulation requiring *256* processors is run using four nodes with *64* cores each. Of course, for each and every simulation mentioned in this subsection, each of the processors runs one and only one process.

One can see that one is not likely to achieve faster performance by going for higher number of processors in this cluster.

One can note that because the whole of the sample problem is made to run on a cluster here, results presented here are useful when one wishes to learn about the performance of BEM when the ‘characteristic matrix’ changes during simulations (e.g., during the simulation of cutting, during the simulations that use nonlinear BEM). One can also note that the BEM used here is the ‘standard’ BEM, not any specialized version of BEM (e.g., the Fast Multipole Boundary Element Method which uses the fast multipole method to accelerate the solution of the system of equations).

3.3.7 Possibility of Simulating Nonlinear Behaviour in Real-Time using BEM

Many a times, realistic description of nonlinear behaviour of biological organs (like liver) requires the use of hyperelastic material models (e.g., Mooney-Rivlin model, Neo-Hookean model). Solving one hyperelastic problem is equivalent to solving many linear problems. Although the total number of iterations (i.e., linear solutions or Newton iterations) required to solve a nonlinear problem within the specified tolerance (for the error) cannot be known beforehand, one can get an idea of the total number of iterations needed to solve a nonlinear problem by referring to the literature that deals with the solution of similar type of problems. For example, [Mark Adams and James W. Demmel, 1999] includes the task of solving a hyperelastic problem, and by going through [Mark Adams and James W. Demmel, 1999], the total number of Newton iterations needed to solve the problem is always between *62* to *70*. By making use of software like ANSYS which have in-built hyperelastic material models, one can solve similar hyperelastic problems to get an idea of the total number of iterations needed to solve such problems. Thus by solving dummy hyperelastic problems on ANSYS, and also by referring to the literature dealing with the solution of hyperelastic problems, one can see that solving a hyperelastic problem takes about *5*, *10*, *20*, *50* or even *80* Newton iterations, while solving different nonlinear problems; it is also observed by the present author that more than *100* linear solutions are rarely needed to solve a hyperelastic problem, although this conclusion is reached just by observing a limited number of examples. Hence it is reasonable to assume that a hyperelastic problem can be solved in real-time if the corresponding linear problem can be solved in real-time *100* times.

But looking at the results presented in the previous subsections of this section, it may be difficult to obtain the real-time performance with a hyperelastic material model.

3.4 Real-time Simulation of Biological Organs on a Computer Cluster

From the previous section, a ‘sample problem’ which uses a total of *96* triangular boundary elements to discretize the geometry can be solved in real-time using a computer cluster, using this author’s BEM code. It is easy to see that, as far as this work is concerned, the speed hardly depends on geometry, boundary conditions, or material properties, but mainly depends on the total number of elements used to describe the geometry only. Hence in this section, each of the biological organs considered are discretized by *96* triangular boundary elements, so that it should be possible to achieve the real-time performance for each of the simulations.

In this section, simulations are carried out on biological organs, a human liver and the left and right kidneys of the Visible Human male in particular. The results obtained by using the BEM codes developed by this author are compared with the results obtained by using the BEM software available from [Yijun Liu, n.d.]. One can note that source code is not available for the software available from [Yijun Liu, n.d.], although the software can be downloaded for free. Further, the software can be used only for the purposes of education, research and further development. The software can run only on the Microsoft Windows operating system (not on Linux). It is noteworthy to mention that source code is available for the program written by this author, and the program can be used for commercial purposes too. A slight difference between the results obtained using this author's code with the results obtained using the program from [Yijun Liu, n.d.] is expected since [Yijun Liu, n.d.] does not provide the implementation details of their software. Hence it is entirely possible that [Yijun Liu, n.d.] handles strong and weak singularities in a different way, and uses a different type of numerical integration scheme when compared to this author's code, which could result in results which are slightly different from those obtained using this author's code.

This section does not compare the execution speed of this author's code with the execution speed of the software available from [Yijun Liu, n.d.]. This is because it does not make sense to compare the speed of the parallelized version of this author's code with the speed of the sequential program available from [Yijun Liu, n.d.]. However, it is important to note that while it is possible to achieve the real-time performance with this author's code, the program from [Yijun Liu, n.d.] needs several seconds to complete the same simulations.

The geometry of the left kidney and the right kidney of the Visible Human male obtained in Chapter 2 is used for the purpose of this chapter, but here both the geometry are represented by just *96* surface triangles (instead of the higher number of triangles in Chapter 2) to enable the simulations to be carried out in real-time. As for the geometry of human liver, the geometry available from ["STLA Files - ASCII stereolithography files", n.d.] is used, but again, the geometry here is represented by just *96* surface triangles (instead of *38142* triangles in ["STLA Files - ASCII stereolithography files", n.d.]).

The geometry of the pig liver, obtained in Chaptet 2 has not been used in this chapter for the simulations. One can see that that particular liver is very thin. It is a well known fact that the BEM is not recommended for the purpose of analyzing thin structures. In fact, this author has still carried out simulations using that particular geometry, but the results are not presented here because it was found that the results obtained were not satisfactory. Of course, it should be possible to obtain accurate solutions if a larger number of elements are used (much more than *96* elements). This is because it is a well known fact that the solutions obtained using constant boundary elements always converge as the total number of elements is increased [W. T. Ang, 2007]. However, real-time performance would be lost if a large number of elements is used.

Values of Young's modulus and Poisson's ratio are taken from [C. Monserrat, et al., 2001]. The reference reports the value of Young's modulus to be equal to *150* N/mm^2 while the value of Poisson's ratio to be equal to *0.4* for a pig liver, while indicating that the same values may be used in the case of human livers. Further, many source in the literature use the same values of material constants for all soft tissues and hence the same material parameters given above can be used in the case of human kidneys also. Hence these values of material constants are used for the purpose of this chapter. Although material constants can vary from person to person, between human beings and animals, from one soft tissue to another, and because of the presence or absence of certain pathologies, one can think that the material parameters used here reasonably approximate those of a representative healthy human kidney and human liver. Use of 'general purpose' material parameters is supported also by the fact that different sources in the literature report different values for the same material parameters and sometimes the values of a particular material constant reported by different sources could be so different that the simulation results obtained by making use of these different values can deviate so much from one another that the results may lose their practical significance.

By referring to [Henry Gray, 1918], human liver and human kidneys are subjected to boundary conditions that are so complicated that it is virtually impossible to reproduce the boundary conditions in a computer model. Hence the liver and the kidneys are subjected to arbitrary boundary conditions here. The idea is that if a computer model can give accurate solutions for many sets of arbitrary boundary conditions, and for different geometry, then it is reasonable to assume that the user can specify whatever set of boundary conditions one wants to impose and the solution obtained for the specified set of boundary conditions would be accurate. The liver and each of the two kidneys considered in this work are subjected to three different sets of boundary conditions. Hence there are nine problems to be solved, and one would expect to see that accurate solutions are obtained for all the nine problems. As can be seen later in this section, this happens to be the case indeed. Of course, two more kidneys (left and right kidneys of the Visible Human female) can also be simulated and further, each of the geometry may be subjected to many more sets of boundary conditions. However, the later two geometry and the extra sets of boundary conditions are not considered here since this author felt that the nine simulations are enough to get a feel of the simulation of biological organs and further simulations may not add much value to the present study.

One of the advantages of using the BEM for the simulation of biological organs is that there is no need to convert the surface models of biological organs into solid models. Further, each of the surface triangles that are used to describe the geometry can itself be considered to be a boundary element, thus eliminating the need for a separate step of discretization. Hence each of the *96* surface triangles that describe the geometry of kidneys acts as a boundary element also. One can note that this is not the case if FEM is to be used for the simulations. If FEM is to be employed, surface models need to be converted into solid models first, and then a separate step of discretization is required.

The boundary conditions applied during each of the nine simulations are explained now. Problem 1, Problem 2, and Problem 3 refer to cases where the left kidney of the Visible Human male is simulated. The location where the kidney is fixed (i.e., zero displacement specified in all the x, y, and z directions), and the location where a specified non-zero displacement is specified are shown in Figure 3.1. For each of these three problems, element numbers *8*, *15*, and *24* are subjected to the zero displacement condition in each of the x, y, and z directions, and the element number *94* is subjected to a non-zero displacement.

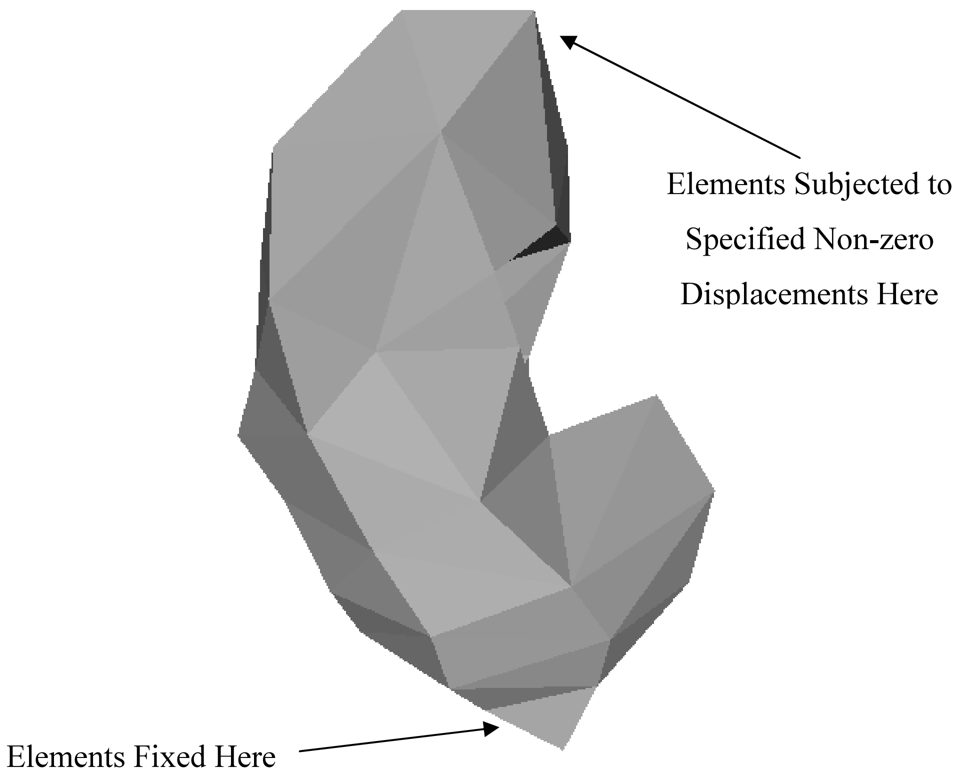

Figure 3.1 Boundary Conditions for the Left Kidney

Similarly, Problem 4, Problem 5, and Problem 6 refer to cases where the right kidney of the Visible Human male is simulated. The location where the kidney is fixed (i.e., zero displacement specified in all the x, y, and z directions), and the location where a specified non-zero displacement is specified are shown in Figure 3.2. For each of these three problems, element numbers *1*, *4*, and *11* are subjected to the zero displacement condition in each of the x, y, and z directions, and the element number *91* is subjected to a non-zero displacement.

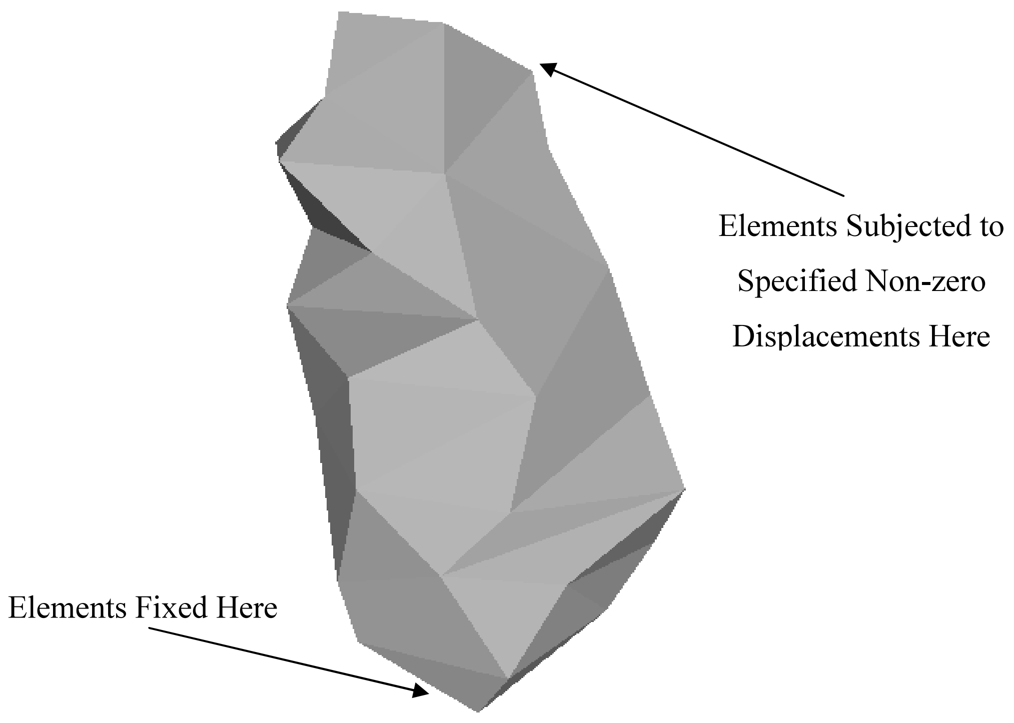

Figure 3.2 Boundary Conditions for the Right Kidney

Similarly, Problem 7, Problem 8, and Problem 9 refer to cases where the liver is simulated. The location where the liver is fixed (i.e., zero displacement specified in all the x, y, and z directions), and the location where a specified non-zero displacement is specified are shown in Figure 3.3. For each of these three problems, element numbers *55*, *60*, and *38* are subjected to the zero displacement condition in each of the x, y, and z directions, and the element number *59* is subjected to a non-zero displacement.

One can note that one needs to fix in all directions at least three boundary elements during any of the simulations. In the case of each of the nine problems considered above, three elements are fixed in all the directions.

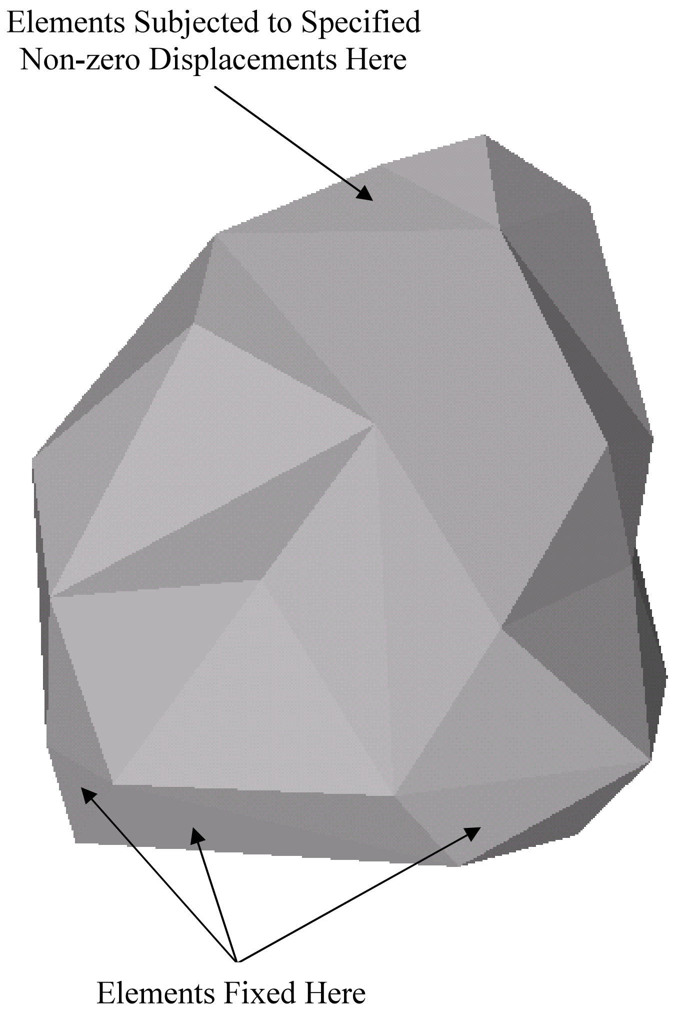

Figure 3.3 Boundary Conditions for the Liver

For Problem 1, the element *94* is subjected to the non-zero displacement of *5 mm* in the x direction. For Problem 2, the element *94* is subjected to the non-zero displacement of *5 mm* in the y direction. For Problem 3, the element *94* is subjected to the non-zero displacement of *5 mm* in the z direction. For Problem 4, the element *91* is subjected to the non-zero displacement of *5 mm* in the x direction. For Problem 5, the element *91* is subjected to the non-zero displacement of *5 mm* in the y direction. For Problem 6, the element *91* is subjected to the non-zero displacement of *5 mm* in the z direction. For Problem 7, the element *59* is subjected to the non-zero displacement of *5 mm* in the x direction. For Problem 8, the element *59* is subjected to the non-zero displacement of *5 mm* in the y direction. For Problem 9, the element *59* is subjected to the non-zero displacement of *5 mm* in the z direction. The value of *5 mm* is chosen for all the problems because this value of displacement corresponds to about *5%* deformation along the largest dimension of the biological organs considered, if the load that causes the deformation is applied along the same direction. Of course, specifying *5 mm* displacement in other directions can result in deformations that are not close to the *5%* deformation. However, one can note that the idea here is to specify physically meaningful non-zero displacement boundary conditions. This author has not aligned the liver or the kidneys to match the largest dimensions of the liver or the kidneys to any of the x, y, and z axes. Hence, although the non-zero displacement boundary conditions are specified along only one of x, y, and z axes (at a time) for all of the problems considered, one can note that it is reasonable to assume that the biological organs have been subjected to arbitray boundary conditions. However, one may note that the direction of the largest dimension for the liver and for both the kidneys is close to the z axis than any of the other two axes.

This author has found that it is of use to make use of a collection of simple, single task, browser based, text manipulation tools available from [Text Mechanic, n.d.] while preparing input files for the programs that are used to run the simulations.

Table 3.15 to Table 3.23 compare the displacement solutions obtained by using this author's code with the displacement solutions obtained by using the software from [Yijun Liu, n.d.], for all the problems considered (Problem 1 to Problem 9). In these tables, *u1*, *u2*, and *u3* refer to the displacements in the x, y, and z directions respectively, calculated by using this author's code. Similarly, *U1*, *U2*, and *U3* refer to the displacements in the x, y, and z directions respectively, calculated by using the software from [Yijun Liu, n.d.]. The solutions are listed for each of the *96* elements, for each of the problems considered (Problem 1 to Problem 9).

No attempt has been made here to compare the tractions obtained by using this author's code with the tractions obtained by using the software from [Yijun Liu, n.d.]. This is because one needs to calculate tractions only if one need to incorporate haptics into the simulations. Since only real-time graphics has been possible in this work (real-time haptics has not been possible), it does not make sense to look into tractions.

Table 3.15 Displacement Solutions for Problem 1

Element	*u1 (mm)*	*U1 (mm)*	*u2 (mm)*	*U2 (mm)*	*u3 (mm)*	*U3 (mm)*
1	0.773	0.734	0.144	0.147	-0.098	-0.008
2	0.727	0.583	-0.147	-0.168	-0.520	-0.481
3	0.955	1.135	-0.148	-0.168	-0.571	-0.663
4	0.970	1.064	0.328	0.284	0.258	0.219
5	0.127	0.218	-0.258	-0.238	-0.412	-0.449
6	-0.010	0.022	-0.449	-0.453	-0.547	-0.680
7	1.448	1.504	-0.028	-0.050	-0.647	-0.653
8	0.000	0.000	0.000	0.000	0.000	0.000
9	0.624	0.587	-0.487	-0.481	-1.125	-1.197
10	0.509	0.499	-0.400	-0.424	-0.938	-1.073
11	0.053	0.126	-0.535	-0.490	-0.763	-0.886
12	0.782	0.699	-0.230	-0.295	-0.703	-0.859
13	-0.011	0.025	-0.527	-0.485	-0.670	-0.766
14	0.061	0.087	-0.295	-0.336	-0.414	-0.546
15	0.000	0.000	0.000	0.000	0.000	0.000
16	0.489	0.414	0.230	0.214	0.247	0.279
17	1.295	1.320	-0.282	-0.270	-1.124	-1.226
18	1.868	2.149	0.024	-0.006	-0.863	-0.874
19	1.027	0.979	-0.245	-0.237	-0.836	-0.920
20	0.496	0.424	-0.430	-0.398	-0.877	-0.884
21	1.058	1.050	0.060	0.085	-0.445	-0.236
22	0.880	0.922	-0.466	-0.439	-1.186	-1.282
23	0.337	0.393	0.171	0.131	0.151	0.107
24	0.000	0.000	0.000	0.000	0.000	0.000
25	1.044	1.044	-0.376	-0.367	-1.187	-1.257
26	1.132	1.130	-0.058	-0.060	-0.697	-0.789
27	1.095	1.095	-0.370	-0.372	-1.154	-1.274
28	0.328	0.517	-0.291	-0.302	-0.522	-0.706
29	0.404	0.468	0.299	0.291	0.336	0.386
30	1.090	1.134	-0.140	-0.119	-0.679	-0.789
31	1.856	1.887	0.402	0.433	0.416	0.274
32	0.990	1.388	0.619	0.546	0.526	0.344
33	1.088	1.314	0.025	0.047	-0.493	-0.487
34	3.116	2.847	-0.101	0.071	-0.238	-0.144
35	1.557	1.609	-0.158	-0.172	-0.855	-0.929
36	1.776	1.927	-0.060	0.051	0.008	-0.096
37	1.151	1.303	0.645	0.648	0.738	0.611
38	0.839	0.957	-0.275	-0.316	-0.786	-0.964
39	2.053	2.098	0.251	0.261	0.265	0.064
40	2.031	2.059	-0.047	0.115	0.274	0.029
41	1.296	1.266	-0.163	-0.196	-0.921	-0.984
42	0.755	0.936	0.369	0.424	0.386	0.450
43	1.279	1.324	-0.295	-0.264	-1.174	-1.236
44	1.660	2.026	0.038	0.014	-0.660	-0.754
45	1.099	1.184	0.532	0.583	0.587	0.611
46	0.837	0.854	0.055	0.069	-0.139	-0.112
47	2.539	2.569	0.225	0.405	0.333	0.364
48	1.999	2.064	0.676	0.675	0.878	0.665
49	1.398	1.435	-0.148	-0.180	-0.978	-1.090

50	1.349	1.370	-0.195	-0.204	-1.054	-1.116
51	1.515	1.792	0.654	0.705	0.846	0.760
52	1.771	1.864	0.540	0.624	0.829	0.742
53	1.498	1.503	-0.176	-0.182	-0.769	-0.784
54	2.422	2.519	0.617	0.594	0.872	0.673
55	3.515	3.360	0.619	0.628	0.748	0.769
56	2.874	2.859	0.523	0.622	0.749	0.766
57	1.463	1.675	0.540	0.653	0.711	0.737
58	1.677	1.675	0.165	0.243	0.036	0.086
59	3.241	3.024	0.006	0.172	-0.219	-0.070
60	4.143	3.889	0.446	0.479	0.507	0.647
61	2.489	2.632	0.671	0.726	1.024	0.949
62	3.829	3.581	0.153	0.223	-0.200	-0.084
63	4.623	4.472	0.143	0.189	-0.127	0.019
64	3.749	3.570	0.000	0.118	-0.489	-0.428
65	2.967	3.073	0.013	0.108	-0.003	-0.104
66	2.090	2.147	0.654	0.711	0.953	0.886
67	2.075	2.079	0.538	0.540	0.719	0.574
68	1.821	1.850	0.335	0.444	0.442	0.507
69	3.623	3.623	0.103	0.062	-0.369	-0.502
70	2.259	2.372	-0.022	0.043	-0.354	-0.441
71	1.651	1.854	0.021	0.057	-0.317	-0.341
72	2.953	2.872	0.679	0.689	1.015	0.881
73	2.758	2.731	-0.045	0.206	-0.582	-0.395
74	1.785	2.127	0.158	0.188	-0.504	-0.433
75	2.520	2.519	-0.101	0.008	-0.612	-0.706
76	2.391	2.545	0.286	0.439	0.290	0.412
77	3.248	3.111	-0.178	0.091	-0.769	-0.613
78	3.195	3.137	-0.167	0.022	-0.772	-0.731
79	2.681	2.736	0.116	0.284	-0.050	0.052
80	3.840	3.806	0.373	0.316	0.174	0.144
81	3.935	3.729	0.429	0.506	0.345	0.431
82	3.781	3.662	-0.129	-0.001	-0.481	-0.562
83	4.129	4.037	0.163	0.171	0.092	0.019
84	3.868	3.728	-0.024	0.087	-0.580	-0.484
85	4.302	4.197	0.009	0.076	-0.469	-0.452
86	3.457	3.596	0.154	0.132	-0.042	-0.246
87	4.503	4.305	0.117	0.163	-0.281	-0.179
88	4.357	4.151	0.362	0.371	0.311	0.391
89	4.484	4.375	-0.032	0.047	-0.592	-0.573
90	3.824	3.886	-0.169	-0.074	-0.622	-0.653
91	4.192	4.128	0.385	0.369	0.484	0.539
92	3.603	3.681	0.412	0.369	0.749	0.595
93	4.717	4.797	0.064	0.029	-0.250	-0.157
94	5.000	5.000	0.000	0.000	0.000	0.000
95	4.926	4.800	0.092	0.097	-0.148	-0.093
96	4.922	4.823	0.055	0.083	-0.312	-0.360

Table 3.16 Displacement Solutions for Problem 2

Element	*u1 (mm)*	*U1 (mm)*	*u2 (mm)*	*U2 (mm)*	*u3 (mm)*	*U3 (mm)*
1	0.026	0.042	0.396	0.311	-0.411	-0.256
2	0.057	0.018	0.183	0.109	-0.500	-0.344
3	-0.383	-0.241	0.895	0.832	0.360	0.286
4	-0.206	-0.096	1.580	1.323	0.582	0.567
5	-0.048	-0.038	-0.038	-0.016	-0.019	-0.077
6	-0.022	-0.042	-0.065	-0.065	0.108	0.047
7	-0.119	0.063	1.121	0.945	-0.277	-0.112
8	0.000	0.000	0.000	0.000	0.000	0.000
9	-0.099	-0.071	0.163	0.110	-0.130	-0.149
10	-0.055	-0.033	0.055	0.043	-0.188	-0.217
11	-0.041	-0.046	-0.071	-0.057	0.140	0.023
12	0.008	0.017	0.174	0.112	-0.429	-0.382
13	-0.033	-0.043	-0.071	-0.059	0.128	0.058
14	-0.086	-0.081	0.094	0.051	0.203	0.149
15	0.000	0.000	0.000	0.000	0.000	0.000
16	0.040	0.072	0.416	0.291	0.050	0.115
17	-0.100	0.112	0.617	0.494	-0.284	-0.406
18	-0.071	0.074	1.500	1.475	-0.304	-0.142
19	-0.046	0.062	0.366	0.282	-0.441	-0.463
20	-0.186	-0.134	0.239	0.161	0.199	0.144
21	0.016	0.026	0.440	0.471	-0.716	-0.413
22	-0.104	0.010	0.315	0.246	-0.203	-0.350
23	-0.125	-0.067	0.683	0.525	0.438	0.401
24	0.000	0.000	0.000	0.000	0.000	0.000
25	-0.032	0.068	0.379	0.298	-0.392	-0.453
26	0.010	0.089	0.449	0.419	-0.463	-0.505
27	-0.160	0.002	0.510	0.363	-0.130	-0.263
28	-0.226	-0.207	0.330	0.344	0.305	0.251
29	-0.016	0.067	0.740	0.576	0.351	0.352
30	-0.055	0.053	0.442	0.429	-0.306	-0.411
31	0.071	0.295	1.474	1.230	-0.576	-0.590
32	-0.149	0.164	1.425	1.183	0.052	-0.203
33	-0.112	0.036	0.743	0.752	-0.241	-0.332
34	0.156	0.241	2.379	2.171	-0.805	-0.812
35	-0.314	-0.066	1.170	0.912	0.066	-0.023
36	0.032	0.215	1.309	1.179	-0.620	-0.539
37	-0.067	0.180	1.494	1.314	0.060	0.002
38	-0.339	-0.203	0.623	0.484	0.334	0.175
39	0.134	0.335	1.495	1.250	-0.713	-0.693
40	0.080	0.235	1.814	1.434	-0.470	-0.472
41	-0.124	0.106	0.778	0.605	-0.427	-0.343
42	-0.152	-0.018	1.287	1.210	0.473	0.482
43	-0.064	0.132	0.546	0.471	-0.443	-0.493
44	-0.059	0.127	1.419	1.360	-0.335	-0.264
45	-0.103	0.072	1.683	1.445	0.387	0.313
46	-0.345	-0.185	1.160	0.948	0.477	0.469
47	0.173	0.328	1.904	1.745	-0.847	-0.906
48	0.106	0.303	1.852	1.562	-0.342	-0.525
49	-0.083	0.169	0.790	0.628	-0.348	-0.333

50	-0.079	0.155	0.696	0.579	-0.461	-0.423
51	0.025	0.238	1.772	1.602	0.079	-0.151
52	-0.055	0.099	2.319	2.059	0.402	0.320
53	-0.370	-0.206	1.229	1.027	0.250	0.225
54	0.181	0.318	2.126	1.875	-0.531	-0.755
55	0.222	0.296	3.044	2.788	-0.737	-0.749
56	0.053	0.200	3.098	2.808	0.320	0.191
57	-0.018	0.165	1.912	1.752	0.287	0.105
58	-0.266	-0.108	2.121	1.823	0.574	0.575
59	0.220	0.268	2.374	2.172	-0.873	-0.946
60	0.097	0.191	3.735	3.356	-0.621	-0.673
61	0.090	0.241	2.711	2.498	0.155	-0.041
62	0.205	0.257	2.839	2.586	-1.088	-1.135
63	0.179	0.184	3.988	3.805	-1.106	-1.200
64	0.055	0.155	2.858	2.678	-0.649	-0.716
65	0.154	0.239	2.477	2.430	-0.590	-0.697
66	0.038	0.177	2.483	2.233	0.338	0.249
67	-0.059	0.040	2.507	2.287	0.458	0.454
68	-0.184	-0.039	2.327	2.090	0.575	0.554
69	0.018	0.091	2.845	2.858	-0.407	-0.361
70	-0.217	-0.092	2.376	2.121	0.446	0.364
71	-0.296	-0.123	1.802	1.731	0.506	0.470
72	0.178	0.302	2.798	2.438	-0.277	-0.513
73	0.057	0.281	2.308	2.051	-0.256	-0.262
74	0.047	0.217	1.591	1.587	-0.245	-0.247
75	-0.182	-0.020	2.352	1.961	0.174	0.107
76	-0.102	0.078	2.740	2.620	0.482	0.407
77	-0.068	0.165	2.782	2.378	-0.200	-0.283
78	-0.115	0.036	2.821	2.535	-0.069	0.020
79	-0.138	0.011	2.859	2.677	0.390	0.391
80	0.287	0.266	2.892	2.835	-1.140	-1.257
81	0.256	0.305	3.114	2.822	-1.155	-1.181
82	-0.239	-0.086	3.436	3.173	0.002	0.037
83	-0.186	-0.082	4.108	3.809	0.157	0.129
84	0.096	0.190	2.912	2.766	-0.814	-0.964
85	0.146	0.165	3.300	3.250	-1.030	-1.038
86	-0.126	-0.044	3.362	3.222	0.213	0.140
87	0.260	0.232	3.632	3.384	-1.173	-1.286
88	0.132	0.187	3.721	3.372	-0.950	-1.097
89	-0.072	0.049	3.688	3.495	-0.520	-0.685
90	-0.136	-0.041	3.192	3.138	-0.259	-0.285
91	-0.040	0.093	4.079	3.788	-0.123	-0.298
92	-0.073	0.032	3.691	3.526	0.100	0.057
93	-0.228	-0.197	4.313	4.386	-0.137	-0.144
94	0.000	0.000	5.000	5.000	0.000	0.000
95	0.122	0.198	4.493	4.329	-0.646	-0.855
96	-0.010	0.065	4.289	4.066	-0.537	-0.650

Table 3.17 Displacement Solutions for Problem 3

Element	*u1 (mm)*	*U1 (mm)*	*u2 (mm)*	*U2 (mm)*	*u3 (mm)*	*U3 (mm)*
1	-0.059	-0.190	-0.016	-0.130	1.498	1.223
2	-0.385	-0.379	0.018	0.074	1.387	1.267
3	-0.125	-0.320	-0.101	-0.110	0.625	0.750
4	0.191	-0.028	-0.065	-0.014	1.088	0.974
5	-0.186	-0.238	0.250	0.245	0.646	0.870
6	-0.146	-0.154	0.404	0.420	0.543	0.756
7	-0.097	-0.345	-0.228	-0.176	1.531	1.404
8	0.000	0.000	0.000	0.000	0.000	0.000
9	-0.275	-0.336	0.101	0.189	0.934	1.075
10	-0.309	-0.350	0.193	0.241	0.989	1.114
11	-0.186	-0.214	0.423	0.407	0.643	0.882
12	-0.375	-0.425	0.052	0.148	1.210	1.241
13	-0.152	-0.148	0.428	0.420	0.593	0.782
14	-0.117	-0.137	0.197	0.250	0.370	0.581
15	0.000	0.000	0.000	0.000	0.000	0.000
16	0.099	0.006	0.064	0.009	1.090	0.875
17	-0.157	-0.441	-0.163	-0.059	1.004	1.217
18	-0.053	-0.234	-0.298	-0.295	1.618	1.580
19	-0.269	-0.446	-0.025	0.037	1.160	1.284
20	-0.269	-0.296	0.082	0.169	0.683	0.806
21	-0.199	-0.263	-0.077	-0.124	1.686	1.328
22	-0.258	-0.401	-0.009	0.085	0.965	1.200
23	0.074	-0.080	0.101	0.110	0.517	0.453
24	0.000	0.000	0.000	0.000	0.000	0.000
25	-0.267	-0.438	-0.030	0.056	1.063	1.251
26	-0.261	-0.429	-0.039	-0.050	1.229	1.335
27	-0.217	-0.414	-0.119	0.002	0.898	1.128
28	-0.189	-0.304	0.073	0.073	0.447	0.663
29	0.164	0.051	0.095	0.116	0.788	0.768
30	-0.228	-0.397	-0.059	-0.056	1.095	1.266
31	0.373	-0.038	-0.070	0.039	2.895	2.862
32	0.199	-0.191	0.020	0.114	1.948	2.292
33	-0.152	-0.294	-0.155	-0.173	1.152	1.329
34	0.093	0.036	-0.557	-0.590	3.448	3.315
35	0.015	-0.271	-0.259	-0.183	0.949	1.038
36	0.188	-0.068	-0.400	-0.335	2.456	2.350
37	0.256	-0.104	0.027	0.046	2.158	2.200
38	-0.215	-0.365	-0.083	-0.027	0.642	0.845
39	0.357	-0.005	-0.230	-0.167	3.014	2.889
40	0.311	0.038	-0.655	-0.466	2.452	2.348
41	-0.102	-0.446	-0.200	-0.121	1.212	1.205
42	0.189	0.001	0.017	0.082	1.110	1.186
43	-0.169	-0.449	-0.117	-0.042	1.092	1.269
44	-0.044	-0.252	-0.317	-0.339	1.678	1.648
45	0.241	0.019	-0.076	-0.013	1.601	1.702
46	0.011	-0.189	-0.061	-0.008	0.679	0.642
47	0.366	0.177	-0.493	-0.400	3.160	3.228
48	0.387	0.081	-0.204	-0.109	2.747	2.938
49	-0.091	-0.451	-0.205	-0.111	1.133	1.189

50	-0.107	-0.451	-0.172	-0.095	1.166	1.240
51	0.338	0.062	-0.103	-0.096	2.249	2.580
52	0.397	0.212	-0.354	-0.307	2.040	2.101
53	-0.021	-0.258	-0.224	-0.180	0.908	0.935
54	0.355	0.166	-0.407	-0.351	2.950	3.159
55	0.172	0.111	-0.868	-0.803	3.358	3.331
56	0.494	0.348	-0.692	-0.678	2.452	2.569
57	0.327	0.133	-0.192	-0.197	1.881	2.227
58	0.336	0.110	-0.265	-0.220	1.344	1.293
59	0.072	0.053	-0.568	-0.599	3.481	3.413
60	0.184	0.134	-0.931	-0.905	3.633	3.491
61	0.455	0.280	-0.602	-0.615	2.497	2.700
62	-0.076	-0.068	-0.689	-0.696	3.651	3.611
63	-0.079	-0.137	-0.772	-0.808	3.711	3.703
64	-0.017	-0.057	-0.571	-0.638	3.570	3.558
65	0.112	-0.013	-0.614	-0.665	3.277	3.321
66	0.429	0.242	-0.452	-0.421	2.224	2.314
67	0.511	0.311	-0.450	-0.463	1.958	1.878
68	0.455	0.285	-0.345	-0.332	1.619	1.667
69	0.012	-0.098	-0.489	-0.589	3.319	3.316
70	0.336	0.112	-0.372	-0.382	1.469	1.492
71	0.211	0.014	-0.205	-0.243	1.100	1.179
72	0.340	0.176	-0.723	-0.640	2.857	3.032
73	0.004	-0.131	-0.441	-0.437	2.574	2.669
74	-0.015	-0.145	-0.299	-0.348	1.828	2.046
75	0.204	-0.022	-0.388	-0.375	1.646	1.628
76	0.469	0.323	-0.524	-0.560	1.844	2.083
77	-0.019	-0.121	-0.657	-0.648	2.604	2.672
78	-0.008	-0.104	-0.556	-0.581	2.485	2.427
79	0.426	0.304	-0.590	-0.605	1.828	1.903
80	-0.058	-0.097	-0.790	-0.794	3.637	3.659
81	-0.035	-0.041	-0.888	-0.845	3.637	3.597
82	-0.020	-0.119	-0.659	-0.692	2.844	2.752
83	0.103	0.013	-0.480	-0.560	3.353	3.272
84	-0.058	-0.104	-0.606	-0.684	3.657	3.651
85	-0.119	-0.159	-0.760	-0.817	3.695	3.673
86	0.261	0.040	-0.705	-0.708	2.484	2.596
87	-0.119	-0.154	-0.795	-0.827	3.688	3.682
88	-0.004	-0.060	-0.947	-0.934	3.741	3.687
89	-0.187	-0.221	-0.806	-0.850	3.564	3.597
90	-0.102	-0.204	-0.681	-0.768	3.216	3.293
91	0.274	0.198	-0.692	-0.751	3.523	3.545
92	0.438	0.303	-0.781	-0.731	2.886	2.964
93	-0.171	-0.219	-0.599	-0.489	3.628	3.804
94	0.000	0.000	0.000	0.000	5.000	5.000
95	0.157	0.017	-0.520	-0.571	3.940	3.835
96	0.023	-0.089	-0.708	-0.752	3.772	3.693

Table 3.18 Displacement Solutions for Problem 4

Element	*u1 (mm)*	*U1 (mm)*	*u2 (mm)*	*U2 (mm)*	*u3 (mm)*	*U3 (mm)*
1	0.000	0.000	0.000	0.000	0.000	0.000
2	0.787	0.697	0.282	0.314	0.599	0.524
3	2.171	2.130	0.017	0.083	0.006	-0.157
4	0.000	0.000	0.000	0.000	0.000	0.000
5	0.130	0.038	0.120	0.114	0.188	0.225
6	0.153	0.115	0.169	0.163	0.270	0.265
7	0.782	0.680	0.014	0.071	0.063	0.008
8	0.399	0.444	0.052	0.112	0.185	0.243
9	0.422	0.464	-0.002	0.043	0.057	0.043
10	0.280	0.281	-0.082	-0.057	-0.122	-0.177
11	0.000	0.000	0.000	0.000	0.000	0.000
12	0.631	0.637	0.168	0.202	0.478	0.462
13	0.706	0.734	-0.138	-0.052	-0.314	-0.365
14	0.542	0.585	-0.173	-0.091	-0.403	-0.426
15	0.462	0.446	0.257	0.315	0.372	0.409
16	0.522	0.568	-0.180	-0.099	-0.484	-0.528
17	0.309	0.313	-0.094	-0.047	-0.300	-0.312
18	0.363	0.295	0.102	0.162	0.113	0.200
19	1.100	0.927	0.221	0.252	0.443	0.422
20	3.109	3.223	0.093	0.216	0.427	0.565
21	0.422	0.493	0.214	0.314	0.268	0.341
22	0.694	0.681	-0.098	0.039	-0.334	-0.237
23	0.464	0.425	0.008	0.054	-0.101	-0.134
24	1.102	0.904	0.177	0.206	0.249	0.127
25	0.745	0.826	-0.198	-0.121	-0.624	-0.678
26	1.014	1.047	-0.099	-0.001	-0.221	-0.305
27	1.157	0.972	0.164	0.218	0.198	0.047
28	1.114	1.139	0.351	0.448	0.617	0.653
29	0.721	0.793	-0.170	-0.078	-0.564	-0.559
30	1.827	1.714	-0.127	0.004	-0.275	-0.418
31	1.305	1.295	0.163	0.284	0.237	0.287
32	1.526	1.398	0.300	0.389	0.542	0.580
33	1.119	1.067	-0.125	0.025	-0.115	-0.162
34	1.388	1.303	-0.162	-0.034	-0.552	-0.553
35	2.232	2.194	0.208	0.294	0.428	0.441
36	1.641	1.501	0.348	0.456	0.650	0.672
37	1.165	1.189	-0.243	-0.122	-0.762	-0.788
38	0.940	0.974	0.193	0.224	0.260	0.066
39	0.719	0.763	0.001	0.156	-0.080	0.004
40	1.271	1.345	0.027	0.148	0.036	0.080
41	1.382	1.354	-0.209	-0.101	-0.706	-0.769
42	1.344	1.235	0.376	0.431	0.564	0.527
43	1.064	1.086	-0.182	-0.094	-0.609	-0.601
44	1.753	1.607	0.036	0.127	0.146	0.134
45	1.499	1.469	0.295	0.384	0.342	0.470
46	2.265	2.162	0.346	0.402	0.589	0.554
47	1.812	1.742	-0.266	-0.100	-0.802	-0.843
48	2.037	1.943	-0.157	0.020	-0.550	-0.529
49	1.560	1.499	0.103	0.180	-0.005	-0.093

50	1.466	1.544	0.257	0.371	0.347	0.353
51	1.167	1.109	-0.131	0.068	-0.430	-0.307
52	1.914	1.838	-0.086	0.069	-0.190	-0.204
53	2.132	2.057	0.448	0.523	0.640	0.687
54	1.805	1.833	-0.259	-0.107	-0.666	-0.799
55	2.413	2.400	0.242	0.320	0.460	0.452
56	2.119	2.061	-0.234	-0.085	-0.694	-0.836
57	2.026	1.946	-0.284	-0.133	-0.768	-0.868
58	2.216	2.157	-0.194	-0.036	-0.133	-0.480
59	2.073	1.975	0.380	0.449	0.531	0.512
60	2.402	2.484	0.397	0.456	0.655	0.678
61	2.442	2.527	0.382	0.424	0.690	0.699
62	2.260	2.244	-0.340	-0.166	-0.868	-0.971
63	2.589	2.689	-0.256	-0.126	-0.837	-0.928
64	2.152	2.269	0.256	0.373	0.275	0.329
65	2.620	2.582	0.230	0.303	0.653	0.569
66	4.470	4.456	-0.048	-0.035	0.219	0.179
67	3.381	3.352	-0.029	0.035	0.081	-0.040
68	3.123	3.272	0.235	0.284	0.628	0.637
69	2.681	2.828	0.086	0.205	0.298	0.349
70	2.540	2.677	-0.262	-0.061	-0.837	-0.649
71	2.749	2.919	-0.055	0.038	-0.212	-0.250
72	3.712	3.659	-0.145	-0.056	-0.359	-0.224
73	2.482	2.543	0.276	0.378	0.432	0.506
74	2.832	3.003	0.247	0.316	0.478	0.566
75	2.150	2.277	0.043	0.134	-0.102	-0.217
76	4.069	4.168	-0.113	-0.183	-0.533	-0.514
77	3.645	3.659	-0.240	-0.230	-0.690	-0.656
78	2.952	3.151	-0.131	-0.121	-0.755	-0.810
79	2.791	2.981	0.094	0.128	-0.038	-0.192
80	4.143	4.252	0.191	0.103	0.450	0.328
81	2.975	3.220	0.227	0.270	0.265	0.254
82	3.363	3.319	-0.012	-0.052	-0.262	-0.514
83	4.029	3.897	0.347	0.323	0.743	0.697
84	4.550	4.536	0.073	0.056	0.390	0.301
85	4.195	4.291	0.281	0.268	0.636	0.577
86	4.472	4.333	0.194	0.222	0.519	0.490
87	4.206	4.158	-0.013	-0.012	-0.365	-0.332
88	4.798	4.714	0.149	0.109	0.041	-0.027
89	4.510	4.492	-0.160	-0.203	-0.401	-0.351
90	4.506	4.474	0.174	0.205	0.571	0.537
91	5.000	5.000	0.000	0.000	0.000	0.000
92	3.822	3.944	0.286	0.276	0.696	0.606
93	4.919	4.908	0.024	0.020	0.193	0.154
94	4.583	4.723	0.093	0.072	0.223	0.203
95	4.917	4.931	0.014	0.002	0.151	0.103
96	4.914	4.945	0.043	0.024	0.260	0.179

Table 3.19 Displacement Solutions for Problem 5

Element	*u1 (mm)*	*U1 (mm)*	*u2 (mm)*	*U2 (mm)*	*u3 (mm)*	*U3 (mm)*
1	0.000	0.000	0.000	0.000	0.000	0.000
2	0.131	0.157	0.624	0.522	-0.086	-0.005
3	-0.030	0.029	2.072	1.953	-0.550	-0.658
4	0.000	0.000	0.000	0.000	0.000	0.000
5	0.178	0.083	0.266	0.180	0.384	0.330
6	0.121	0.086	0.179	0.107	0.233	0.193
7	0.090	0.089	0.504	0.416	-0.652	-0.479
8	0.062	0.100	0.167	0.217	-0.334	-0.262
9	0.061	0.075	0.187	0.230	-0.404	-0.356
10	0.051	0.052	0.136	0.171	-0.241	-0.183
11	0.000	0.000	0.000	0.000	0.000	0.000
12	0.106	0.120	0.395	0.395	-0.309	-0.251
13	0.059	0.057	0.577	0.602	-0.455	-0.424
14	0.048	0.053	0.549	0.538	-0.176	-0.224
15	0.145	0.190	0.384	0.432	0.119	0.217
16	0.059	0.072	0.743	0.752	0.071	0.072
17	0.159	0.139	0.652	0.605	0.387	0.327
18	0.248	0.216	0.755	0.582	0.541	0.489
19	0.181	0.165	0.880	0.684	-0.581	-0.414
20	0.045	0.116	2.740	2.697	-0.747	-0.836
21	0.241	0.263	0.684	0.701	0.445	0.423
22	0.198	0.236	1.319	1.115	0.650	0.576
23	0.213	0.201	0.907	0.771	0.556	0.478
24	0.174	0.168	0.846	0.648	-0.674	-0.502
25	0.117	0.093	1.165	1.077	0.310	0.159
26	0.003	0.014	0.942	0.944	-0.633	-0.560
27	0.168	0.183	0.945	0.747	-0.622	-0.398
28	0.165	0.198	1.134	0.980	0.066	0.040
29	0.170	0.174	1.298	1.244	0.538	0.476
30	-0.112	0.007	1.838	1.615	-0.696	-0.650
31	0.076	0.135	1.137	1.050	-0.317	-0.364
32	0.135	0.168	1.322	1.051	-0.496	-0.393
33	-0.008	0.034	1.094	0.920	-0.682	-0.526
34	-0.051	0.024	1.519	1.287	-0.489	-0.509
35	0.050	0.089	1.856	1.689	-0.449	-0.479
36	0.123	0.175	1.406	1.144	-0.374	-0.233
37	0.020	0.049	1.470	1.300	0.019	-0.149
38	0.206	0.237	1.293	1.257	0.376	0.473
39	0.213	0.243	1.214	1.089	0.585	0.497
40	0.031	0.085	1.307	1.235	-0.275	-0.397
41	0.054	0.093	1.825	1.565	0.312	0.137
42	0.212	0.264	1.419	1.222	0.157	0.244
43	0.121	0.135	1.656	1.532	0.584	0.485
44	-0.020	0.044	1.664	1.448	-0.500	-0.513
45	0.113	0.135	1.377	1.236	-0.459	-0.429
46	0.067	0.098	1.827	1.622	-0.500	-0.472
47	-0.048	0.072	2.181	1.968	-0.157	-0.021
48	0.130	0.187	2.532	2.270	0.275	0.366
49	0.205	0.249	2.019	1.810	0.426	0.452

50	0.198	0.271	1.721	1.641	0.298	0.295
51	0.125	0.202	1.732	1.512	0.525	0.494
52	-0.104	0.014	1.845	1.647	-0.740	-0.688
53	0.173	0.215	1.982	1.740	-0.164	-0.167
54	-0.121	0.000	2.046	1.901	-0.479	-0.446
55	-0.005	0.045	2.150	2.014	-0.507	-0.553
56	-0.066	0.023	2.342	2.112	-0.504	-0.470
57	0.040	0.087	2.528	2.256	0.100	0.173
58	-0.066	0.002	2.258	2.105	-0.622	-0.615
59	0.193	0.269	2.069	1.881	0.023	0.104
60	0.097	0.152	2.007	1.983	-0.426	-0.409
61	0.044	0.071	1.957	1.913	-0.517	-0.559
62	-0.033	0.033	2.540	2.380	-0.294	-0.118
63	0.009	0.045	2.873	2.696	-0.178	-0.287
64	0.217	0.292	2.343	2.257	0.220	0.168
65	-0.029	0.033	2.207	2.100	-0.623	-0.659
66	-0.206	-0.074	3.969	3.868	-0.969	-1.063
67	0.005	0.089	3.006	2.921	-0.709	-0.719
68	0.097	0.158	2.713	2.753	-0.662	-0.694
69	0.029	0.095	2.428	2.416	-0.601	-0.713
70	-0.026	0.053	2.657	2.546	-0.387	-0.461
71	0.001	0.077	2.605	2.629	-0.516	-0.591
72	-0.073	0.058	3.459	3.239	-0.550	-0.666
73	0.177	0.208	2.406	2.245	-0.138	-0.241
74	0.189	0.215	2.734	2.697	-0.101	-0.266
75	0.187	0.218	2.521	2.516	0.295	0.304
76	0.024	0.015	4.310	4.232	0.006	-0.036
77	-0.024	0.018	3.697	3.451	-0.253	-0.385
78	0.074	0.065	3.304	3.234	-0.029	-0.055
79	0.210	0.214	3.026	3.068	0.151	0.149
80	-0.079	-0.016	3.348	3.486	-0.915	-1.008
81	0.245	0.266	3.040	3.131	0.056	-0.006
82	0.181	0.126	3.647	3.476	0.140	0.130
83	0.030	0.144	3.322	3.318	-0.634	-0.649
84	-0.195	-0.095	3.872	3.894	-0.924	-1.061
85	-0.056	0.021	3.350	3.444	-0.929	-1.021
86	-0.058	0.119	3.742	3.716	-0.655	-0.635
87	0.076	0.148	4.454	4.242	0.128	0.133
88	0.231	0.228	4.761	4.683	0.008	0.045
89	-0.103	-0.081	4.523	4.328	-0.251	-0.430
90	-0.163	-0.050	3.695	3.663	-0.896	-1.001
91	0.000	0.000	5.000	5.000	0.000	0.000
92	0.109	0.186	3.377	3.460	-0.402	-0.435
93	-0.220	-0.138	4.541	4.497	-0.706	-0.826
94	0.145	0.129	4.244	4.365	-0.148	-0.320
95	-0.194	-0.147	4.722	4.695	-0.447	-0.645
96	-0.176	-0.154	4.572	4.648	-0.645	-0.668

Table 3.20 Displacement Solutions for Problem 6

Element	*u1 (mm)*	*U1 (mm)*	*u2 (mm)*	*U2 (mm)*	*u3 (mm)*	*U3 (mm)*
1	0.000	0.000	0.000	0.000	0.000	0.000
2	0.349	0.356	0.061	0.088	0.653	0.514
3	0.167	0.118	-0.130	-0.155	2.006	1.978
4	0.000	0.000	0.000	0.000	0.000	0.000
5	0.214	0.120	0.186	0.166	0.267	0.215
6	0.206	0.176	0.158	0.134	0.230	0.176
7	0.219	0.246	-0.037	-0.031	0.692	0.618
8	0.236	0.284	-0.009	0.014	0.476	0.470
9	0.227	0.261	-0.052	-0.048	0.568	0.558
10	0.179	0.200	0.018	0.042	0.591	0.640
11	0.000	0.000	0.000	0.000	0.000	0.000
12	0.317	0.334	-0.009	0.017	0.573	0.545
13	0.079	0.085	0.090	0.106	1.125	1.160
14	0.077	0.090	0.157	0.166	1.002	1.070
15	0.315	0.311	0.130	0.191	0.335	0.334
16	0.042	0.045	0.226	0.250	0.924	0.972
17	0.168	0.154	0.269	0.299	0.583	0.643
18	0.238	0.223	0.266	0.283	0.377	0.333
19	0.341	0.327	-0.019	0.002	0.742	0.658
20	0.321	0.401	-0.448	-0.485	2.780	2.646
21	0.273	0.254	0.235	0.257	0.352	0.352
22	0.086	0.074	0.270	0.284	0.751	0.701
23	0.157	0.120	0.285	0.312	0.466	0.494
24	0.297	0.266	-0.017	0.012	0.701	0.610
25	0.022	0.020	0.279	0.271	1.027	1.126
26	0.051	0.113	0.012	0.048	1.375	1.394
27	0.308	0.229	0.019	0.070	0.707	0.633
28	0.365	0.380	0.148	0.188	0.701	0.716
29	0.044	0.024	0.281	0.274	0.899	0.913
30	0.109	0.127	-0.090	-0.047	1.945	1.881
31	0.319	0.357	0.051	0.058	0.962	0.928
32	0.437	0.474	0.030	0.097	0.957	0.855
33	0.134	0.183	-0.033	0.034	1.478	1.307
34	0.025	0.056	0.031	0.049	1.657	1.596
35	0.421	0.465	-0.055	-0.010	1.675	1.664
36	0.455	0.464	0.079	0.160	0.978	0.875
37	-0.027	-0.014	0.168	0.154	1.328	1.424
38	0.195	0.112	0.188	0.228	0.617	0.660
39	0.114	0.096	0.268	0.264	0.540	0.534
40	0.285	0.259	0.089	0.107	1.260	1.426
41	-0.012	-0.019	0.096	0.131	1.184	1.271
42	0.308	0.283	0.146	0.202	0.748	0.692
43	0.001	-0.007	0.199	0.197	0.955	0.987
44	0.240	0.255	-0.072	-0.012	1.583	1.587
45	0.380	0.452	0.033	0.072	1.005	0.971
46	0.490	0.510	0.006	0.085	1.649	1.580
47	-0.006	0.010	0.060	0.088	1.979	1.814
48	-0.119	-0.050	-0.046	-0.005	1.531	1.353
49	0.082	0.071	0.074	0.118	1.021	1.002

50	0.188	0.183	0.118	0.137	0.825	0.868
51	0.014	0.003	0.167	0.174	0.935	0.851
52	0.107	0.148	-0.123	-0.096	1.876	1.796
53	0.316	0.371	0.021	0.103	1.146	1.093
54	0.009	-0.005	-0.030	-0.015	2.026	2.012
55	0.424	0.487	-0.159	-0.114	2.083	2.075
56	-0.069	-0.056	-0.113	-0.073	2.220	2.145
57	-0.112	-0.070	-0.025	0.021	1.807	1.701
58	0.075	0.021	-0.115	-0.097	2.013	2.000
59	0.219	0.222	-0.008	0.051	1.115	1.058
60	0.418	0.410	-0.054	-0.076	1.522	1.524
61	0.519	0.556	-0.037	-0.054	1.668	1.750
62	-0.155	-0.100	-0.068	-0.066	2.205	2.012
63	-0.178	-0.119	-0.195	-0.242	2.326	2.418
64	0.087	0.128	-0.079	-0.105	1.210	1.238
65	0.481	0.493	-0.209	-0.197	2.060	2.115
66	0.070	0.178	-0.772	-0.749	3.604	3.572
67	0.261	0.274	-0.474	-0.442	3.191	3.160
68	0.309	0.370	-0.513	-0.496	2.390	2.418
69	0.312	0.357	-0.261	-0.321	2.475	2.522
70	-0.160	-0.026	-0.154	-0.210	2.674	2.712
71	0.115	0.148	-0.260	-0.303	2.755	2.862
72	0.080	0.205	-0.621	-0.571	3.723	3.502
73	0.132	0.202	-0.226	-0.220	1.508	1.511
74	0.233	0.336	-0.285	-0.335	1.929	2.099
75	-0.009	0.026	-0.089	-0.131	1.337	1.419
76	-0.188	-0.125	-0.256	-0.244	3.883	3.939
77	-0.116	-0.015	-0.577	-0.589	3.366	3.328
78	-0.189	-0.110	-0.315	-0.357	2.437	2.583
79	0.046	0.098	-0.291	-0.299	1.835	1.966
80	0.228	0.282	-0.845	-0.762	3.206	3.315
81	0.153	0.251	-0.360	-0.412	1.899	2.063
82	0.045	0.006	-0.393	-0.314	2.624	2.551
83	0.238	0.334	-0.833	-0.735	2.852	2.786
84	-0.040	0.065	-0.817	-0.788	3.269	3.348
85	0.187	0.231	-0.846	-0.806	3.086	3.088
86	0.078	0.204	-0.953	-0.893	3.026	2.905
87	-0.072	0.082	-0.110	-0.205	3.786	3.450
88	0.247	0.293	-0.054	-0.042	4.058	3.897
89	-0.219	-0.113	-0.328	-0.435	4.358	4.192
90	0.016	0.121	-0.944	-0.884	3.110	3.103
91	0.000	0.000	0.000	0.000	5.000	5.000
92	0.338	0.363	-0.786	-0.753	2.588	2.621
93	-0.450	-0.314	-0.587	-0.572	3.684	3.568
94	0.245	0.226	-0.524	-0.544	3.362	3.400
95	-0.433	-0.348	-0.370	-0.387	3.964	3.795
96	-0.366	-0.346	-0.648	-0.530	3.528	3.567

Table 3.21 Displacement Solutions for Problem 7

Element	*u1 (mm)*	*U1 (mm)*	*u2 (mm)*	*U2 (mm)*	*u3 (mm)*	*U3 (mm)*
1	2.859	2.297	0.237	0.332	0.577	0.403
2	0.216	0.590	0.112	0.161	-0.134	-0.331
3	0.075	0.343	0.045	0.023	-0.095	-0.205
4	0.040	0.169	0.206	0.202	0.114	0.096
5	-0.047	0.077	0.011	0.015	-0.046	-0.066
6	4.601	3.988	1.007	1.212	-0.387	-0.612
7	0.165	0.203	0.071	0.053	-0.138	-0.128
8	0.470	0.420	0.146	0.074	-0.286	-0.250
9	0.178	0.301	0.147	0.125	-0.195	-0.216
10	-0.037	0.064	0.125	0.082	0.016	-0.055
11	0.095	0.155	-0.022	0.062	-0.062	-0.041
12	1.216	1.094	0.489	0.467	0.769	0.480
13	0.136	0.244	0.140	0.110	-0.181	-0.204
14	-0.011	0.125	-0.025	0.014	-0.162	-0.114
15	0.210	0.309	0.169	0.132	-0.175	-0.216
16	0.066	0.202	-0.076	0.020	-0.166	-0.124
17	0.083	0.180	0.072	0.054	-0.156	-0.173
18	5.083	4.780	0.914	1.176	-0.093	-0.075
19	0.186	0.365	0.060	-0.012	-0.111	-0.159
20	0.638	0.627	0.338	0.278	0.021	0.023
21	0.062	0.193	0.202	0.190	0.200	0.104
22	0.032	0.079	0.050	0.030	-0.099	-0.097
23	-0.071	0.049	0.062	0.023	-0.109	-0.096
24	0.223	0.332	0.181	0.161	-0.086	-0.136
25	0.205	0.316	-0.293	-0.157	-0.260	-0.219
26	0.266	0.329	0.168	0.103	-0.215	-0.227
27	-0.063	0.030	0.067	0.041	-0.099	-0.080
28	0.185	0.000	0.144	-0.000	-0.061	-0.000
29	-0.064	0.081	0.073	0.043	-0.098	-0.083
30	0.008	0.129	-0.056	0.014	-0.154	-0.152
31	0.060	0.195	-0.085	0.022	-0.159	-0.133
32	3.268	2.694	0.134	0.271	0.349	0.163
33	0.371	0.529	0.174	0.203	0.240	0.120
34	3.814	3.967	0.471	0.823	0.387	0.325
35	0.039	0.079	0.050	0.031	-0.102	-0.090
36	0.754	0.834	-0.274	-0.107	-0.316	-0.315
37	0.407	0.000	0.136	0.000	-0.195	-0.000
38	0.000	0.000	0.000	0.000	0.000	0.000
39	-0.068	0.044	0.138	0.115	0.010	-0.003
40	1.494	1.579	-0.029	0.103	-0.617	-0.832
41	1.312	1.329	0.387	0.354	-0.303	-0.293
42	1.156	1.128	0.176	0.083	-0.563	-0.523
43	0.004	0.130	-0.061	0.017	-0.175	-0.163
44	4.880	4.623	0.374	0.405	0.094	0.065
45	0.080	0.185	0.089	0.054	-0.178	-0.199
46	-0.043	0.074	0.079	0.084	-0.082	-0.053
47	0.445	0.523	0.151	0.095	-0.274	-0.278
48	2.819	2.442	0.485	0.492	-0.086	-0.067
49	0.076	0.269	-0.106	0.018	-0.135	-0.119

50	-0.002	0.083	0.027	0.004	-0.037	-0.068
51	-0.019	0.058	0.055	0.018	-0.076	-0.100
52	0.186	0.275	0.198	0.213	0.263	0.162
53	0.058	0.036	0.028	0.092	-0.135	0.006
54	0.081	0.000	0.032	0.000	-0.096	0.000
55	0.000	0.000	0.000	0.000	0.000	0.000
56	-0.031	0.058	0.029	0.021	-0.089	-0.107
57	0.093	0.259	0.163	0.152	0.032	-0.072
58	0.414	0.513	0.189	0.205	0.330	0.205
59	5.000	5.000	0.000	0.000	0.000	0.000
60	0.000	0.000	0.000	0.000	0.000	0.000
61	-0.043	0.052	0.064	0.019	-0.084	-0.094
62	0.160	0.275	0.167	0.166	0.160	0.029
63	0.454	0.628	0.056	-0.013	-0.297	-0.423
64	0.185	0.453	0.144	0.255	-0.061	-0.243
65	2.998	2.341	0.193	0.347	0.624	0.495
66	-0.020	0.075	0.186	0.137	0.115	0.036
67	0.031	0.131	0.051	0.033	-0.120	-0.163
68	0.258	0.297	0.242	0.157	0.224	0.046
69	0.059	0.207	0.151	0.122	-0.005	-0.105
70	-0.026	0.062	0.179	0.133	0.078	0.013
71	2.515	2.011	0.421	0.424	0.275	0.055
72	1.025	0.903	0.270	0.241	0.759	0.461
73	0.216	0.000	0.112	-0.000	-0.132	-0.000
74	-0.013	0.063	-0.025	0.012	-0.119	-0.117
75	0.351	0.853	0.116	0.218	-0.190	-0.401
76	1.143	1.138	0.217	0.142	-0.566	-0.543
77	3.043	2.740	0.630	0.770	-0.822	-0.763
78	0.093	0.215	-0.081	0.024	-0.141	-0.136
79	0.121	0.214	-0.028	0.029	-0.107	-0.099
80	0.346	0.417	0.191	0.133	-0.194	-0.207
81	0.816	0.706	0.321	0.305	0.762	0.489
82	0.392	0.500	0.235	0.215	0.001	-0.108
83	-0.024	0.067	0.019	0.011	-0.036	-0.070
84	0.113	0.198	0.110	0.076	-0.159	-0.170
85	0.002	0.120	0.096	0.082	-0.067	-0.102
86	0.056	0.243	-0.114	0.040	-0.142	-0.119
87	2.404	1.911	0.399	0.418	0.504	0.359
88	0.288	0.412	0.199	0.189	-0.033	-0.082
89	0.191	0.251	0.308	0.247	0.346	0.181
90	-0.219	0.035	-0.132	-0.017	0.032	0.016
91	-0.045	0.087	-0.019	0.006	-0.158	-0.121
92	0.061	0.249	-0.127	0.038	-0.119	-0.116
93	-0.033	0.080	-0.040	0.011	-0.157	-0.135
94	0.061	0.159	0.196	0.144	0.111	-0.007
95	0.056	0.113	0.017	0.017	-0.083	-0.098
96	0.251	0.389	0.170	0.184	0.158	0.022

Table 3.22 Displacement Solutions for Problem 8

Element	*u1 (mm)*	*U1 (mm)*	*u2 (mm)*	*U2 (mm)*	*u3 (mm)*	*U3 (mm)*
1	1.325	1.084	3.807	3.250	-0.863	-0.857
2	0.225	0.156	0.642	1.196	-0.402	-0.725
3	-0.071	-0.160	0.343	0.818	-0.218	-0.329
4	0.028	0.117	0.797	0.850	-0.280	-0.363
5	-0.173	0.043	-0.169	-0.168	0.029	-0.004
6	0.763	0.562	3.842	2.914	-0.731	-0.463
7	0.156	0.124	0.449	0.461	-0.366	-0.278
8	0.441	0.317	0.917	0.753	-0.607	-0.468
9	0.497	0.552	0.507	0.511	-0.987	-0.866
10	0.181	0.254	0.280	0.233	-0.559	-0.454
11	-0.111	-0.072	-0.125	0.298	-0.081	-0.166
12	0.650	0.483	2.110	2.021	-0.113	-0.036
13	0.489	0.584	0.403	0.392	-1.044	-0.912
14	-0.601	-0.392	0.086	0.173	-0.211	-0.027
15	0.586	0.608	0.567	0.475	-1.098	-0.967
16	-0.671	-0.442	-0.263	0.114	-0.374	-0.074
17	0.248	0.303	0.168	0.173	-0.530	-0.445
18	0.781	0.414	4.384	3.793	-0.405	-0.144
19	0.147	-0.034	0.429	0.612	-0.208	-0.186
20	0.869	0.644	1.535	1.524	-1.078	-1.068
21	0.306	0.412	0.756	0.771	-0.654	-0.740
22	-0.079	0.026	0.284	0.232	-0.260	-0.215
23	-0.334	-0.228	0.146	0.119	-0.141	-0.033
24	0.638	0.697	0.728	0.673	-1.193	-1.133
25	-0.650	-0.544	0.397	0.429	0.031	0.038
26	0.487	0.415	0.678	0.502	-0.953	-0.696
27	-0.206	-0.081	0.075	0.097	-0.154	-0.087
28	0.213	-0.000	0.738	0.000	-0.420	-0.000
29	-0.454	-0.368	0.317	0.370	-0.153	-0.030
30	-0.326	-0.223	-0.181	0.016	-0.076	0.015
31	-0.618	-0.368	-0.423	0.005	-0.384	-0.042
32	1.336	1.125	4.292	3.865	-0.791	-0.825
33	0.607	0.567	1.247	1.364	-0.978	-1.071
34	-0.025	-0.091	3.276	3.338	0.480	0.393
35	-0.055	0.030	0.245	0.230	-0.277	-0.196
36	0.100	-0.010	-0.931	-0.133	-0.231	0.052
37	0.402	0.000	0.855	0.000	-0.593	-0.000
38	0.000	0.000	0.000	0.000	0.000	0.000
39	-0.105	0.047	0.402	0.412	-0.276	-0.290
40	0.556	0.576	1.282	1.488	-0.365	-0.472
41	1.083	0.880	1.600	1.774	-1.044	-1.045
42	0.725	0.581	1.213	1.354	-0.787	-0.741
43	-0.416	-0.256	-0.238	-0.016	-0.104	0.038
44	0.694	0.494	4.581	4.484	-0.216	-0.118
45	0.405	0.439	0.199	0.150	-0.790	-0.589
46	-0.521	-0.298	0.516	0.527	-0.121	-0.043
47	0.419	0.345	0.901	0.804	-0.635	-0.595
48	1.433	1.173	3.071	3.281	-1.104	-1.007
49	-0.793	-0.513	-0.710	0.059	-0.542	-0.140

50	0.078	0.191	-0.064	-0.071	-0.034	-0.113
51	0.098	0.123	-0.003	-0.020	-0.279	-0.193
52	0.389	0.356	0.947	0.968	-0.669	-0.675
53	0.098	0.104	0.364	0.619	0.023	0.015
54	-0.090	-0.000	0.296	0.000	-0.200	-0.000
55	0.000	0.000	0.000	0.000	0.000	0.000
56	-0.124	-0.057	-0.111	-0.116	-0.006	0.022
57	0.532	0.660	0.577	0.600	-1.032	-1.063
58	0.530	0.463	1.226	1.301	-0.707	-0.857
59	0.000	0.000	5.000	5.000	0.000	0.000
60	0.000	0.000	0.000	0.000	0.000	0.000
61	-0.058	-0.001	0.028	-0.020	-0.179	-0.067
62	0.510	0.601	0.825	0.776	-1.003	-1.038
63	0.386	0.278	0.799	0.979	-0.573	-0.731
64	0.213	0.224	0.738	1.351	-0.420	-0.708
65	0.858	0.818	3.745	3.133	-0.615	-0.450
66	0.118	0.210	0.488	0.425	-0.463	-0.466
67	0.280	0.371	0.044	0.071	-0.554	-0.497
68	-0.171	-0.217	1.421	1.421	0.150	0.185
69	0.474	0.597	0.416	0.393	-0.932	-0.903
70	-0.035	-0.033	0.594	0.543	-0.282	-0.224
71	1.615	1.148	3.155	2.739	-1.192	-1.174
72	0.810	0.566	1.943	1.739	-0.741	-0.747
73	0.232	-0.000	0.639	-0.000	-0.388	-0.000
74	-0.231	-0.127	-0.114	-0.009	-0.065	-0.004
75	0.351	0.453	0.749	1.633	-0.523	-1.037
76	0.822	0.685	1.189	1.323	-0.931	-0.935
77	0.935	1.014	1.940	2.573	-0.755	-0.764
78	-0.355	-0.306	-0.329	0.025	-0.095	-0.025
79	-0.182	-0.191	-0.033	0.180	-0.093	-0.071
80	0.541	0.413	0.793	0.667	-0.935	-0.711
81	0.578	0.362	1.776	1.570	-0.318	-0.312
82	0.689	0.605	1.162	1.040	-1.107	-1.002
83	-0.063	0.057	-0.142	-0.153	0.063	0.006
84	0.224	0.287	0.343	0.275	-0.621	-0.493
85	0.281	0.404	0.190	0.236	-0.651	-0.644
86	-0.732	-0.443	-0.801	-0.096	-0.490	-0.104
87	1.668	1.145	3.286	2.803	-1.223	-1.093
88	0.653	0.691	0.937	0.945	-1.188	-1.154
89	0.071	0.025	1.238	1.092	-0.097	-0.138
90	0.106	-0.009	-1.082	-0.240	-0.412	-0.061
91	-0.454	-0.260	-0.030	0.025	-0.172	0.004
92	-0.643	-0.404	-0.940	-0.111	-0.403	-0.081
93	-0.313	-0.183	-0.178	-0.044	-0.083	0.037
94	-0.238	-0.235	1.063	0.937	-0.012	0.015
95	-0.019	0.097	0.130	0.137	-0.197	-0.213
96	0.595	0.664	1.040	1.057	-1.108	-1.169

Table 3.23 Displacement Solutions for Problem 9

Element	*u1 (mm)*	*U1 (mm)*	*u2 (mm)*	*U2 (mm)*	*u3 (mm)*	*U3 (mm)*
1	-0.179	0.204	-0.742	-0.581	2.262	2.286
2	-0.256	-0.283	-0.233	-0.030	1.079	2.381
3	-0.206	-0.333	-0.157	0.089	0.640	1.547
4	-0.297	-0.211	-0.123	0.019	1.255	1.526
5	-0.404	-0.687	0.087	0.019	0.153	0.240
6	-0.017	-0.036	-0.243	-0.497	2.557	2.262
7	-0.201	-0.013	-0.182	-0.137	0.935	0.861
8	-0.275	-0.114	-0.458	-0.312	1.190	1.123
9	-0.280	-0.139	-0.282	-0.175	1.392	1.401
10	-0.484	-0.533	0.126	0.248	1.293	1.257
11	0.012	0.031	-0.165	-0.217	0.172	0.629
12	-0.072	0.102	-0.607	-0.342	1.821	1.978
13	-0.299	-0.217	-0.179	-0.049	1.482	1.501
14	-0.232	-0.264	-0.136	-0.087	0.335	0.401
15	-0.270	-0.126	-0.273	-0.110	1.525	1.533
16	-0.126	-0.162	-0.228	-0.188	0.057	0.338
17	-0.265	-0.162	-0.160	-0.082	0.855	0.833
18	0.533	0.571	0.130	-0.126	3.145	2.952
19	-0.262	-0.323	-0.212	0.166	0.621	1.208
20	-0.168	0.166	-0.582	-0.403	1.731	1.882
21	-0.364	-0.277	-0.078	0.080	1.607	1.801
22	-0.067	-0.003	-0.183	-0.064	0.707	0.644
23	-0.400	-0.429	0.078	0.081	0.450	0.508
24	-0.253	-0.104	-0.305	-0.130	1.677	1.767
25	-0.174	-0.170	0.067	-0.051	0.614	0.668
26	-0.226	-0.070	-0.384	-0.198	1.393	1.286
27	-0.466	-0.559	0.129	0.182	0.468	0.686
28	-0.265	0.000	-0.231	0.000	1.210	0.000
29	-0.327	-0.329	-0.013	0.038	0.539	0.651
30	-0.177	-0.203	-0.157	-0.146	0.113	0.272
31	-0.107	-0.140	-0.246	-0.194	0.004	0.256
32	-0.242	0.149	-0.420	-0.345	2.423	2.428
33	-0.151	0.138	-0.450	-0.319	1.800	1.973
34	0.051	0.104	-0.049	-0.172	2.701	2.626
35	0.036	0.044	-0.267	-0.115	0.543	0.518
36	0.001	-0.063	-0.080	0.101	-0.037	0.351
37	-0.256	0.000	-0.363	0.000	1.341	0.000
38	0.000	0.000	0.000	0.000	0.000	0.000
39	-0.465	-0.496	0.080	0.201	0.998	1.216
40	-0.374	-0.170	-0.217	-0.332	2.100	2.248
41	-0.254	0.076	-0.644	-0.531	1.810	1.989
42	-0.326	-0.080	-0.510	-0.435	1.660	1.947
43	-0.194	-0.248	-0.155	-0.144	0.053	0.218
44	0.405	0.596	0.223	0.267	3.633	3.902
45	-0.395	-0.300	-0.058	0.017	1.202	1.043
46	-0.294	-0.309	-0.111	0.067	0.676	0.881
47	-0.244	-0.052	-0.471	-0.395	1.169	1.281
48	-0.287	0.128	-0.777	-0.603	2.202	2.327
49	-0.089	-0.118	-0.405	-0.271	-0.122	0.335

50	-0.458	-0.554	0.054	0.080	0.109	0.246
51	-0.570	-0.609	0.147	0.152	0.674	0.630
52	-0.247	-0.127	-0.229	-0.037	1.670	1.825
53	0.039	-0.044	-0.017	-0.020	0.469	0.678
54	-0.190	0.000	-0.169	0.000	0.558	0.000
55	0.000	0.000	0.000	0.000	0.000	0.000
56	-0.389	-0.461	0.026	-0.010	0.109	0.188
57	-0.353	-0.200	-0.115	-0.016	1.700	1.803
58	-0.143	0.109	-0.421	-0.260	1.726	1.949
59	0.000	0.000	0.000	0.000	5.000	5.000
60	0.000	0.000	0.000	0.000	0.000	0.000
61	-0.521	-0.552	0.146	0.072	0.524	0.413
62	-0.275	-0.165	-0.214	-0.029	1.786	1.897
63	-0.314	-0.026	-0.387	-0.529	1.246	1.917
64	-0.265	-0.359	-0.231	-0.163	1.210	2.248
65	-0.109	0.181	-0.809	-0.598	2.620	2.555
66	-0.416	-0.454	0.086	0.230	1.365	1.479
67	-0.455	-0.404	0.102	0.119	1.005	0.998
68	-0.052	-0.093	-0.242	-0.098	1.461	1.514
69	-0.395	-0.291	-0.018	0.080	1.595	1.637
70	-0.381	-0.391	-0.002	0.133	1.168	1.214
71	-0.187	0.172	-0.804	-0.633	2.131	2.215
72	-0.104	0.191	-0.656	-0.393	1.964	2.084
73	-0.270	0.000	-0.231	0.000	1.056	0.000
74	-0.193	-0.229	-0.113	-0.065	0.172	0.257
75	-0.256	0.018	-0.331	-0.540	1.207	3.026
76	-0.308	-0.031	-0.564	-0.480	1.677	1.915
77	-0.537	-0.204	-0.634	-0.672	2.260	2.445
78	-0.072	-0.074	-0.216	-0.230	0.210	0.355
79	-0.113	-0.023	-0.177	-0.194	0.385	0.501
80	-0.204	0.011	-0.436	-0.293	1.404	1.336
81	-0.082	0.139	-0.554	-0.248	1.818	1.949
82	-0.154	0.094	-0.480	-0.327	1.697	1.710
83	-0.377	-0.566	0.090	0.044	0.133	0.276
84	-0.163	-0.058	-0.223	-0.096	1.048	0.968
85	-0.470	-0.434	0.097	0.177	1.255	1.379
86	-0.065	-0.086	-0.388	-0.297	-0.062	0.263
87	-0.156	0.213	-0.809	-0.599	2.076	2.177
88	-0.196	0.015	-0.390	-0.234	1.717	1.840
89	-0.173	-0.060	-0.284	-0.027	1.493	1.595
90	0.135	0.055	-0.091	0.065	-0.549	0.033
91	-0.302	-0.338	-0.070	-0.038	0.210	0.299
92	-0.037	-0.061	-0.420	-0.306	-0.050	0.292
93	-0.292	-0.349	-0.120	-0.068	0.073	0.216
94	-0.189	-0.217	-0.224	-0.045	1.228	1.204
95	-0.021	-0.018	-0.186	-0.096	0.405	0.470
96	-0.207	-0.005	-0.360	-0.214	1.840	1.966

From the above tables, one can see that the solutions obtained by using this author's code are in good agreement with the solutions obtained by using the BEM software available from [Yijun Liu, n.d.]. The differences between the displacement solutions obtained by using this author's code and the software available from [Yijun Liu, n.d.] are found to be less than *0.1 mm* in many cases, less than *0.5 mm* in most of the cases, and less than *1 mm* almost always. These differences would not have any practical significance since in general the hardware of surgical simulators cannot offer a resolution better than *1 mm*.

As already explained in Chapter 1, this is the first time that the approach that does not make use of any type of precomputations has been followed to achieve the real-time simulations, while using BEM in general and for the simulation of biological organs in particular. Since the same approach has not been followed elsewhere, there is no question of comparing the results obtained using this author's implementation of the approach with the results obtained using someone else's implementation of the same approach. Quantitative comparison of the results from the present approach with the results from other approaches is not needed in support of the present approach since just simple qualitative reasoning makes it clear that the present approach has many advantages and that it should be the preferred approach (as already elaborated while discussing the advantages of the present approach, in Chapter 1).

Although nonlinear formulations can give more realistic results when it comes to the simulation of biological organs, as already mentioned in the beginning of this chapter, present chapter concerns itself mostly about linear elastic analysis only. Coming to the question of the validity of the linear elastic material models for large deformations, there is no clear line that separates linear analysis from nonlinear analysis. Some sources in the literature mention that linear elastic analysis is valid up to an 'elongation' of *0.2%*, while some other sources utilize linear elastic formulations when the 'elongations' are *1%*, *2%*, *5%*, or even *10%*. Clearly, whether a particular simulation must make use of nonlinear formulations depends on the application, and linear elastic formulations are all right as long as the results obtained are useful and satisfactory. Again, the same type of argument is applicable when it comes to the allowable error in a simulation when the simulation employs just the linear elastostatics. Of course, literature contains many sources that use just the linear analysis for achieving the real-time simulation of biological organs while some of these sources use very few elements (finite elements or boundary elements) to describe the geometry, which would inevitably result in the loss of accuracy even for those linear elastic simulations. However, these linear elastic simulations carried out by using very few elements have not been considered to be useless, but such a simulation is assumed to be a step towards building more and more realistic simulators.

3.5 Error Estimation

In a previous section of this chapter, nine simulations were carried out on biological organs (six simulations on kidneys and three simulations on a liver). The results given by this author's code and also the results obtained by using someone else's software [Yijun Liu, n.d.] were tabulated.

Undeformed and deformed shapes of kidneys and the liver are plotted now. Figure 3.4 shows the undeformed shape of the kidney appearing in Problem 1. Figure 3.5 shows the deformed shape of the kidney for Problem 1, deformations calculated by using this author's code. Figure 3.6 shows the deformed shape of the kidney for Problem 1, deformations calculated by using the software from [Yijun Liu, n.d.].

Similarly, Figure 3.7 to Figure 3.9 show the undeformed and deformed shapes for Problem 2, Figure 3.10 to Figure 3.12 show the undeformed and deformed shapes for Problem 3 etc.

The results obtained by using this author's code are compared with the results obtained by using the software available from [Yijun Liu, n.d.] because [Yijun Liu, n.d.] was the only software that was readily available and which could simulate 3D linear elastostatics. The purpose of comparing the results is just to verify that the results from this author's code do match well with the results obtained by at least one BEM software package that is developed by someone else (even for complicated geometry like liver and kidney).

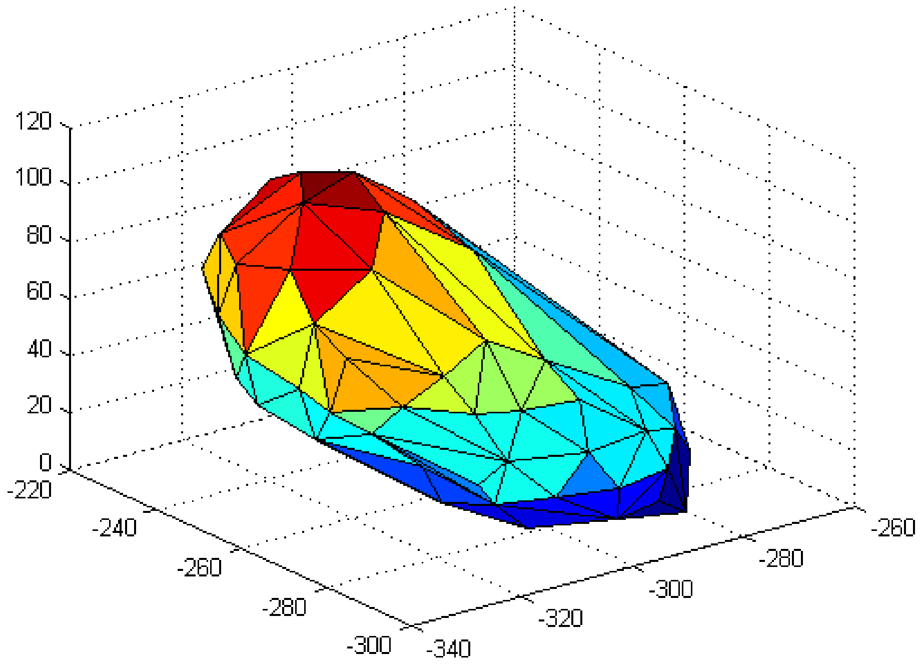

Figure 3.4 Undeformed Geometry for Problem 1

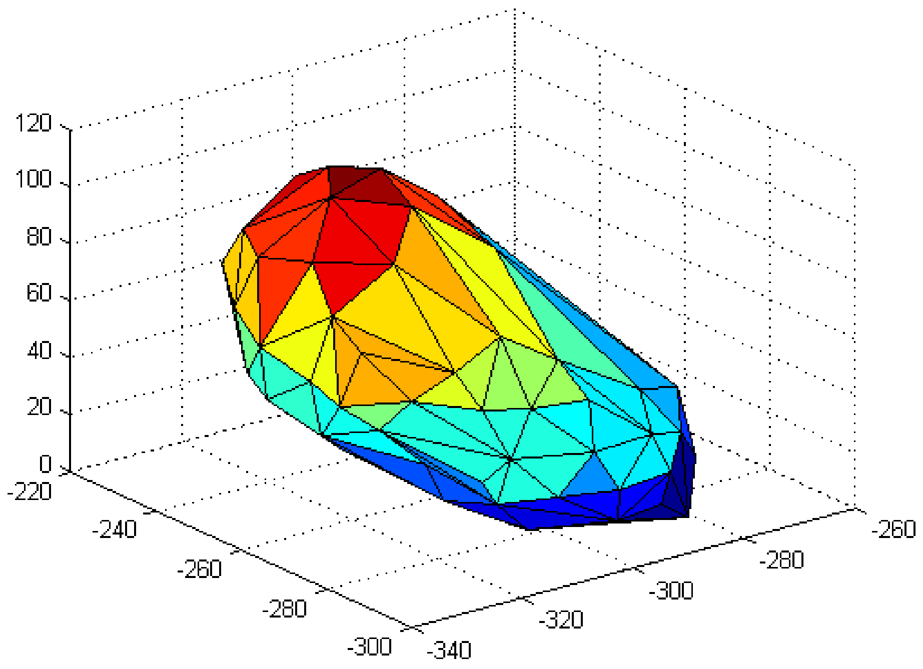

Figure 3.5 Deformed Geometry for Problem 1 (Deformations Calculated by using the Present Author's Code)

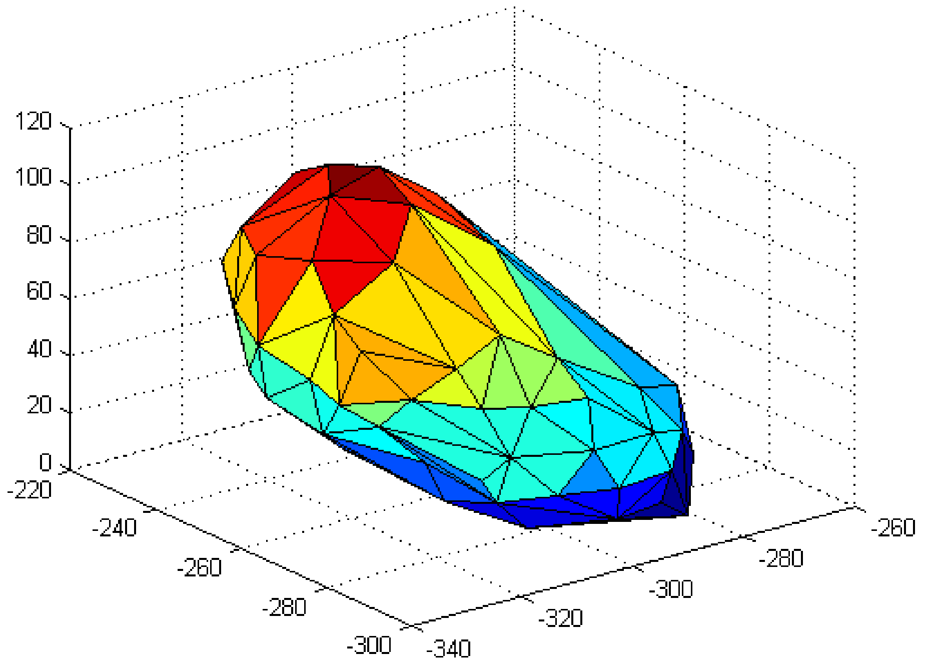

Figure 3.6 Deformed Geometry for Problem 1 (Deformations Calculated by using the Software from [Yijun Liu, n.d.])

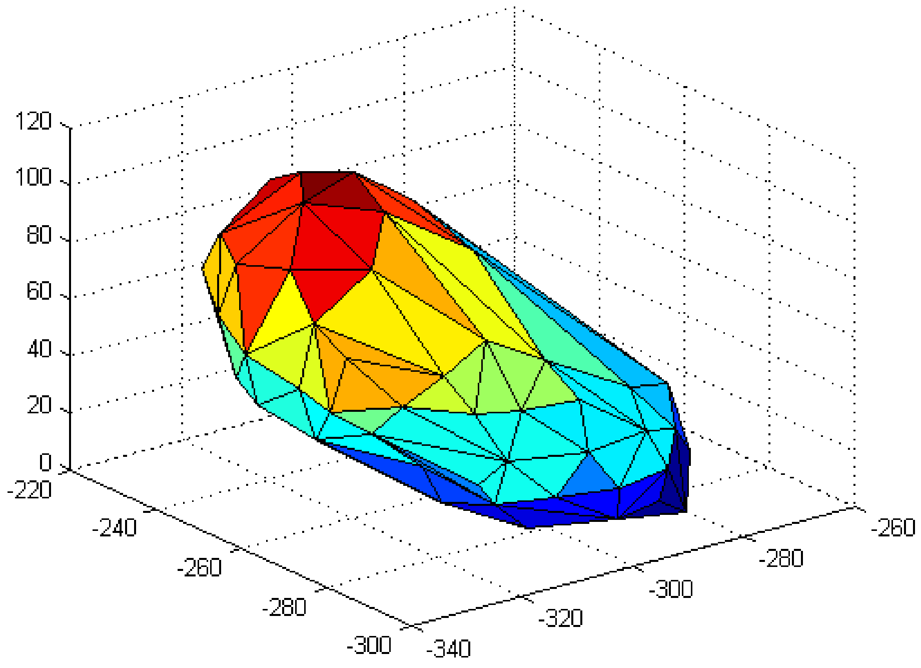

Figure 3.7 Undeformed Geometry for Problem 2

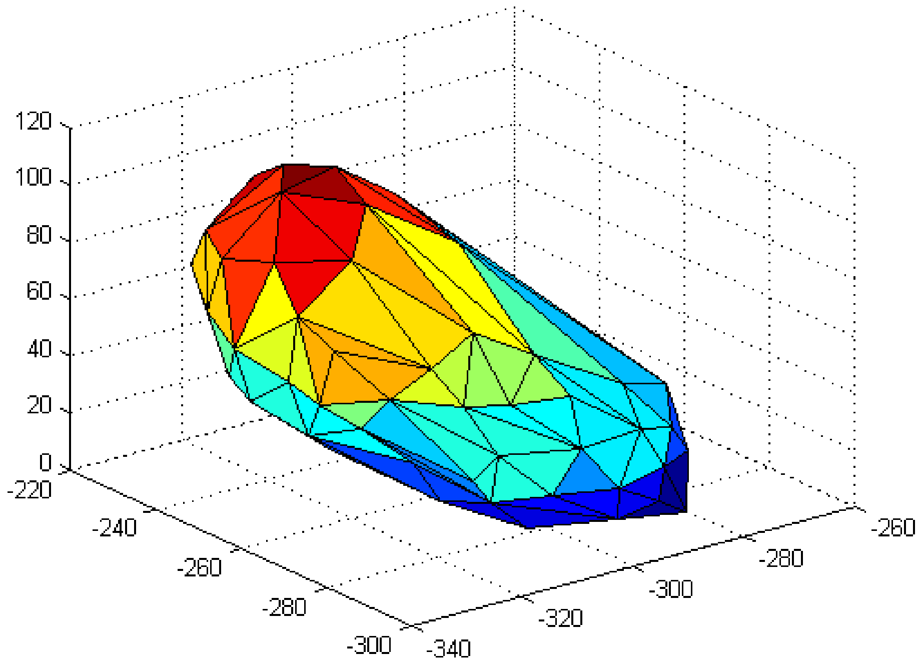

Figure 3.8 Deformed Geometry for Problem 2 (Deformations Calculated by using the Present Author's Code)

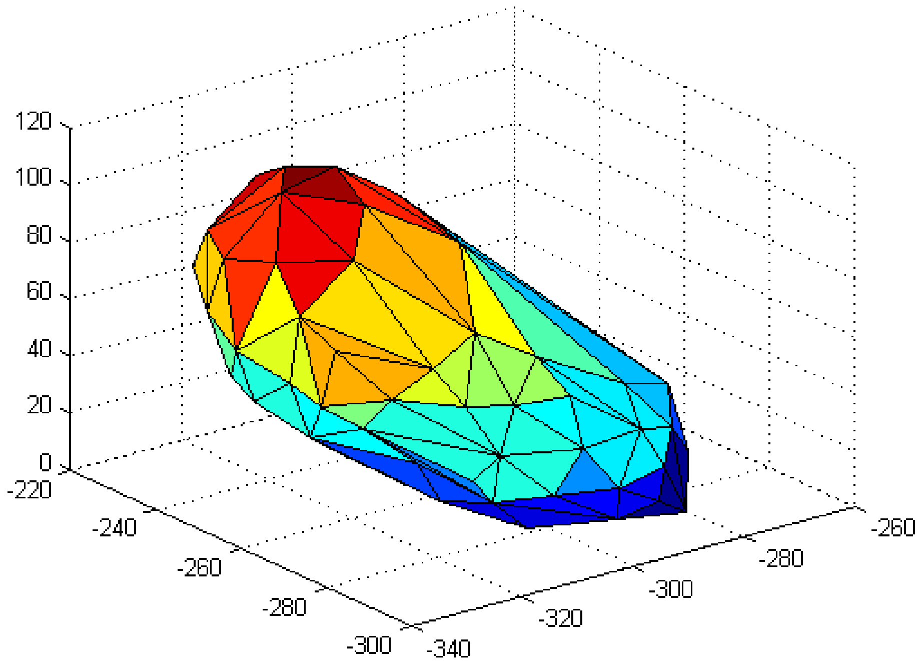

Figure 3.9 Deformed Geometry for Problem 2 (Deformations Calculated by using the Software from [Yijun Liu, n.d.])

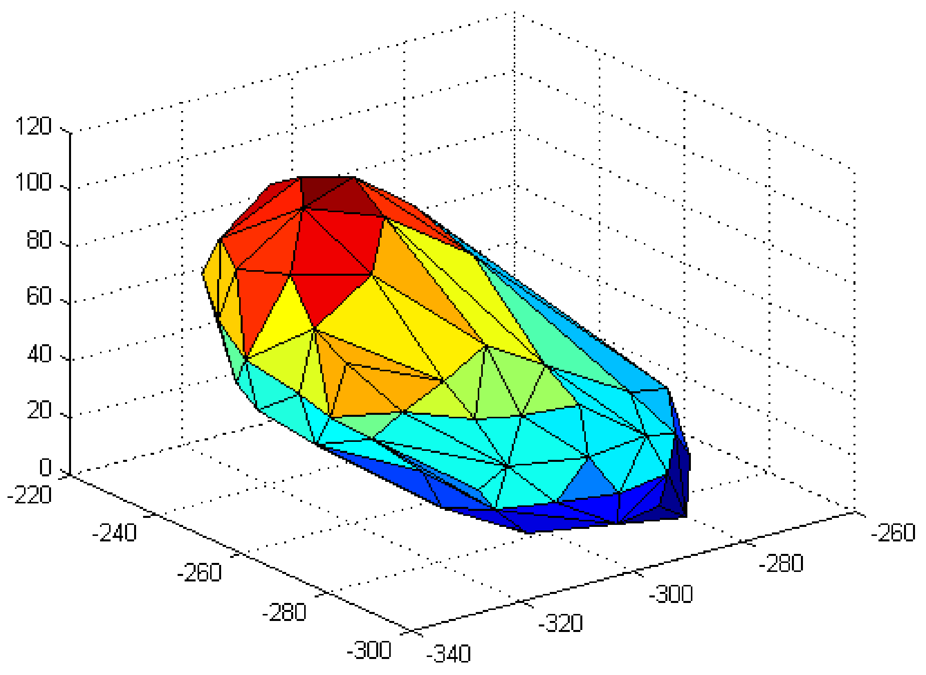

Figure 3.10 Undeformed Geometry for Problem 3

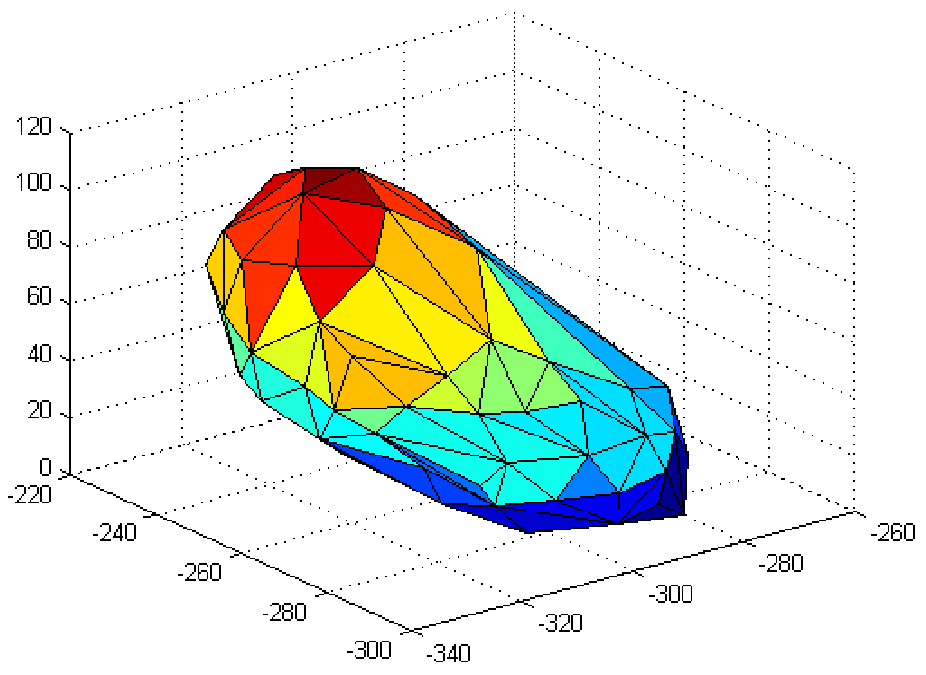

Figure 3.11 Deformed Geometry for Problem 3 (Deformations Calculated by using the Present Author's Code)

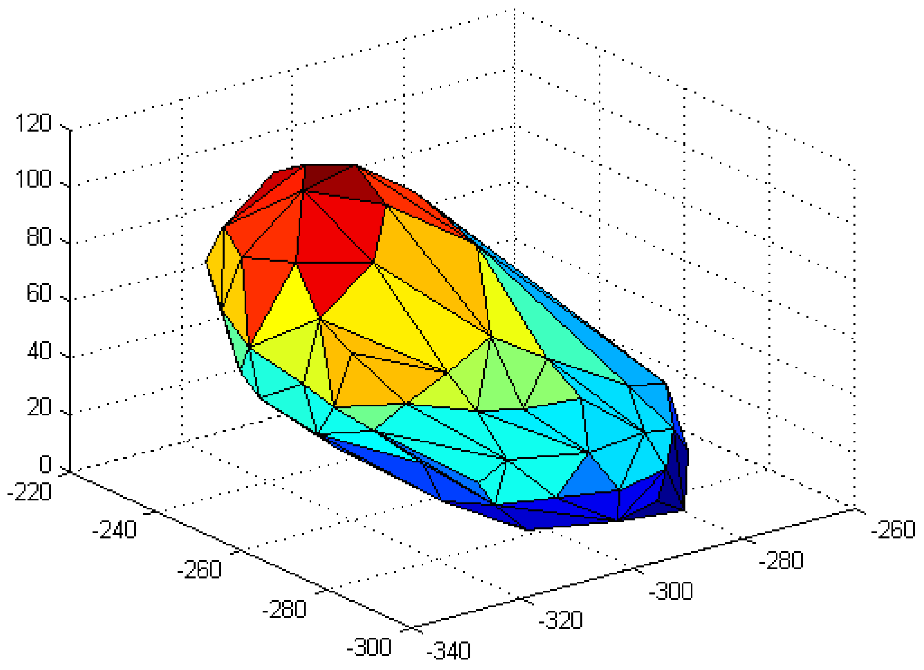

Figure 3.12 Deformed Geometry for Problem 3 (Deformations Calculated by using the Software from [Yijun Liu, n.d.])

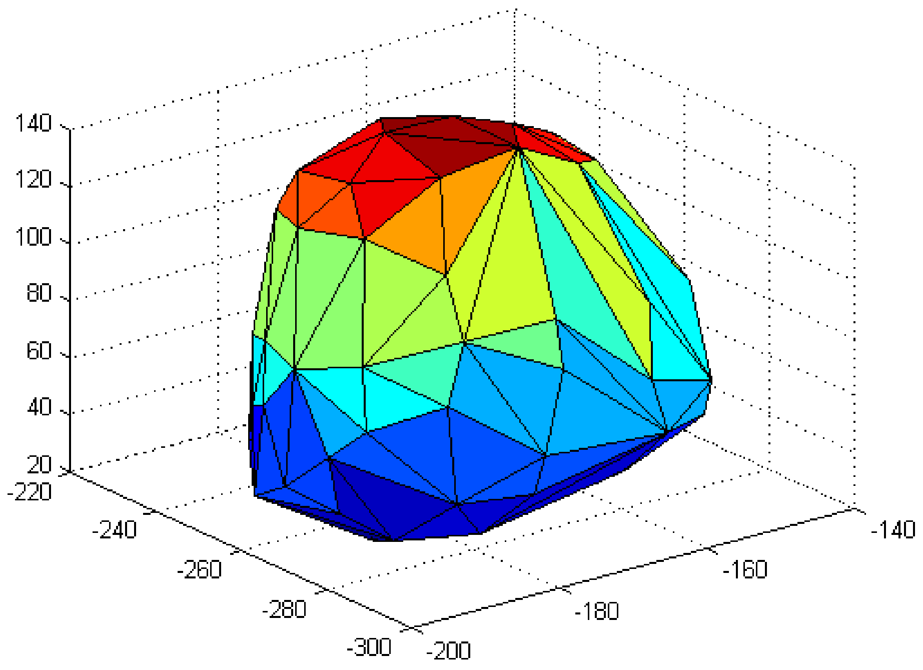

Figure 3.13 Undeformed Geometry for Problem 4

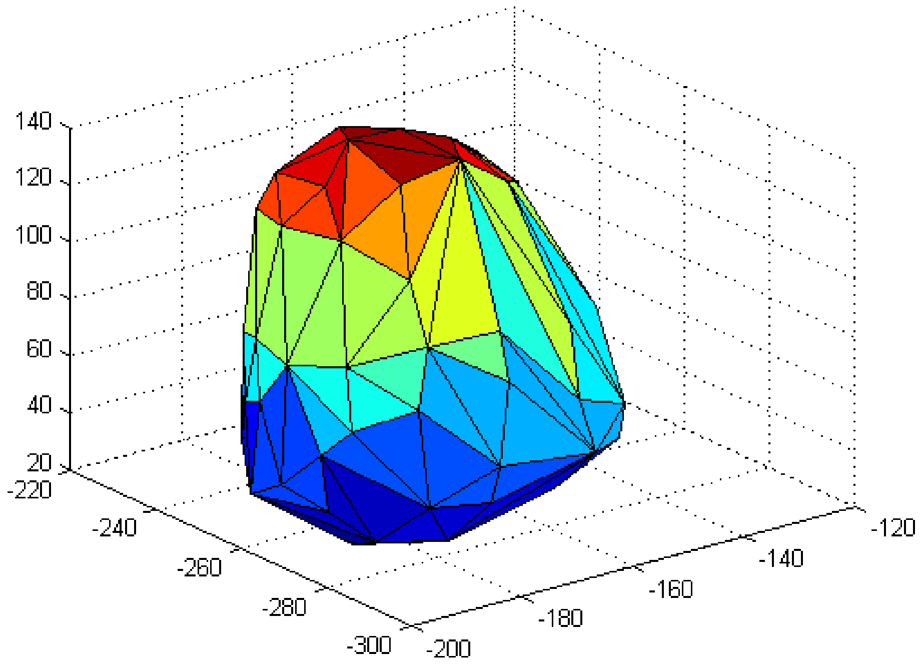

Figure 3.14 Deformed Geometry for Problem 4 (Deformations Calculated by using the Present Author's Code)

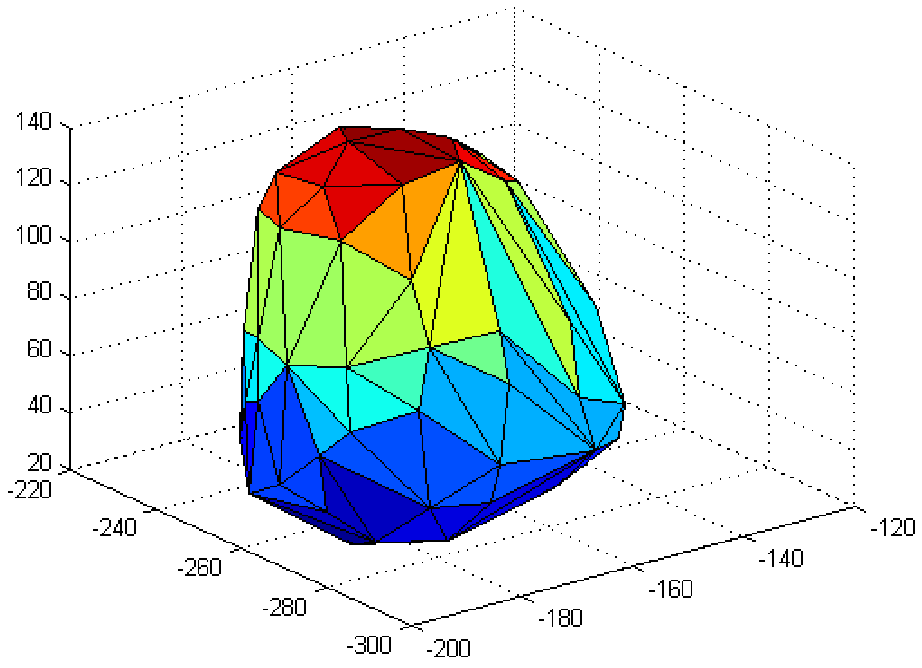

Figure 3.15 Deformed Geometry for Problem 4 (Deformations Calculated by using the Software from [Yijun Liu, n.d.])

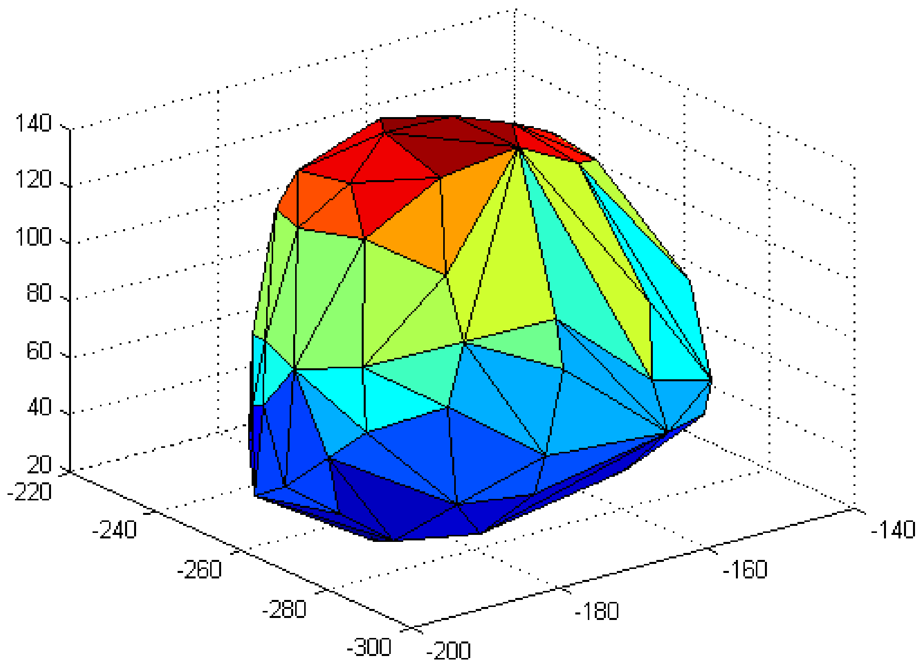

Figure 3.16 Undeformed Geometry for Problem 5

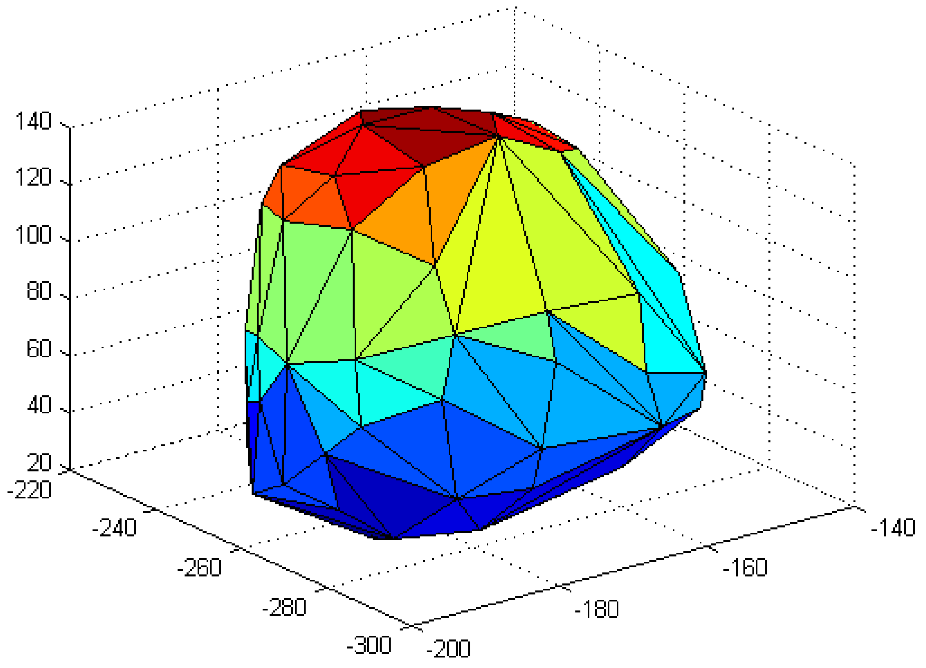

Figure 3.17 Deformed Geometry for Problem 5 (Deformations Calculated by using the Present Author's Code)

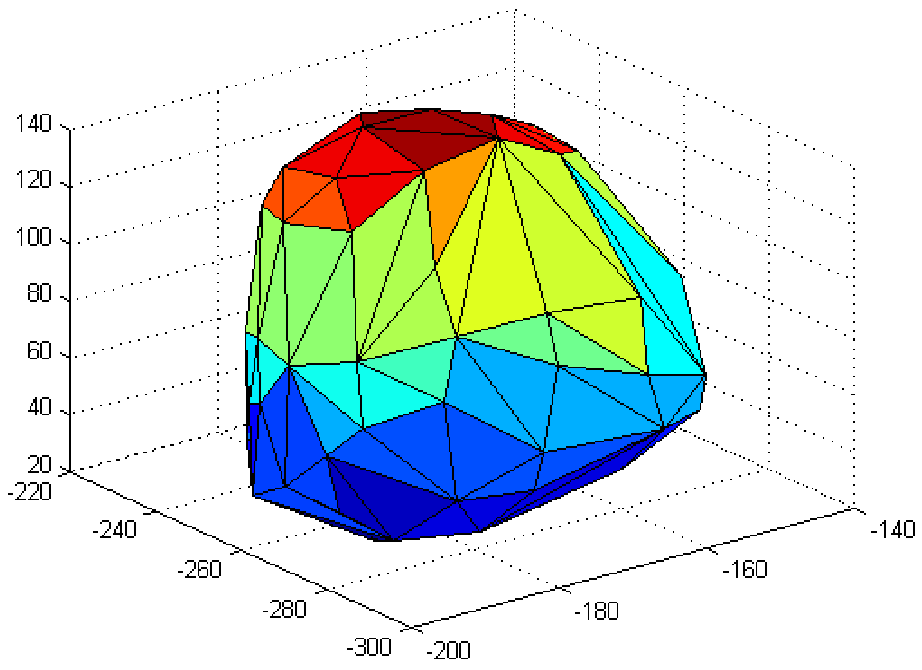

Figure 3.18 Deformed Geometry for Problem 5 (Deformations Calculated by using the Software from [Yijun Liu, n.d.])

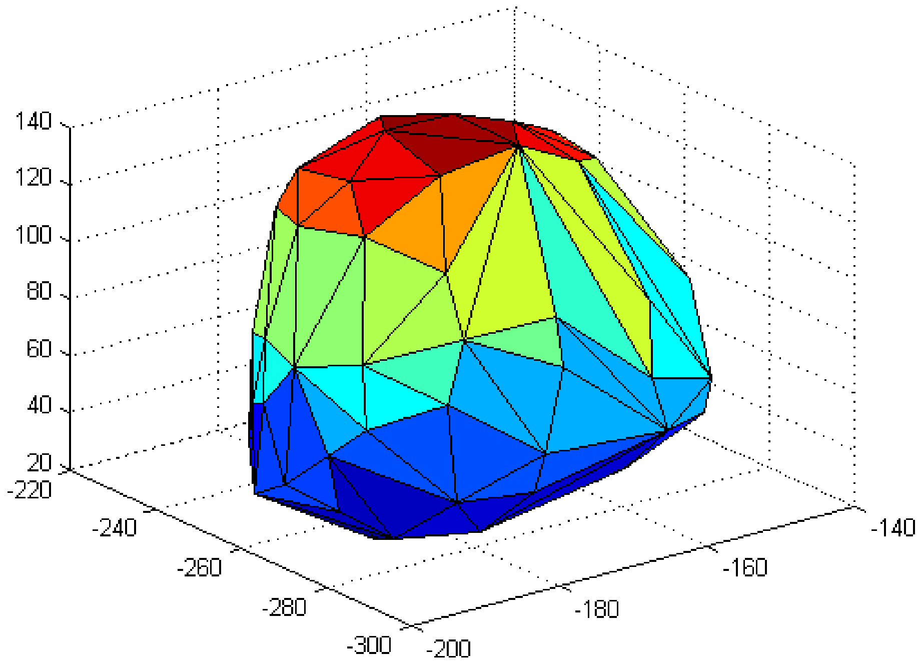

Figure 3.19 Undeformed Geometry for Problem 6

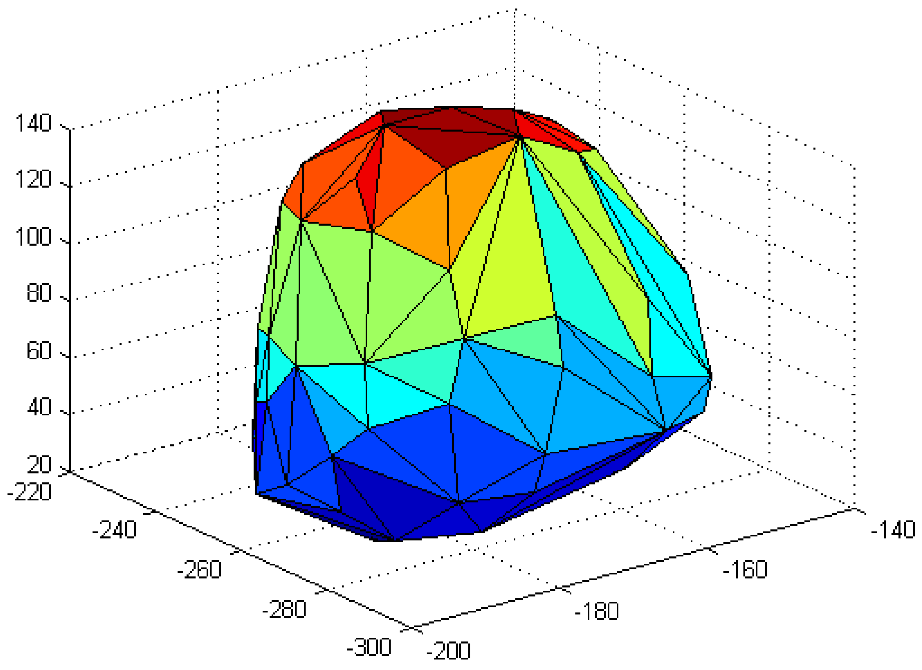

Figure 3.20 Deformed Geometry for Problem 6 (Deformations Calculated by using the Present Author's Code)

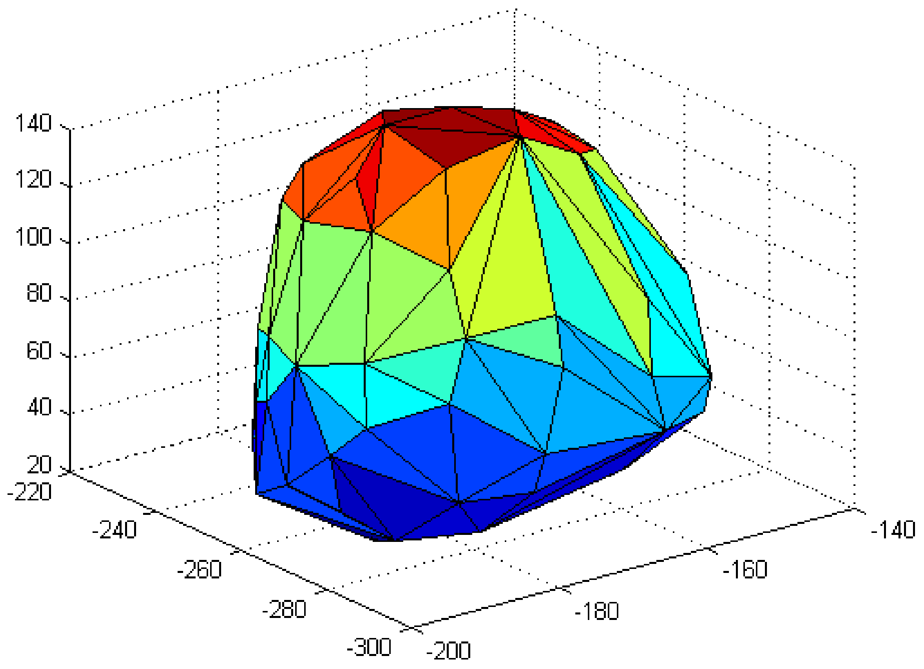

Figure 3.21 Deformed Geometry for Problem 6 (Deformations Calculated by using the Software from [Yijun Liu, n.d.])

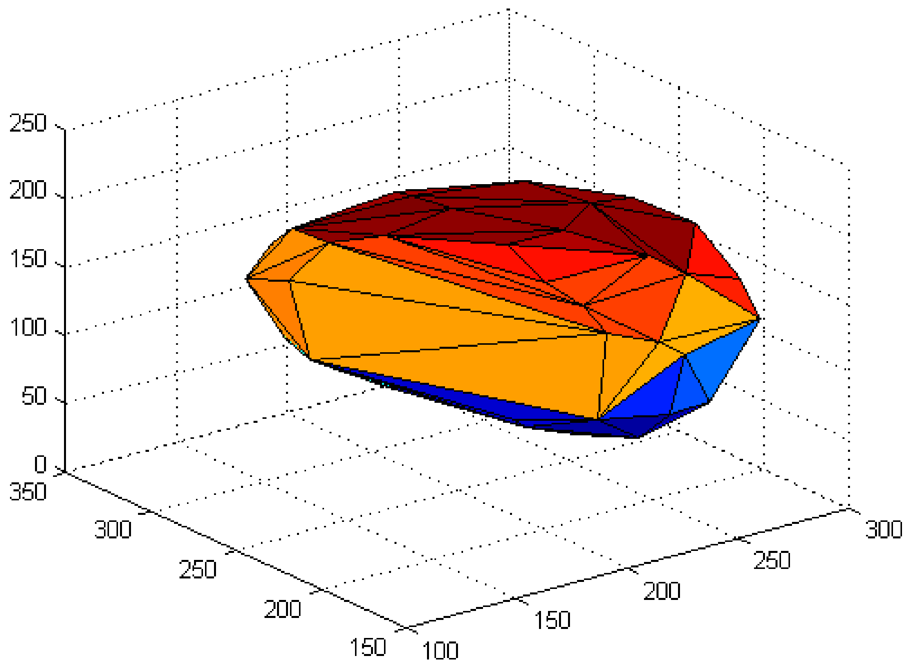

Figure 3.22 Undeformed Geometry for Problem 7

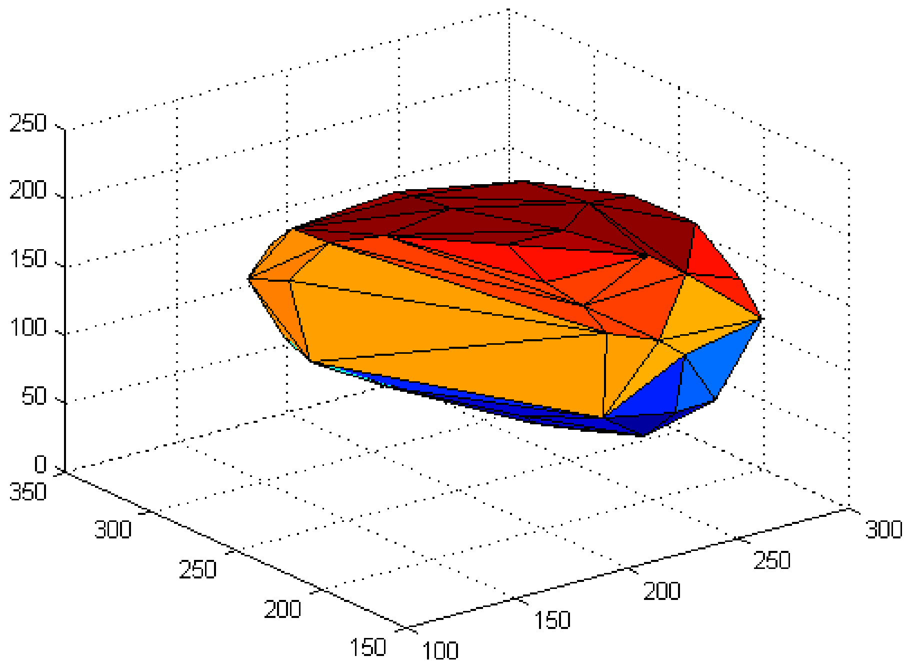

Figure 3.23 Deformed Geometry for Problem 7 (Deformations Calculated by using the Present Author's Code)

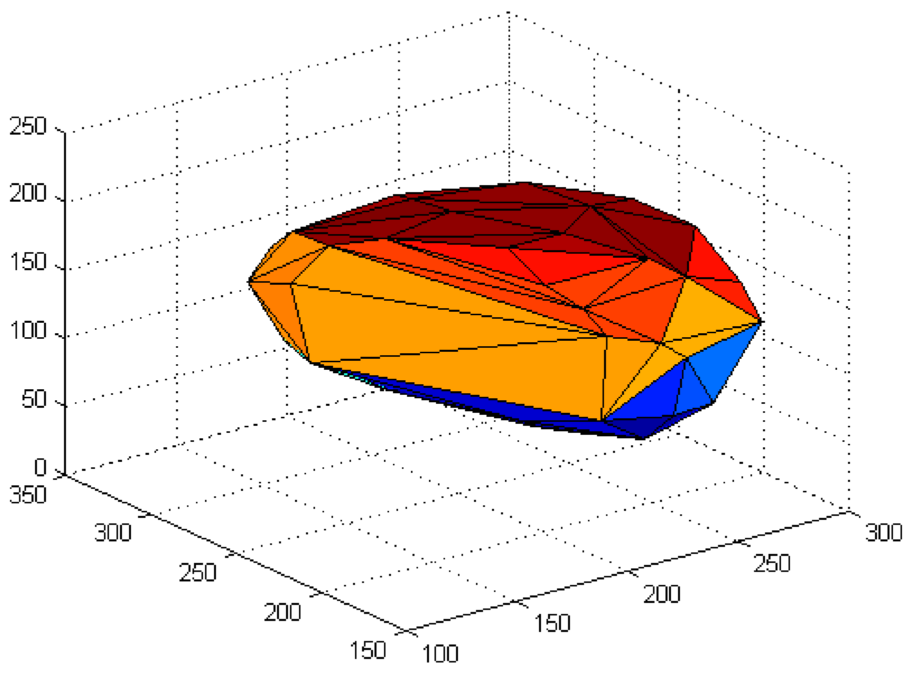

Figure 3.24 Deformed Geometry for Problem 7 (Deformations Calculated by using the Software from [Yijun Liu, n.d.])

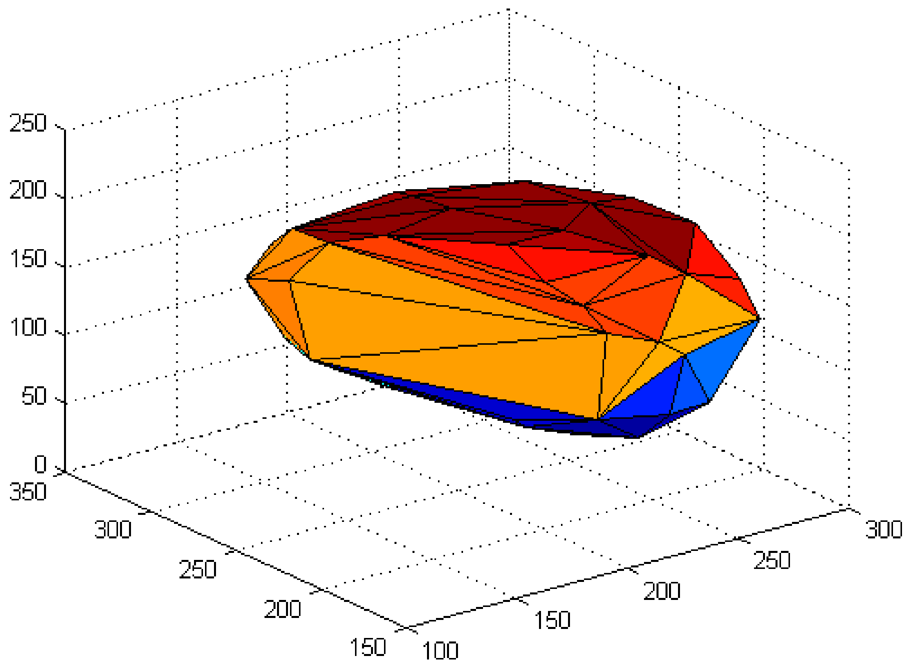

Figure 3.25 Undeformed Geometry for Problem 8

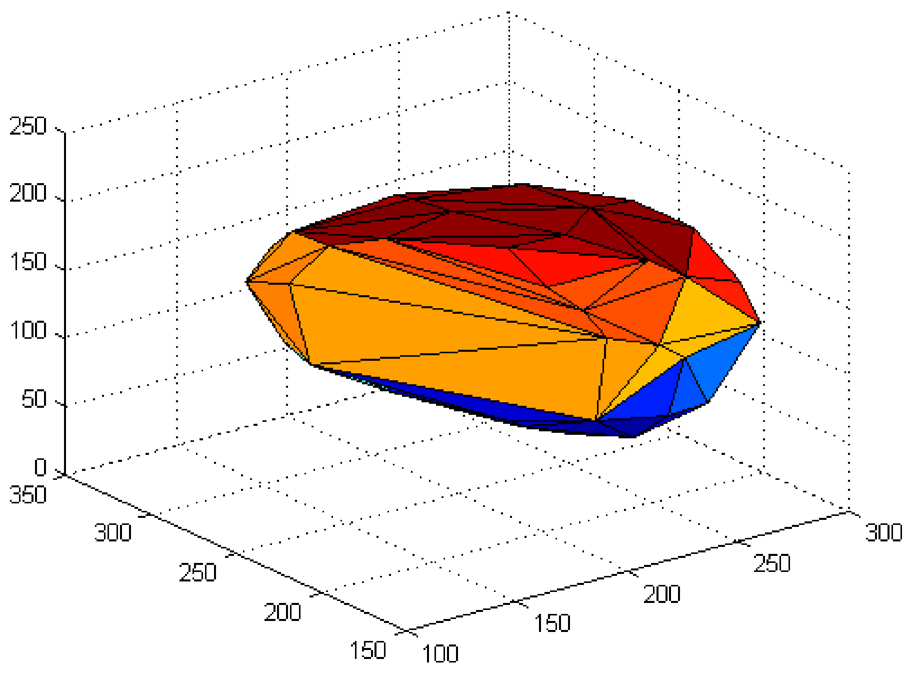

Figure 3.26 Deformed Geometry for Problem 8 (Deformations Calculated by using the Present Author's Code)

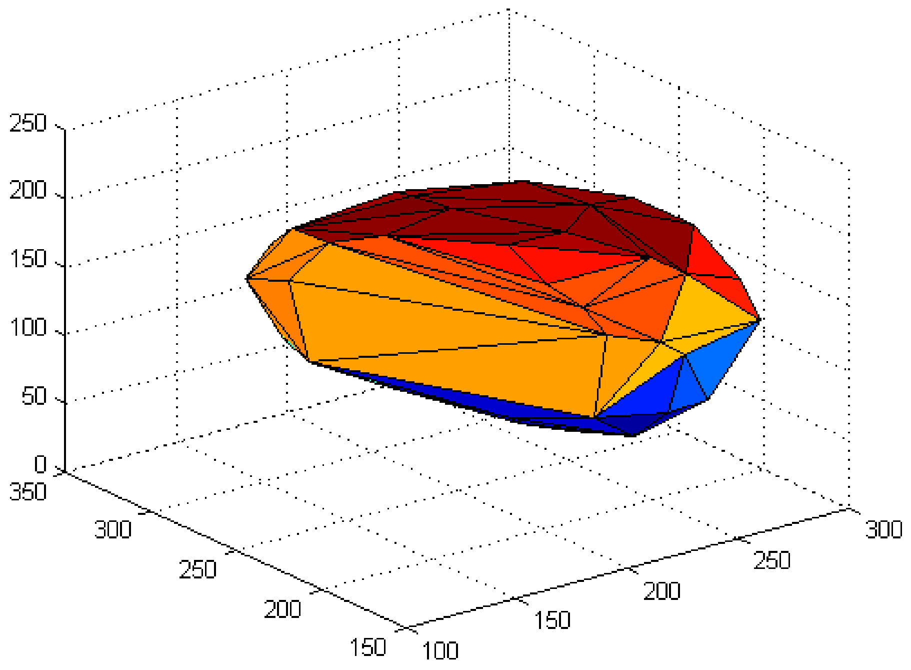

Figure 3.27 Deformed Geometry for Problem 8 (Deformations Calculated by using the Software from [Yijun Liu, n.d.])

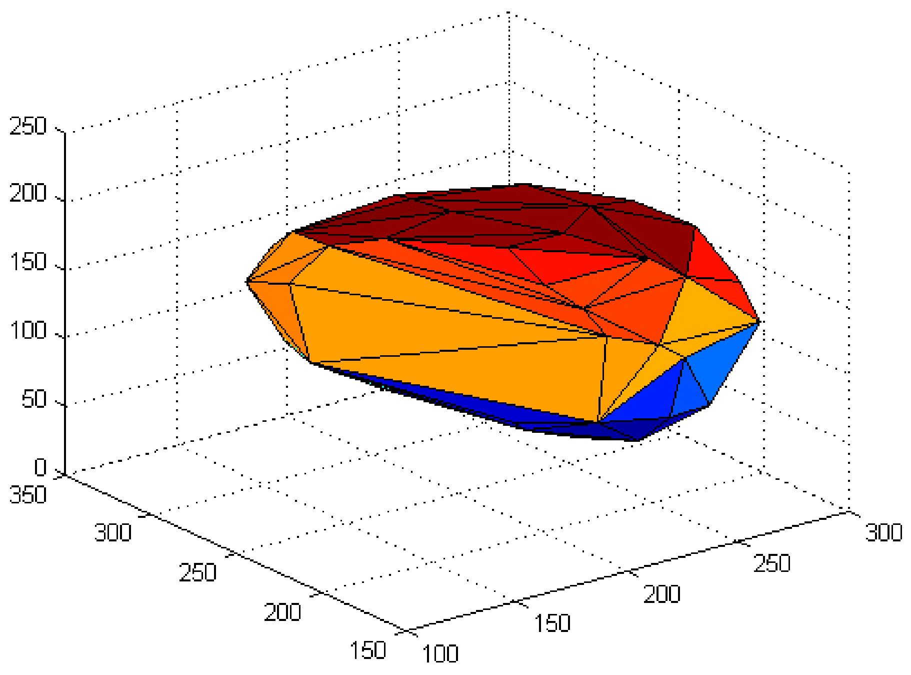

Figure 3.28 Undeformed Geometry for Problem 9

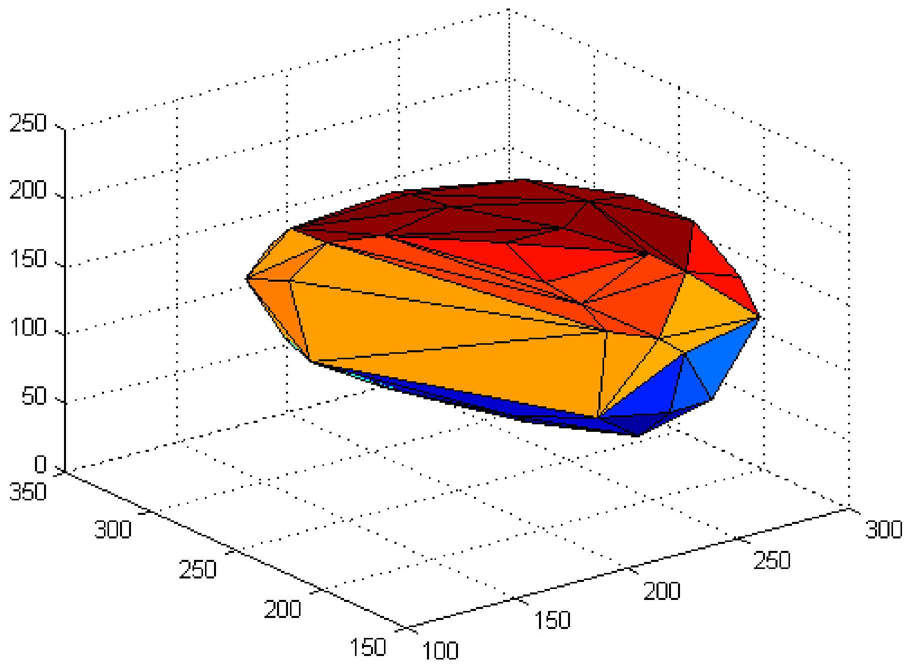

Figure 3.29 Deformed Geometry for Problem 9 (Deformations Calculated by using the Present Author's Code)

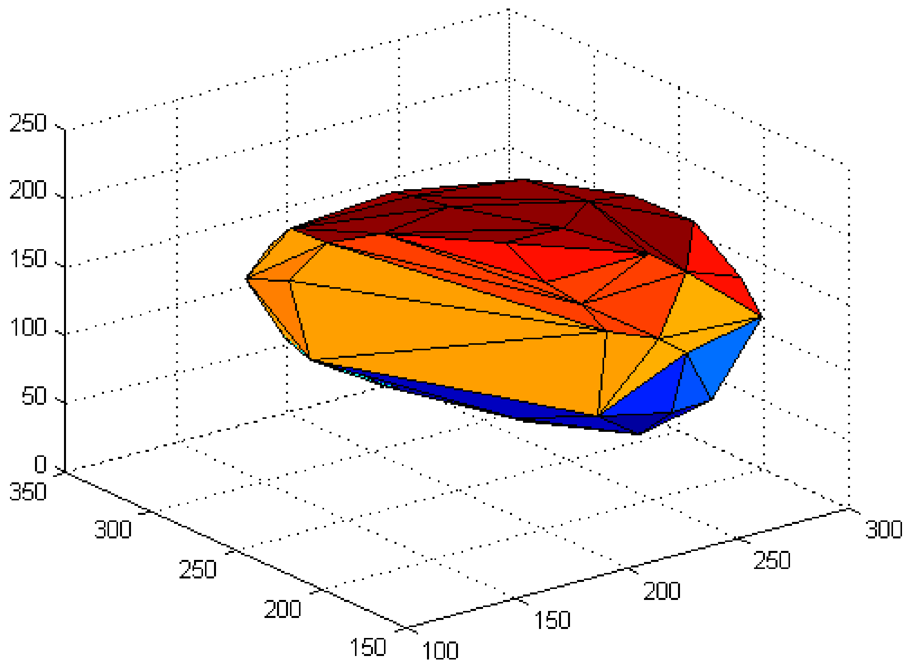

Figure 3.30 Deformed Geometry for Problem 9 (Deformations Calculated by using the Software from [Yijun Liu, n.d.])

From the figures, one can see that the deformations calculated by using this author's code are in good agreement with the deformations calculated by making use of the software from [Yijun Liu, n.d.]. The plots are generated through MATLAB, by making use of the functions 'DelaunayTri', 'convexHull', and 'trisurf'.

Just to make it clear that there is no noticeable difference between the deformed shape that is calculated by using the present author's code and the deformed shape that is calculated by using the software from [Yijun Liu, n.d.] while there is noticeable difference between the undeformed and deformed shapes, undeformed and deformed shapes are plotted in Figure 3.31 to Figure 3.57, after scaling the deformations by a factor of 4.

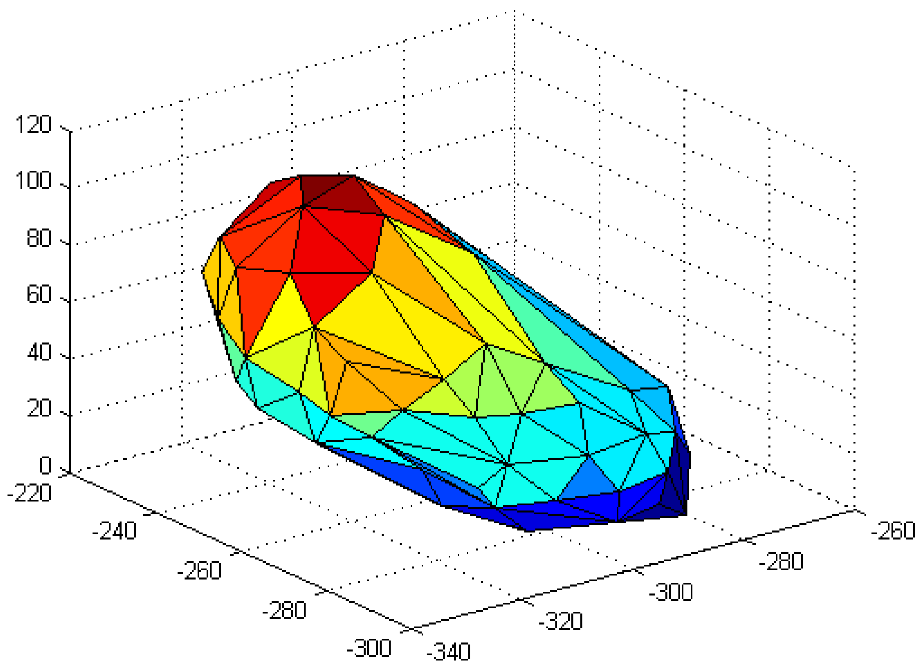

Figure 3.31 Undeformed Geometry for Problem 1

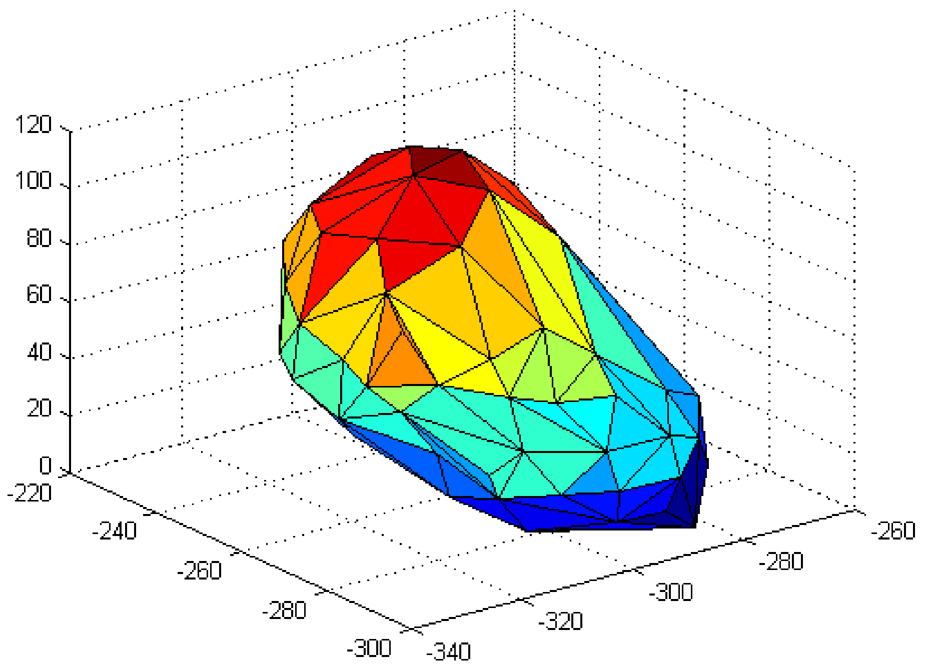

Figure 3.32 Deformed Geometry for Problem 1 (Deformations Calculated by using the Present Author's Code, and Deformations Scaled by a Factor of 4)

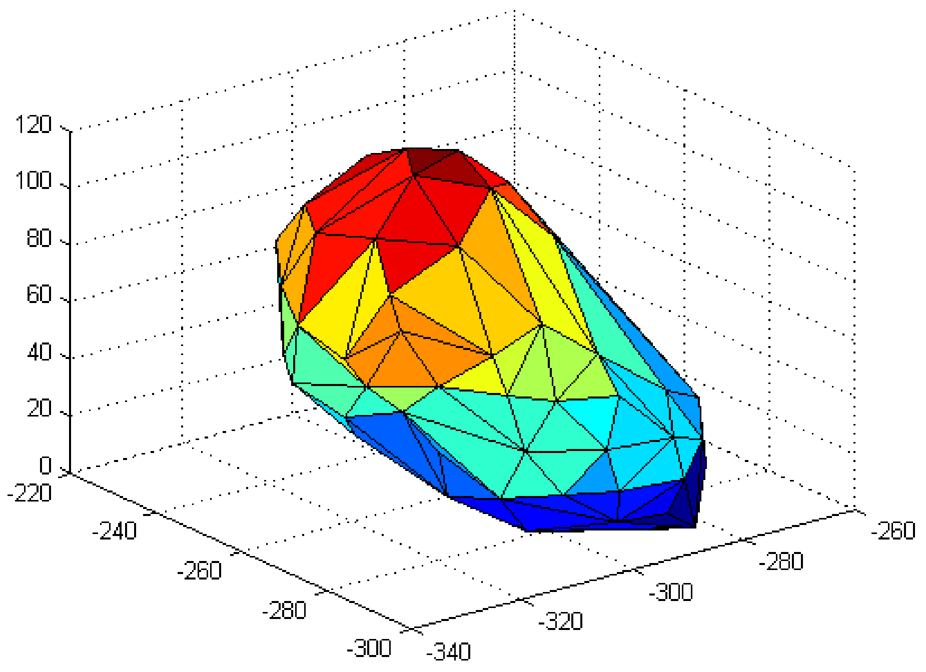

Figure 3.33 Deformed Geometry for Problem 1 (Deformations Calculated by using the Software from [Yijun Liu, n.d.], and Deformations Scaled by a Factor of 4)

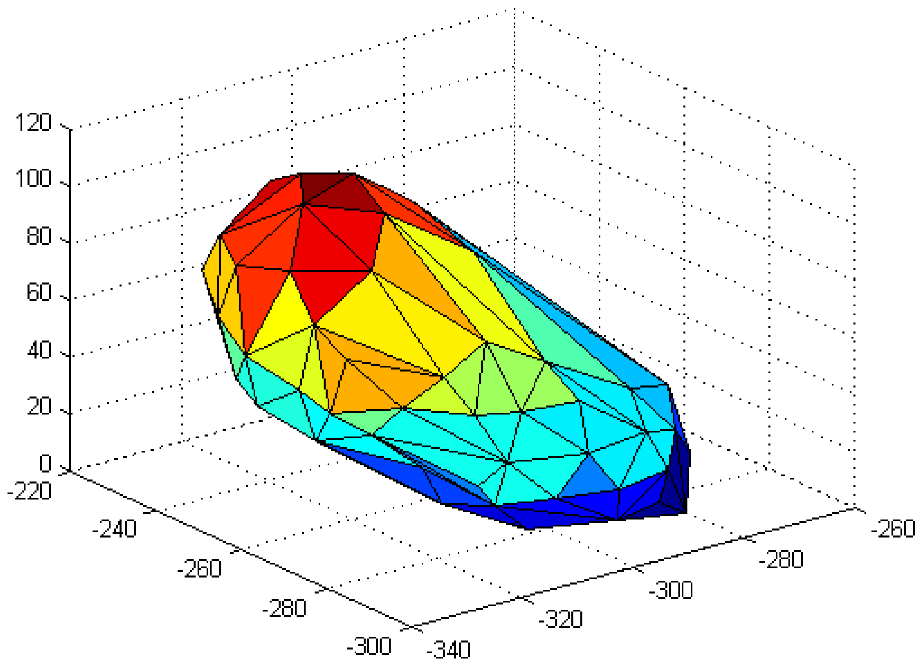

Figure 3.34 Undeformed Geometry for Problem 2

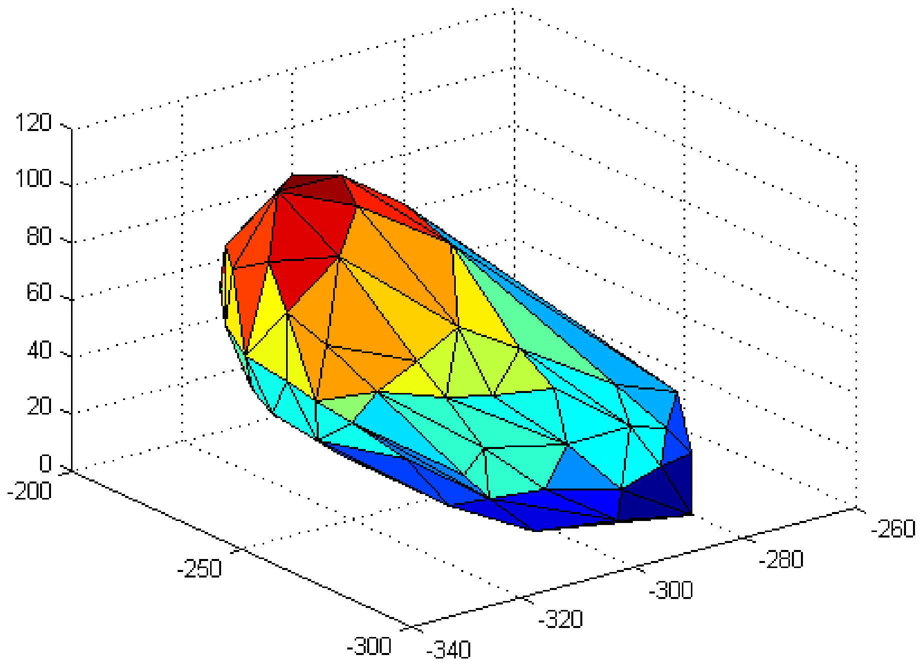

Figure 3.35 Deformed Geometry for Problem 2 (Deformations Calculated by using the Present Author's Code, and Deformations Scaled by a Factor of 4)

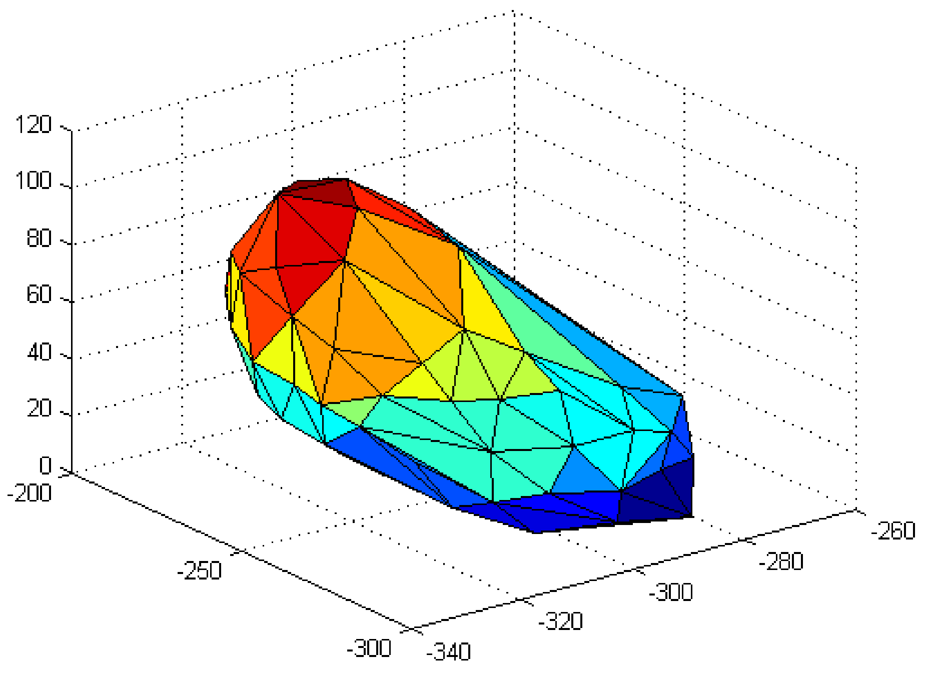

Figure 3.36 Deformed Geometry for Problem 2 (Deformations Calculated by using the Software from [Yijun Liu, n.d.], and Deformations Scaled by a Factor of 4)

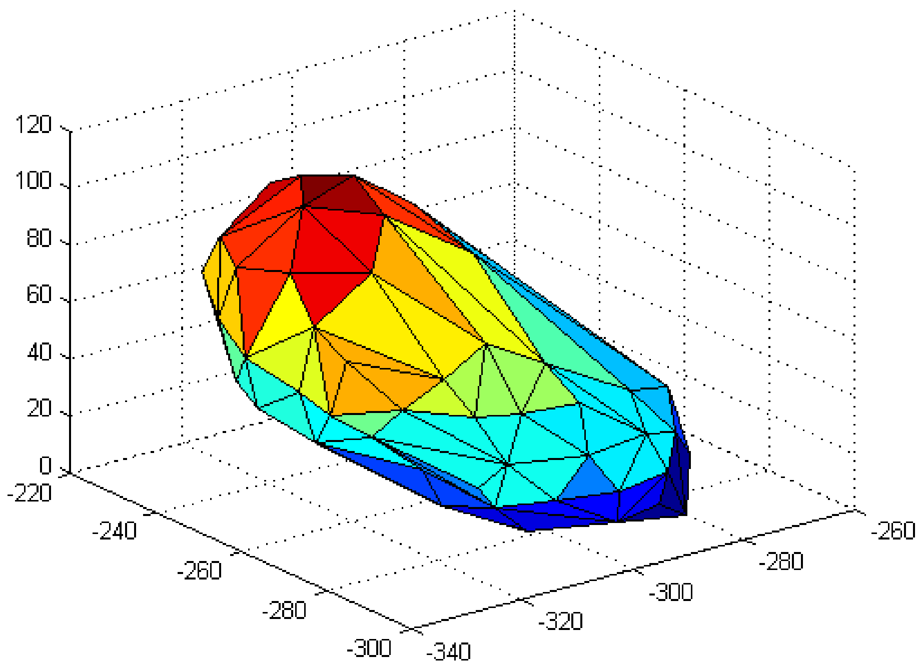

Figure 3.37 Undeformed Geometry for Problem 3

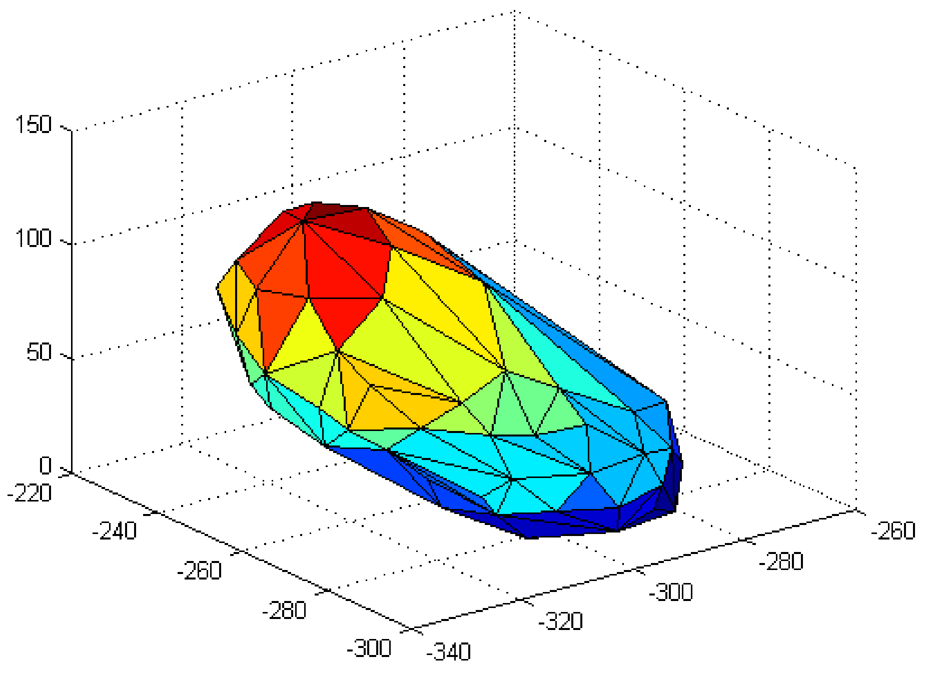

Figure 3.38 Deformed Geometry for Problem 3 (Deformations Calculated by using the Present Author's Code, and Deformations Scaled by a Factor of 4)

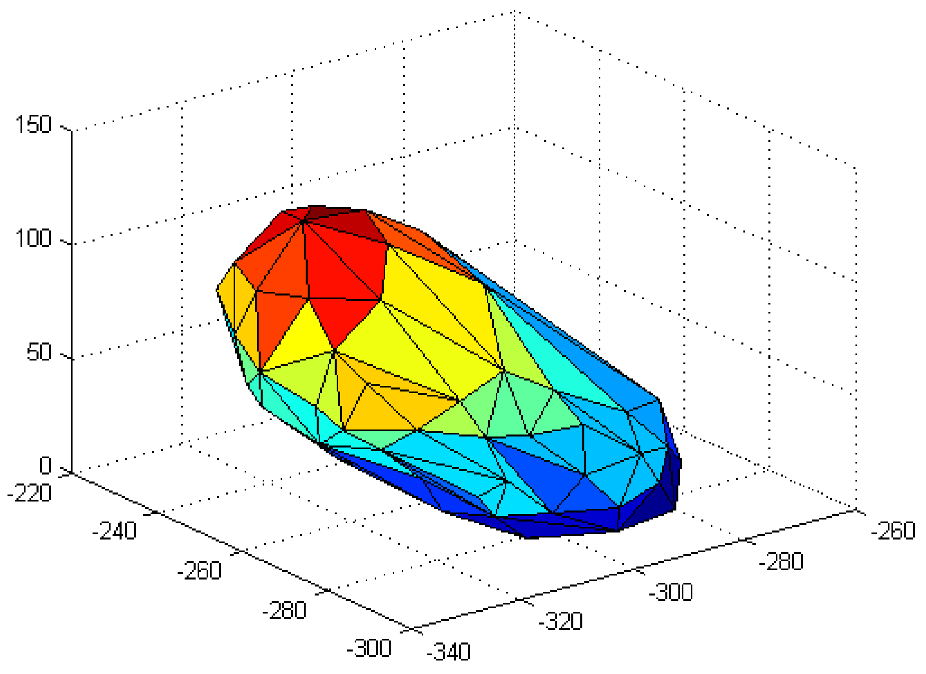

Figure 3.39 Deformed Geometry for Problem 3 (Deformations Calculated by using the Software from [Yijun Liu, n.d.], and Deformations Scaled by a Factor of 4)

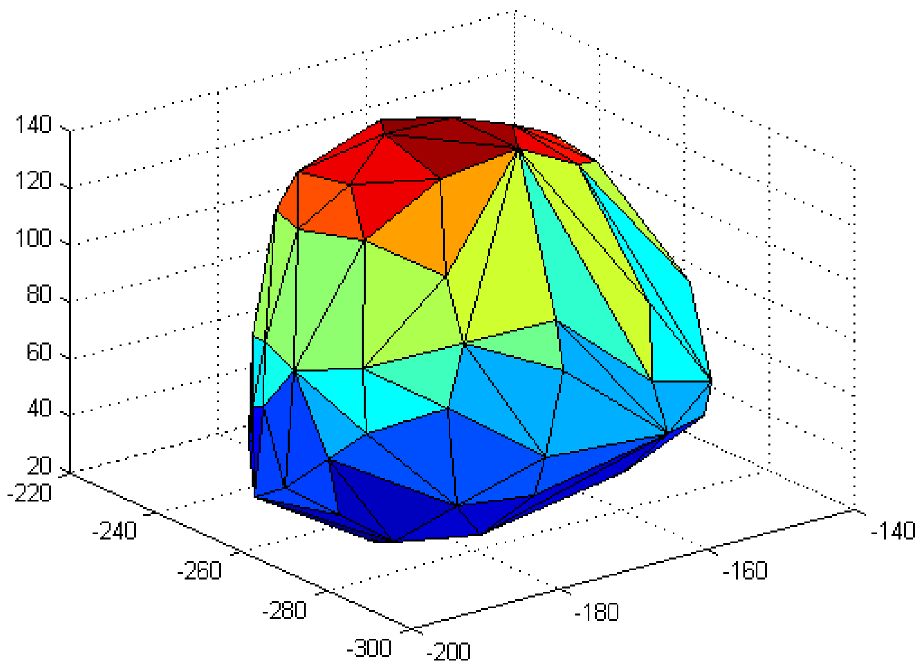

Figure 3.40 Undeformed Geometry for Problem 4

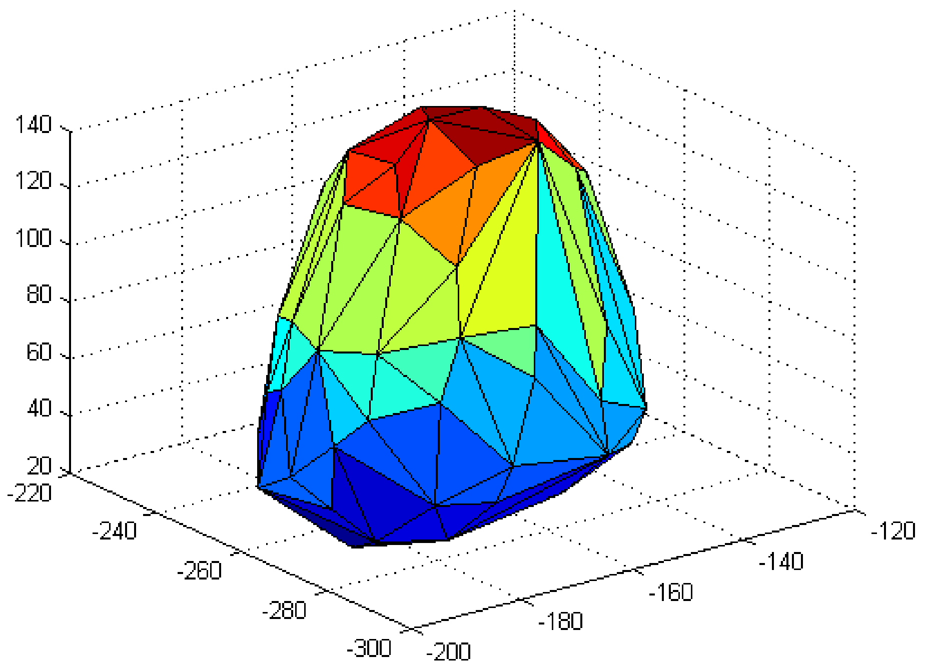

Figure 3.41 Deformed Geometry for Problem 4 (Deformations Calculated by using the Present Author's Code, and Deformations Scaled by a Factor of 4)

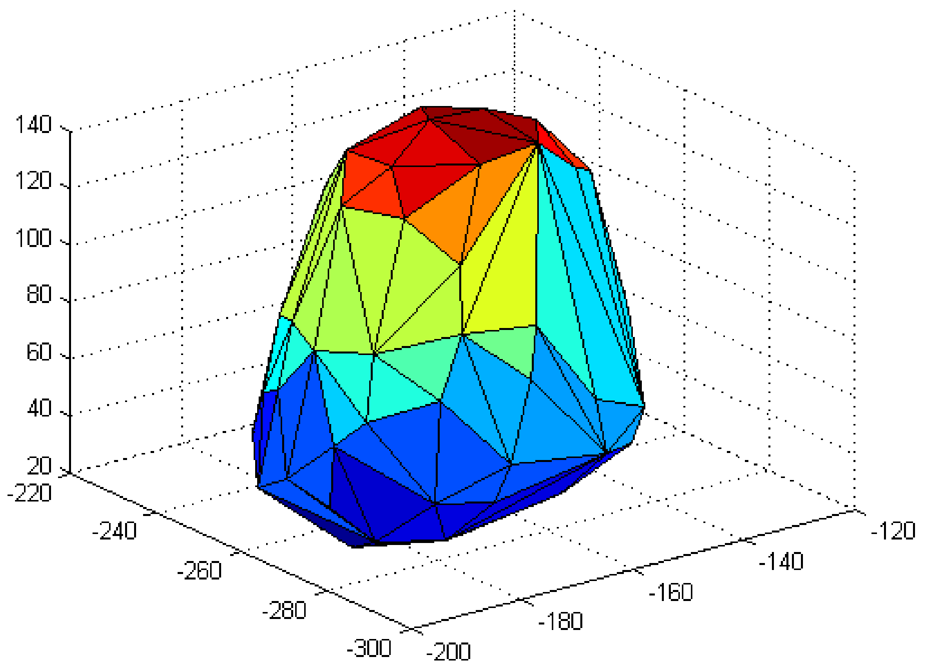

Figure 3.42 Deformed Geometry for Problem 4 (Deformations Calculated by using the Software from [Yijun Liu, n.d.], and Deformations Scaled by a Factor of 4)

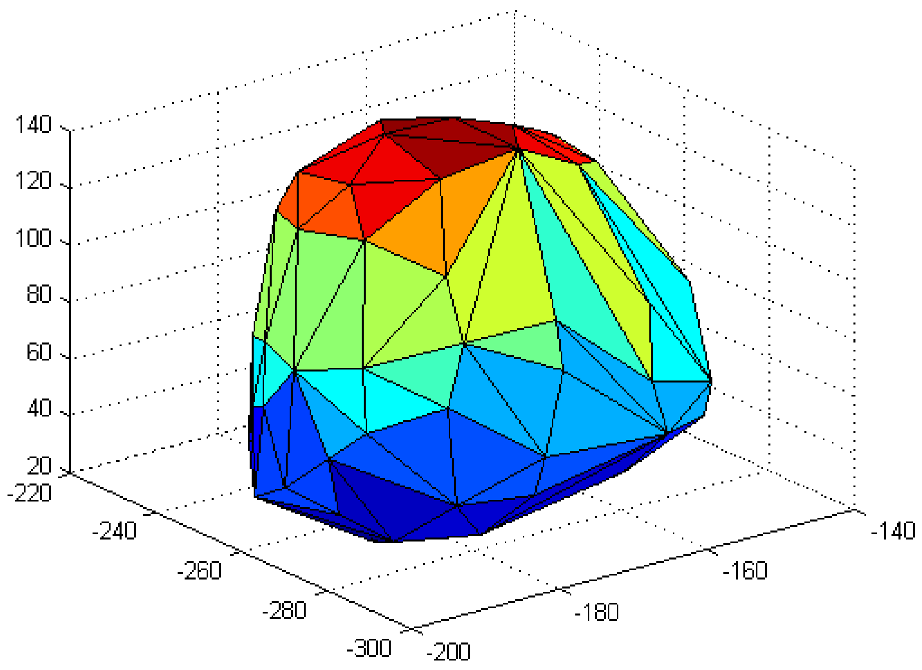

Figure 3.43 Undeformed Geometry for Problem 5

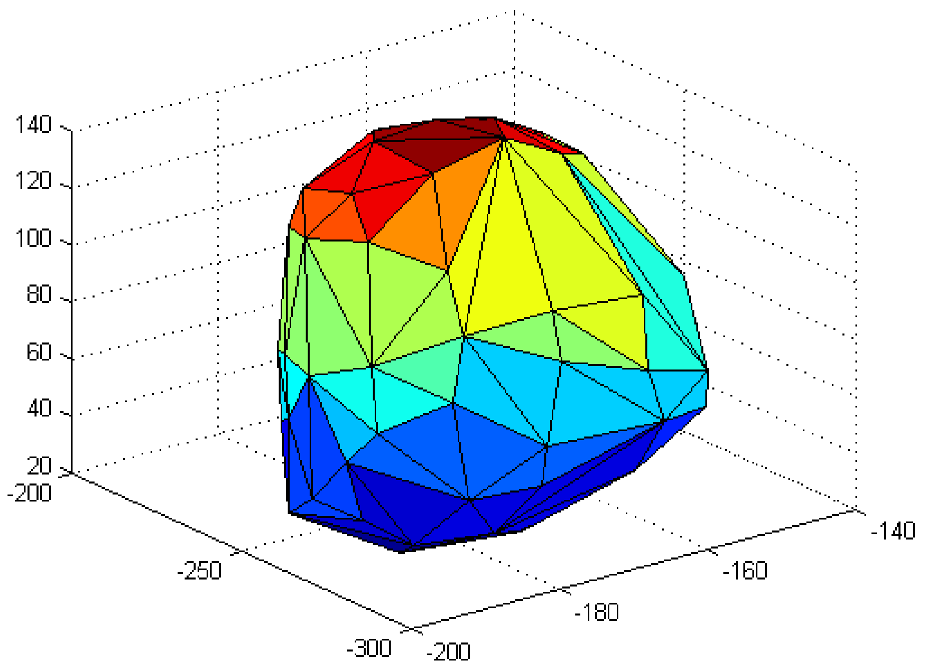

Figure 3.44 Deformed Geometry for Problem 5 (Deformations Calculated by using the Present Author's Code, and Deformations Scaled by a Factor of 4)

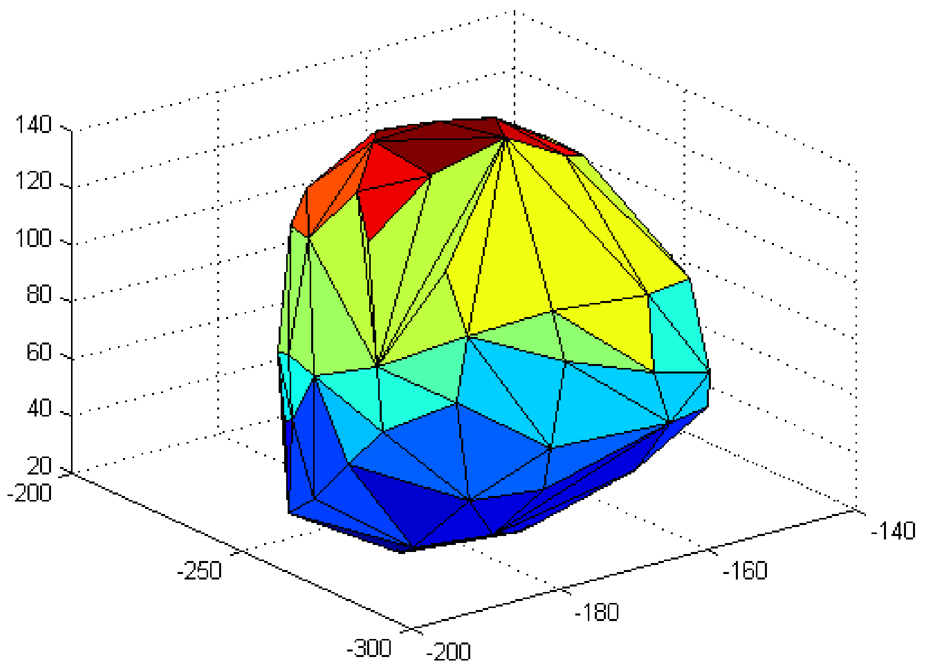

Figure 3.45 Deformed Geometry for Problem 5 (Deformations Calculated by using the Software from [Yijun Liu, n.d.], and Deformations Scaled by a Factor of 4)

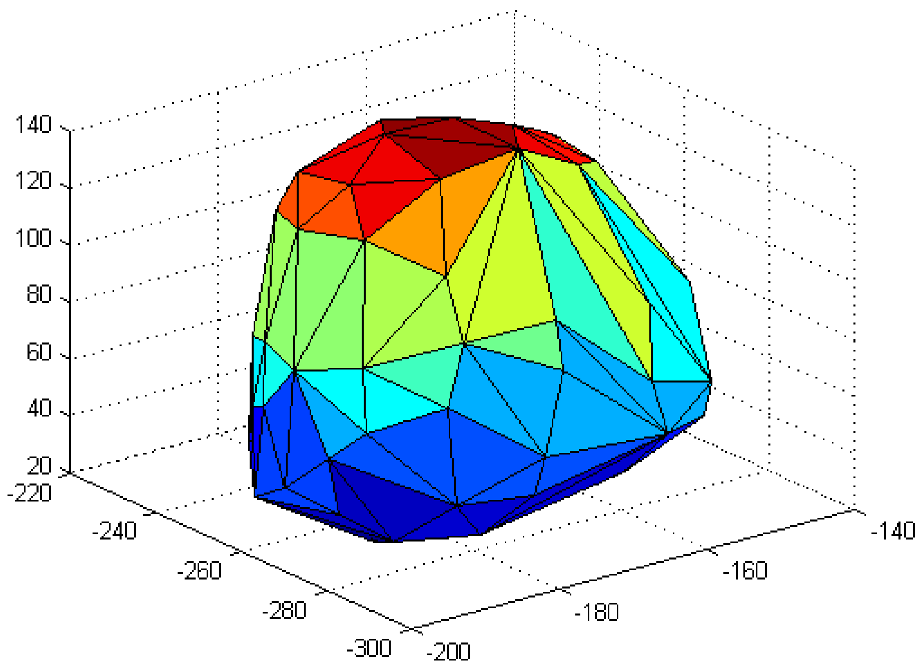

Figure 3.46 Undeformed Geometry for Problem 6

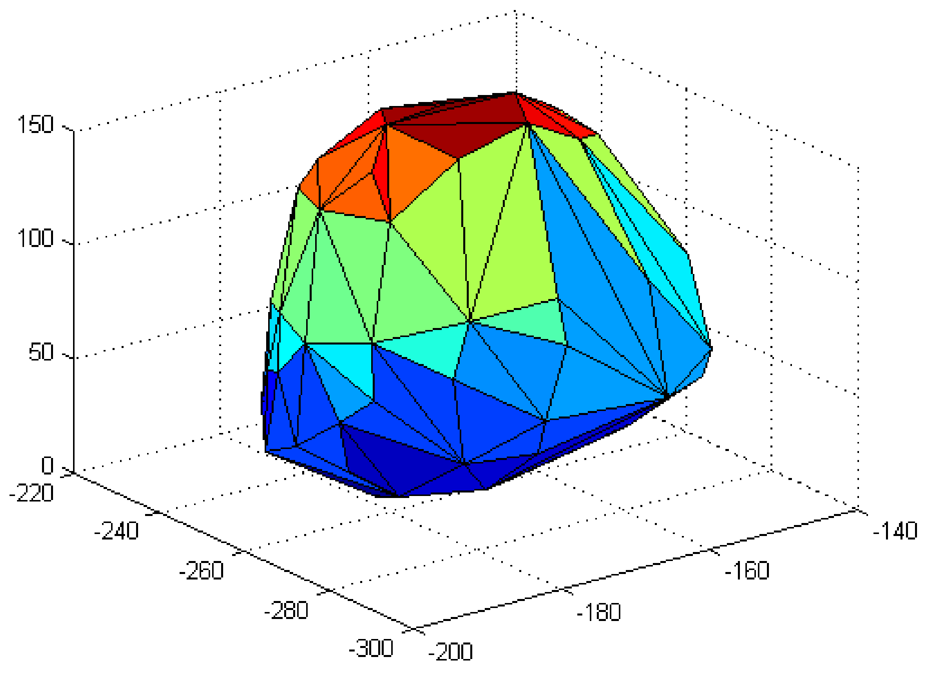

Figure 3.47 Deformed Geometry for Problem 6 (Deformations Calculated by using the Present Author's Code, and Deformations Scaled by a Factor of 4)

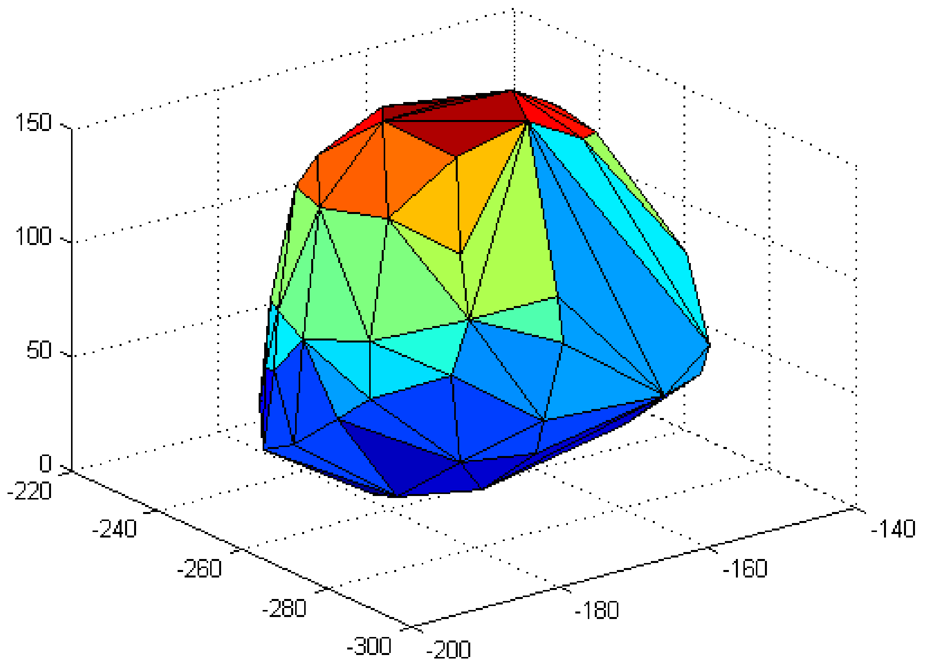

Figure 3.48 Deformed Geometry for Problem 6 (Deformations Calculated by using the Software from [Yijun Liu, n.d.], and Deformations Scaled by a Factor of 4)

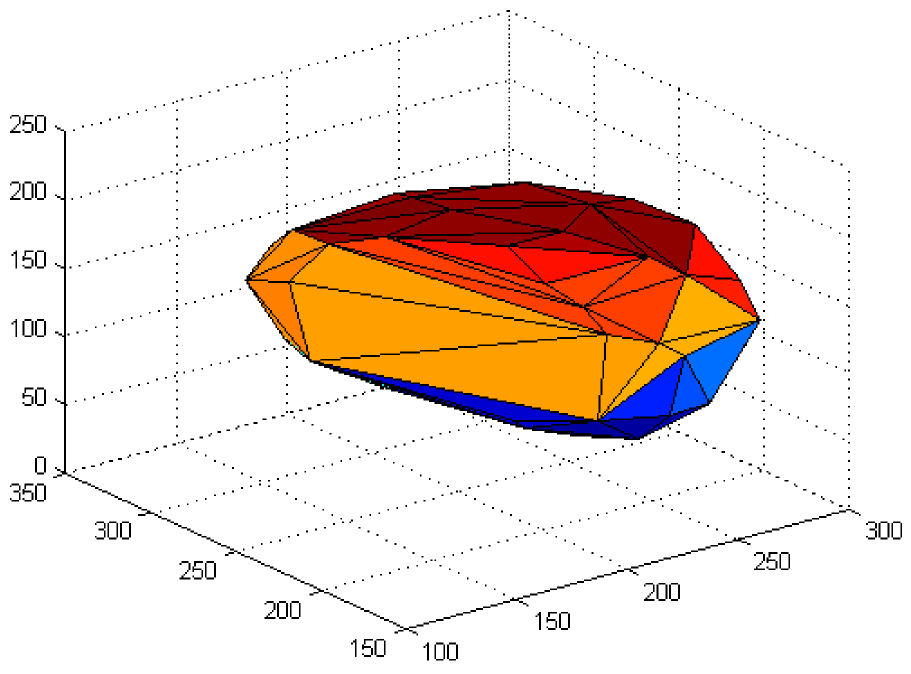

Figure 3.49 Undeformed Geometry for Problem 7

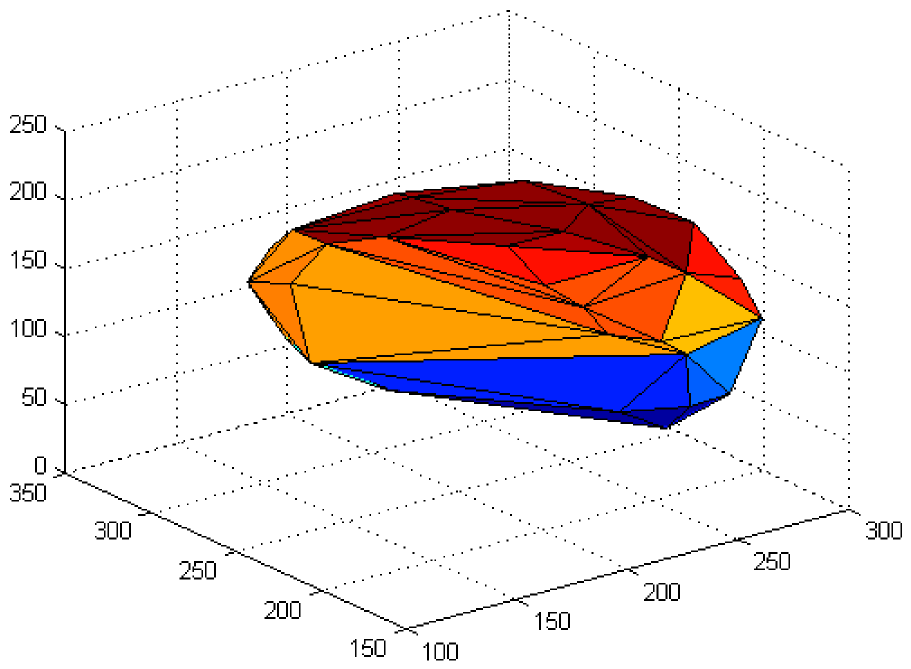

Figure 3.50 Deformed Geometry for Problem 7 (Deformations Calculated by using the Present Author's Code, and Deformations Scaled by a Factor of 4)

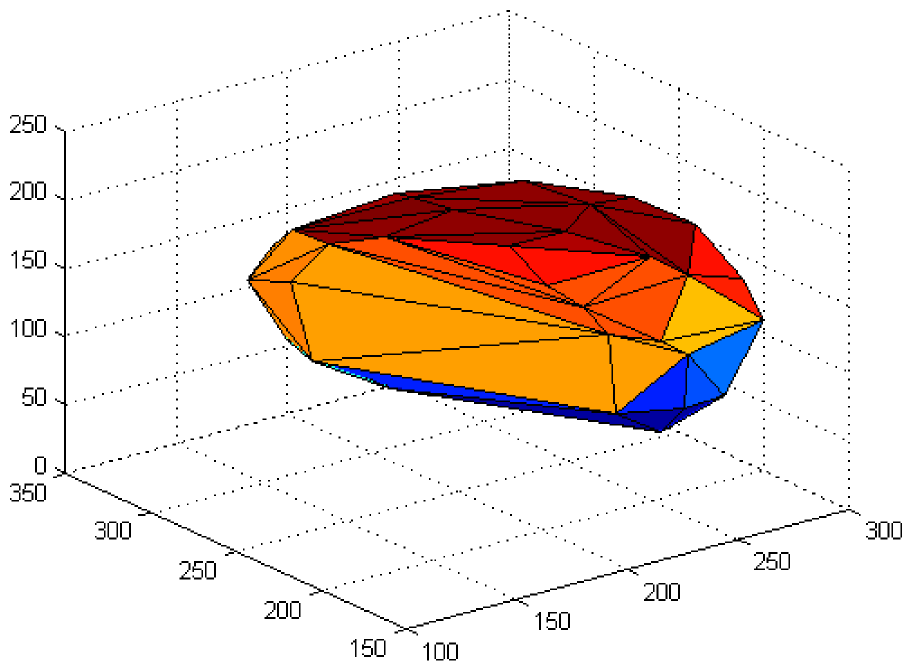

Figure 3.51 Deformed Geometry for Problem 7 (Deformations Calculated by using the Software from [Yijun Liu, n.d.], and Deformations Scaled by a Factor of 4)

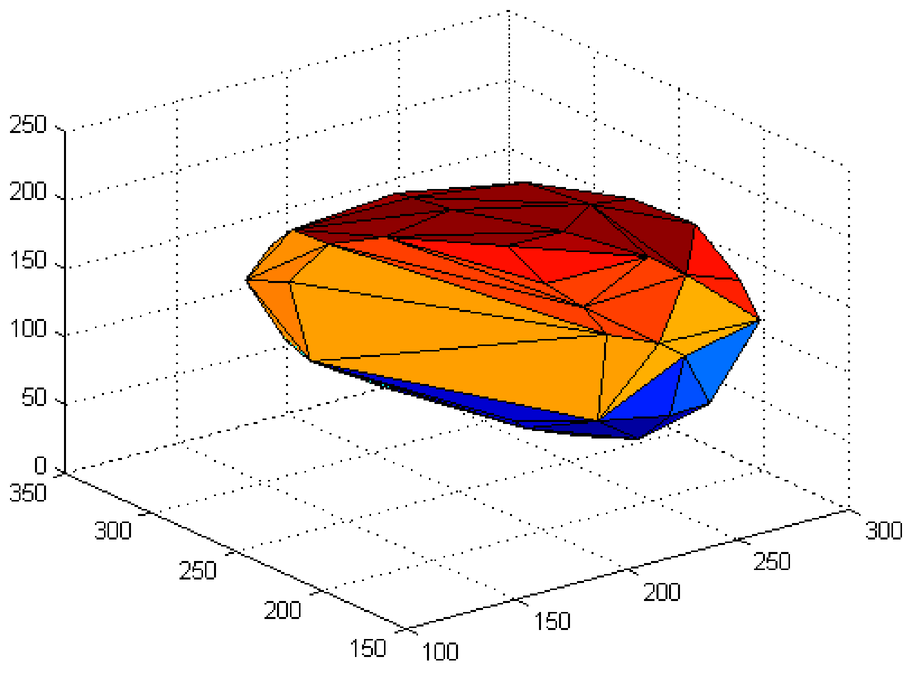

Figure 3.52 Undeformed Geometry for Problem 8

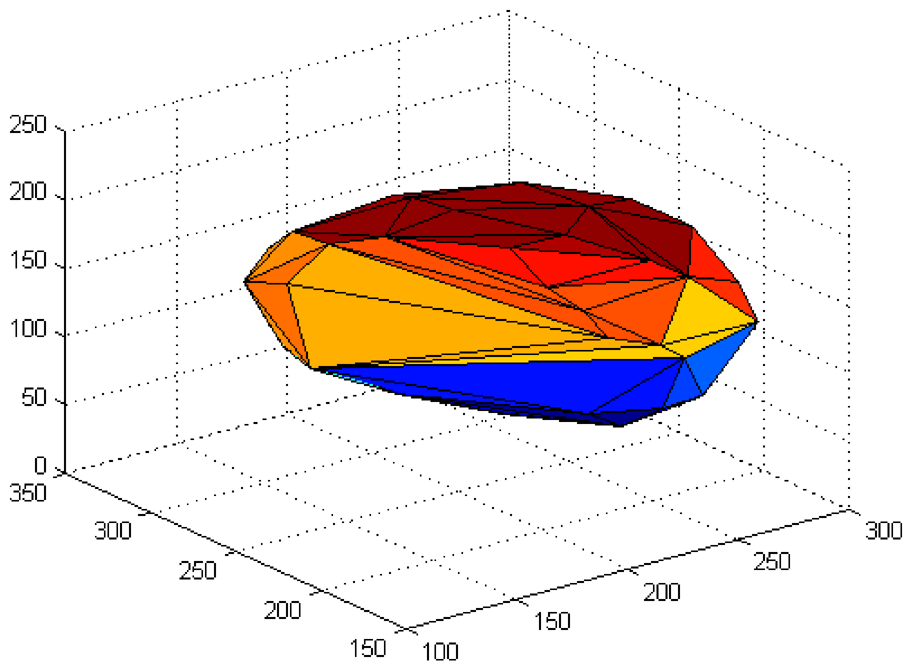

Figure 3.53 Deformed Geometry for Problem 8 (Deformations Calculated by using the Present Author's Code, and Deformations Scaled by a Factor of 4)

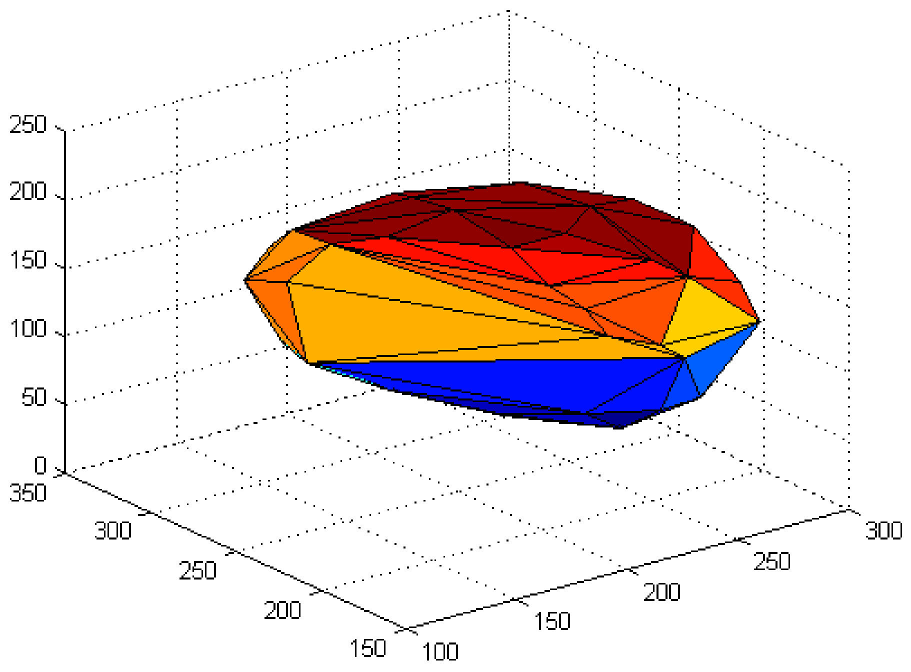

Figure 3.54 Deformed Geometry for Problem 8 (Deformations Calculated by using the Software from [Yijun Liu, n.d.], and Deformations Scaled by a Factor of 4)

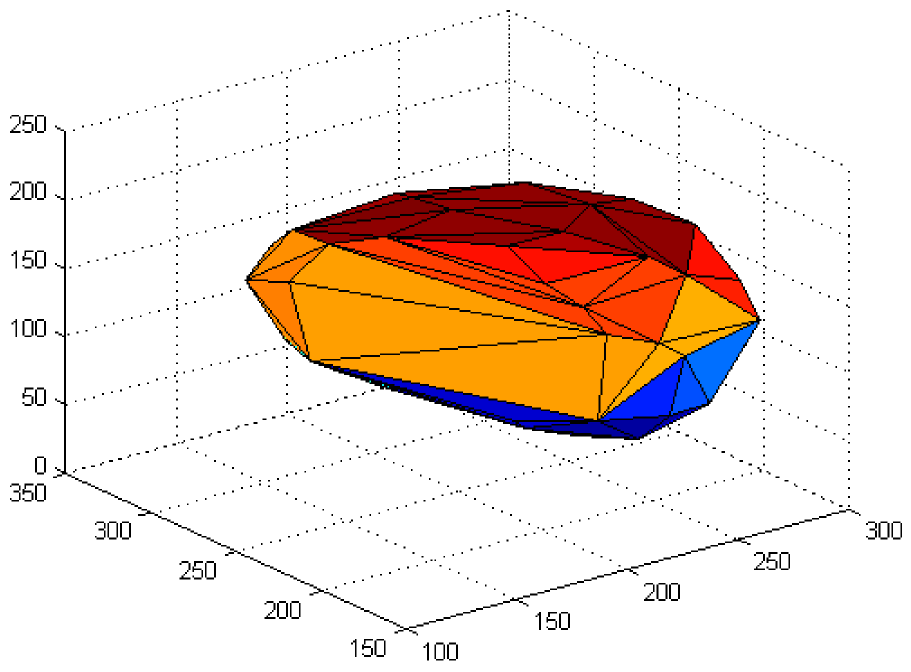

Figure 3.55 Undeformed Geometry for Problem 9

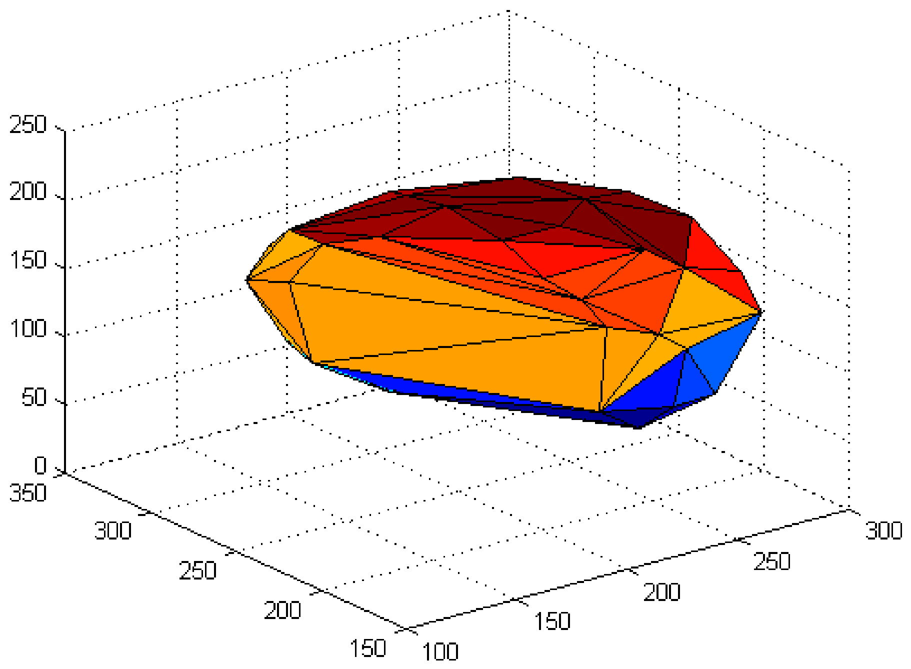

Figure 3.56 Deformed Geometry for Problem 9 (Deformations Calculated by using the Present Author's Code, and Deformations Scaled by a Factor of 4)

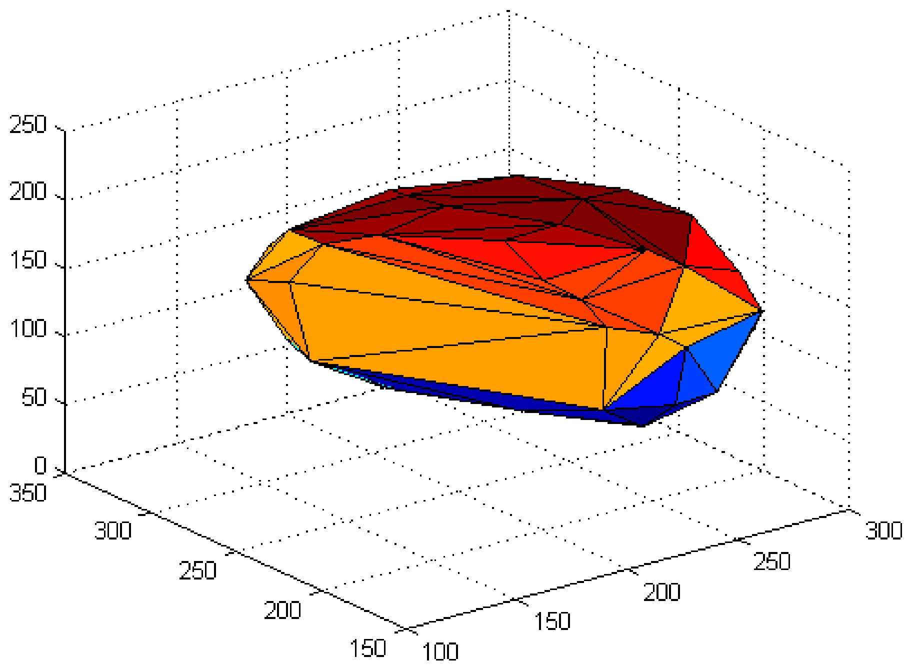

Figure 3.57 Deformed Geometry for Problem 9 (Deformations Calculated by using the Software from [Yijun Liu, n.d.], and Deformations Scaled by a Factor of 4)

3.6 Summary

This chapter first talks about BEM codes developed by this author during the course of the present research. The codes can solve any three dimensional linear elastostatic problem using constant boundary elements while ignoring body forces. The codes comprise a MATLAB code, and a Fortran translation of the MATLAB code. The Fortran translation comes in two varieties: first one can run on a single core of a computer cluster (i.e., sequential version), and the second variety can utilize multiple processors available in computer clusters (i.e., parallelized version). These codes are essential for carrying out the BEM simulations next.

First, a try is given to parallelize the entire MATLAB code with the intention of running it on a desktop computer equipped with a GPU. The goal is to compute the 'characteristic matrix' and the 'right hand side', and also to solve the system of equations, all in real-time. Author has not been successful in achieving this goal using MATLAB and the Parallel Computing Toolbox.

Next, thought was given to obtain the real-time performance, again by utilizing just an ordinary desktop computer and a GPU, by resorting to precomputations. In this case, one has to precompute the 'characteristic matrix' and its inverse offline. Again, MATLAB and Parallel Computing Toolbox are used for this purpose. Results show that it is possible to get real-time performance (real-time graphics, not real-time haptics) when the size of the 'characteristic matrix' is up to about *16000 by 16000*; one can also see that one can get a speed up of about *14* times when the 'characteristic matrix' is of about *16000 by 16000* size. This means that one can perform real-time simulations using boundary elements if the geometry is described by about *5300* constant boundary elements. Of course, the usual limitations with approaches that use precomputations apply in this case too (like one may not be able to compute the 'characteristic matrix' beforehand if there is change in the geometry during simulations, e.g., during simulation of cutting). One can note that it is not possible to perform the same simulation (with the same precomputations of course) in real-time using just an ordinary desktop if a GPU is also not made use of.

Next, a computer cluster is used to carry out BEM simulations. The fully parallelized version of the code (i.e., the parallel Fortran code) is used for this purpose. In this case, the goal is to compute the 'characteristic matrix' and the 'right hand side', and also to solve the system of equations, all in real-time. Simulations are carried out on *1*, *4*, *16*, *64*, and *256* processors. Simulations are also carried out for different block sizes while Block-Cyclically distributing matrices among processors. From the results, one can conclude that it is possible to achieve the real-time performance with BEM (real-time graphics, not real-time haptics), when biological organs are represented by about *96* boundary elements (assuming linear elastostatic behaviour of course). The fastest simulation could complete *24* computations per second (as against the approximately *30* computations per second desirable for high quality real-time graphics). During the simulation of biological organs, the results obtained by using this author's BEM code are found to be in good agreement with the results obtained by using BEM software developed by someone else. During the simulation of biological organs, using an *8 by 8* integration scheme instead of the *16 by 16* integration scheme employed in the present study can speed up the computations, which in turn can enable one to achieve real-time graphics of very high quality and/or enable one to solve a problem of larger size in real-time; this author has observed that using *8 by 8* integration instead of the *16 by 16* integration does not result in unacceptable degradation of accuracy. In the present chapter, all the computations involving real numbers are carried out in double precision, and carrying out at least a few computations using just the single precision can help to speed up the computations.

From the results presented in this chapter, one can see that it would be difficult to obtain the real-time performance (even real-time graphics) if nonlinear material behaviour is to be incorporated into simulations, if custom methods and communication protocols are not used while parallelizing simulations. Results also imply that it may be difficult to obtain real-time haptic feedback with nonlinear material models, even if custom methods and communication protocols are used while parallelizing simulations. However, results also indicate that BEM could be a very useful numerical technique for the realistic simulation of biological organs – including the highly nonlinear soft biological organs – if one is happy with nearly real-time performance, i.e., one computation taking just a few seconds.

One can see that the results presented in the present chapter are relevant not only for the real-time linear elastostatic simulation of biological organs but any simulation that attempts to simulate linear elastostatic response in real-time.

Present chapter also serves to make some record of the performance (like speed) that can be obtained by present day typical computing hardware (like a desktop computer, a graphics processing unit (GPU), a computer cluster) together with contemporary software (like MATLAB, MATLAB Parallel Computing Toolbox, Fortran, MPI, BLACS, ScaLAPACK).

Although using a supercomputer (e.g., IBM Blue Gene) instead of the computer cluster utilized in the present work may offer slightly different speed/performance, one may not expect the speed to improve too much because the well known list of top supercomputers of the world includes several computer clusters at present (which implies that the performance offered by clusters is comparable to the performance offered by supercomputers).

Chapter 4: Discussion, Conclusions, and Scope for Future Work

4.1 A Discussion on Incorporating Nonlinearity into the Simulations

Biological organs are inherently nonlinear. Hence the simulation of biological organs that take into account nonlinearity would result in more accurate results. However, whether the simulation would be more realistic after incorporating nonlinearity depends not only on the accuracy of the results obtained but also on how fast the simulations are (whether or not the simulations are real-time, since not just accuracy but the real-time performance also increases realism).

Hence one has to incorporate nonlinearity into the BEM based simulation of biological organs if one wants to answer the questions: "Is it possible to achieve more realistic simulations by incorporating nonlinearity into the BEM based simulation of biological organs? Is it possible to achieve the real-time performance with such models? Is the accuracy offered by such models sufficient in case the models can only be made of very few elements to achieve the real-time performance? Or, whether one can achieve better realism by using more number of nonlinear elements even if it could result in simulations that are not strictly real-time (nearly real-time simulations)?"

Clearly, from the results presented in the previous chapters, one can conclude that it is difficult to achieve the real-time performance if nonlinearity is to be incorporated into the simulations. However, it should be possible to achieve the real-time performance if very few nonlinear boundary elements are used. On the contrary, it may be possible to achieve nearly real-time performance (although not strictly real-time performance), even if the total number of boundary elements is kept the same as those in the simulations carried out in the previous chapters. Whether or not such nearly real-time nonlinear simulations are preferred over the simulations which are strictly real-time but do not incorporate any nonlinearity is a question which can only be answered by competent and experienced surgeons, after they use surgery simulators built by using both these technologies (one technology at a time). But the first step towards building a BEM based simulator that can simulate nonlinearity could be to carry out the nonlinear simulation of biological organs using BEM.

It might be important to note that no literature is available on the simulation of biological organs using nonlinear BEM, whether in the context of real-time simulations or otherwise.

Even when solving nonlinear problems, the BEM usually uses the same fundamental solutions as those used for linear simulations. This might lead to lesser accuracy. Moreover, the BEM formulations that are generally employed to solve nonlinear problems usually require meshing of the interior of the problem domain, not just the boundary of the problem domain. This can make the BEM less attractive (over techniques like FEM) because one of the reasons for choosing the BEM over techniques like FEM is that it requires meshing of only the boundary of the problem domain (at least for linear problems). It may be noted that once the BEM loses this advantage (when solving nonlinear problems), there may not be any advantage in using the BEM over FEM (in terms of speed and accuracy). Of course, still there is a need to carry out the simulation of biological organs by using the BEM and FEM both, and find out which of the numerical techniques is better suited for the simulation of biological organs. However, while codes and software packages for nonlinear FEM are readily available, codes and software packages for nonlinear BEM are not readily available. This author is not aware of any nonlinear BEM code that may be useful for the simulation of biological organs (e.g., 3D hyperelasticity). Even the commercial boundary element simulation software BEASY (developed by Prof. Brebbia who is widely considered to be the one who invented the BEM) does not include hyperelasticity. Developing one's own nonlinear BEM codes would require significant amount of time, resources, and expertise. These may be the reasons why no one has used nonlinear BEM for the real-time simulation of biological organs.

4.1.1 Literature on Nonlinear BEM

As regards to the use of the BEM for solving nonlinear problems (2D and 3D), one cannot find as much literature as one would expect to see. In fact, one can find only a few references on this topic. An attempt has been made in the following paragraphs to summarize important references on nonlinear BEM, especially those that are important from the point of view of the simulation of biological organs.

The reference [Wei-Zhe Feng, et al., 2015] presents a new BEM for solving 2D and 3D elastoplastic problems without initial stresses/strains. The reference [Trevor G. Davies and Xiao-Wei Gao, 2006] uses the boundary element method to carry out three-dimensional elasto-plastic analysis. The reference [Katia Bertoldi, et al., 2005] presents a new boundary element technique for elastoplastic solids. The technique does not use domain integrals. The reference [Xiao-Wei Gao and Trevor G. Davies, 2000] presents an effective boundary element algorithm for 2D and 3D elastoplastic problems.

The reference [P.M. Baiz and M.H. Aliabadi, 2007] analyzes the buckling of shear deformable shallow shells using the boundary element method, while [M.H. Aliabadi, 2006] uses the boundary element method to analyze shear deformable plates with combined geometric and material nonlinearities. The reference [T. Dirgantara and M.H. Aliabadi, 2006] uses a boundary element formulation to perform geometrically nonlinear analysis of shear deformable shells. The reference [P.H. Wen, et al., 2005] carries out large deflection analysis of Reissner plate using the boundary element method. The reference [Hui-Shen Shen, 2000] discusses the nonlinear bending of simply supported rectangular Reissner–Mindlin plates resting on elastic foundations under transverse and in-plane loads.

The reference [M. Brun, et al., 2003] discusses a boundary element technique for incremental, nonlinear elasticity.

Many a times biological organs may be assumed to be hyperelastic. As far as accuracy of the simulations is concerned (if speed is of no concern), one is expected to get more realistic results by assuming that biological organs are hyperelastic instead of assuming that they follow the linear elastostatic behaviour. Hence, sources from the literature that use the BEM for modelling hyperelasticity are identified in the next paragraph.

The reference [O. Köhler and G. Kuhn, 2001] discusses the application of the Domain-Boundary Element Method (DBEM) for solving hyperelastic and elastoplastic finite deformation problems (axisymmetric and 2D/3D problems). The reference [G. Karami and D. Derakhshan, 2001] introduces a field boundary element method for large deformation analysis of hyperelastic problems.

4.1.2 Nonlinear Formulations and Coding

The references [O. Köhler and G. Kuhn, 2001; G. Karami and D. Derakhshan, 2001] quite extensively and clearly describe the BEM formulation for hyperelasticity. These formulations may readily be used for the nonlinear BEM-based simulation of biological organs.

One of the differences between a nonlinear BEM code and a linear BEM code is that while carrying out nonlinear simulations, the characteristic matrix has to be calculated using a nonlinear formulation, e.g., the formulation explained in [O. Köhler and G. Kuhn, 2001; G. Karami and D. Derakhshan, 2001]. The other difference is that the system of algebraic equations to be solved to get the final solution is not linear. Hence a solution method that is capable of solving a system of nonlinear algebraic equations has to be employed at the last stage.

Codes may be parallelized to make them run faster. Hardware that may be utilized include: computer cluster, supercomputer, GPU, using Intel Many Integrated Core Architecture (Intel MIC, which is a coprocessor), using a processor with many cores (e.g., Knights Landing, which is a standalone processor). From the reference [Victor W. Lee, et. al, 2010], it seems that GPUs may or may not be as good as they appear to be. However, a single processor with many cores is likely to be helpful for real-time simulations since the time for data transfer between computing cores for this type of processors is very small because all the cores are present in the same chip (same piece of semiconductor). One may also note that a few researchers are trying to develop processors that would have a few hundred cores each.

4.1.3 Comparison between the Results from Nonlinear Analysis (using ANSYS) and Linear Analysis (using ANSYS)

In this section, comparison is made between the results obtained by using linear elastostatic analysis and nonlinear analysis. The commercial software package ANSYS is used for the purpose; ANSYS is used for both linear and nonlinear analysis here.

The comparison between linear and nonlinear analysis is done for the three simulations involving left kidney of the Visible Human male. The simulations (meanings geometry and boundary conditions) are explained in the last chapter and are named in the last chapter as Problem 1, Problem 2 and Problem 3. The material properties used in this chapter are the same as those used in the last chapter for BEM simulations (Young's modulus = 150 N/mm^2, Poisson's ratio = 0.4), and only geometric nonlinearity is taken into account (the material is considered to be linear elastic, but the material can undergo large deformation). Literature says that it is much more important to incorporate geometric nonlinearity when compared to incorporating nonlinear material models, and many a times just incorporating geometric nonlinearity while using just the linear elastic constitutive model results in highly accurate simulations [Adam Wittek, et al., 2009].

The element type used is Tet 10node 187. The geometry is discretized into 782 nodes in total. The discretized geometry (undeformed geometry), as displayed in ANSYS, is shown in Figure 4.1. Figure 4.2 shows the undeformed and deformed geometry, displayed over the same frame, for Problem 1 and for the linear analysis (rendered mesh refers to the undeformed geometry whereas the wireframe mesh refers to the deformed geometry). Similarly, Figure 4.3 shows the undeformed and deformed geometry, displayed over the same frame, for Problem 1 and for the nonlinear analysis (rendered mesh refers to the undeformed geometry whereas the wireframe mesh refers to the deformed geometry).

Similarly, Figure 4.4 shows the undeformed and deformed geometry, for Problem 2 and for the linear analysis (For Figure 4.4 to Figure 4.7, rendered mesh refers to the undeformed geometry whereas the wireframe mesh refers to the deformed geometry). Figure 4.5 shows the undeformed and deformed geometry, for Problem 2 and for the nonlinear analysis. Figure 4.6 shows the undeformed and deformed geometry, for Problem 3 and for the linear analysis. Figure 4.7 shows the undeformed and deformed geometry, for Problem 3 and for the nonlinear analysis.

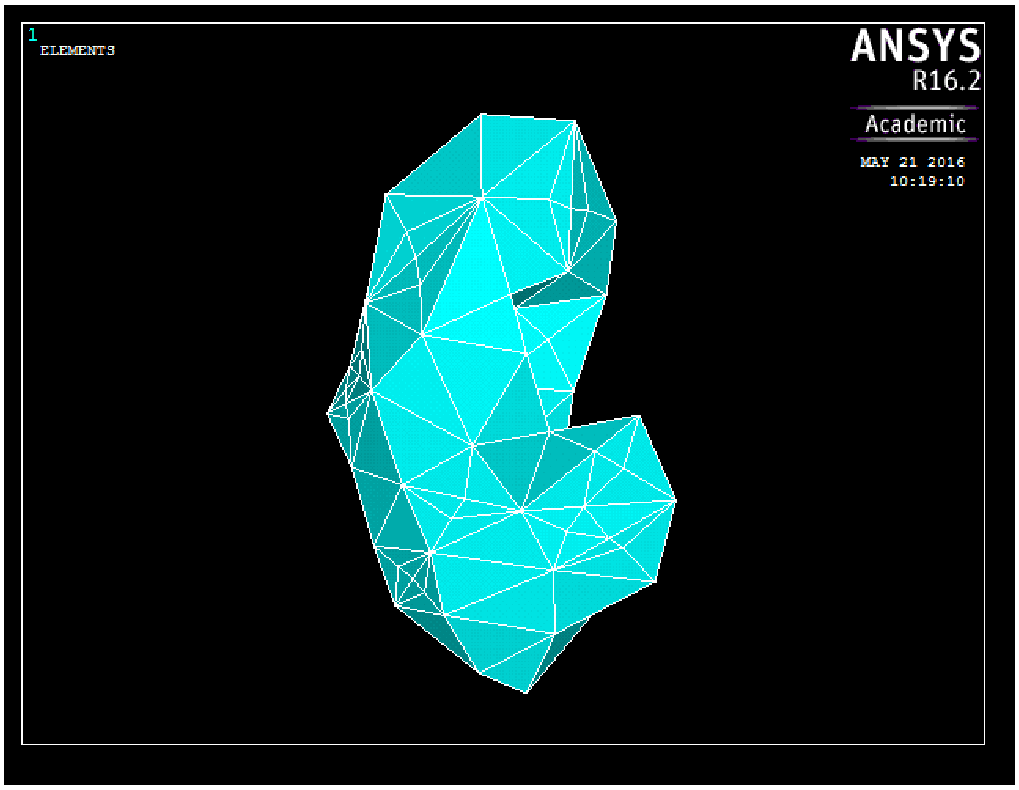

Figure 4.1 Discretized Geometry as Displayed in ANSYS

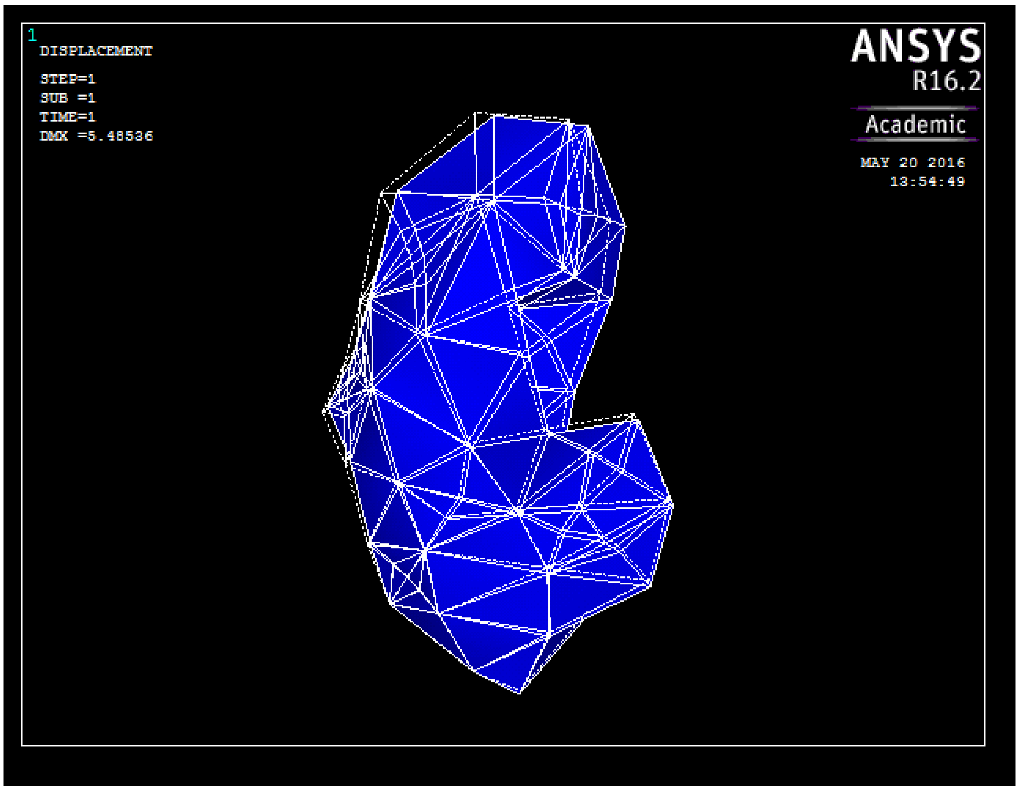

Figure 4.2 Undeformed and Deformed Geometry for Problem 1 (Linear Analysis)

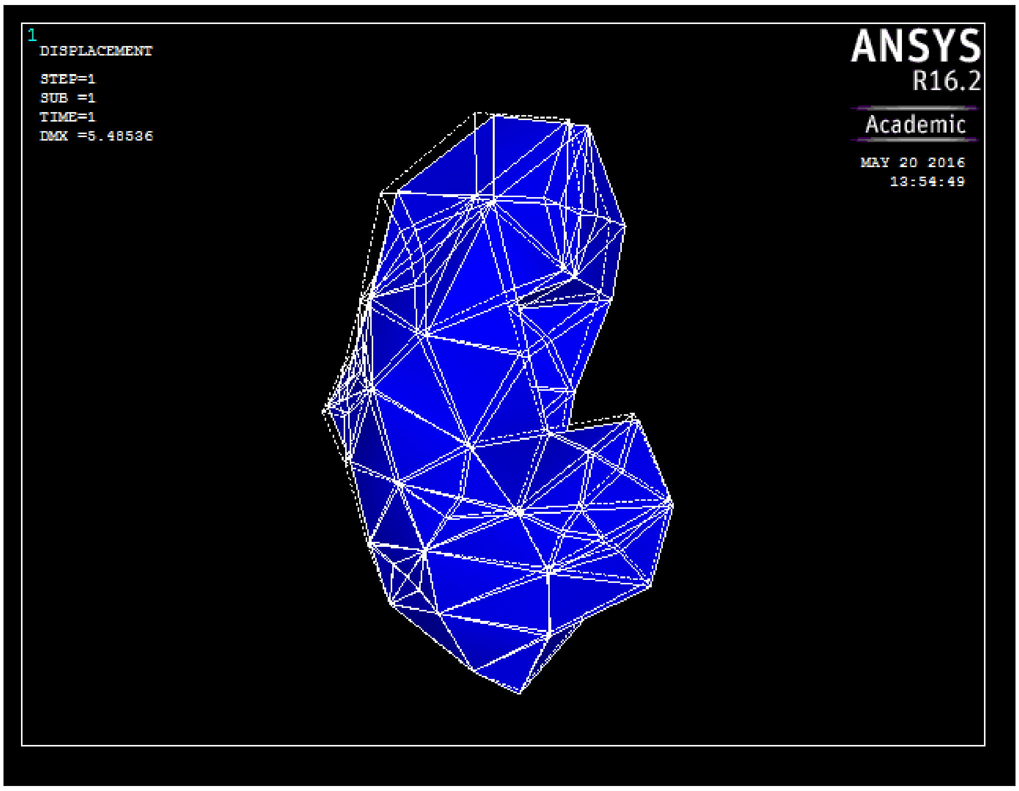

Figure 4.3 Undeformed and Deformed Geometry for Problem 1 (Nonlinear Analysis)

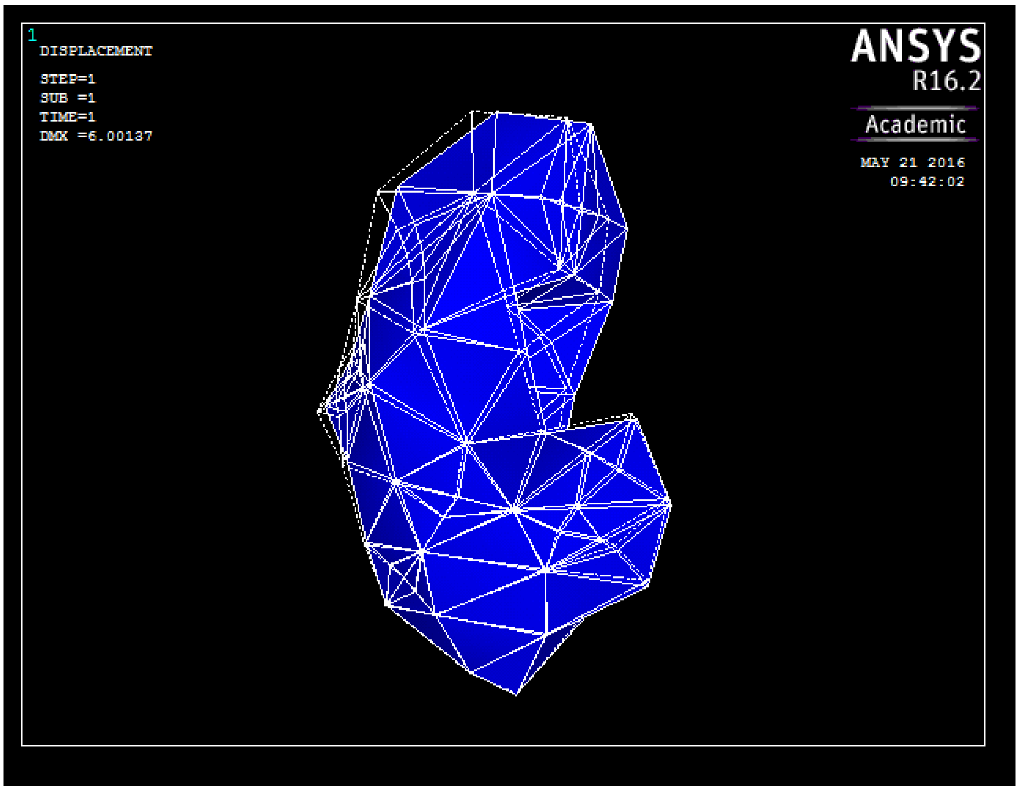

Figure 4.4 Undeformed and Deformed Geometry for Problem 2 (Linear Analysis)

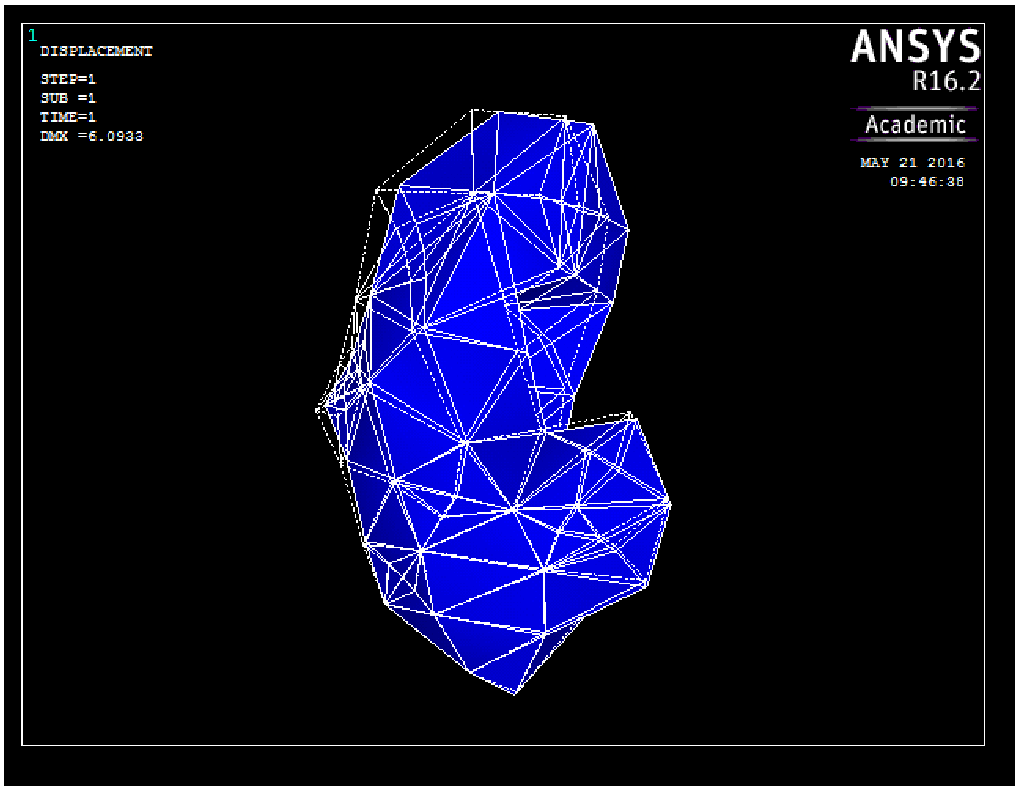

Figure 4.5 Undeformed and Deformed Geometry for Problem 2 (Nonlinear Analysis)

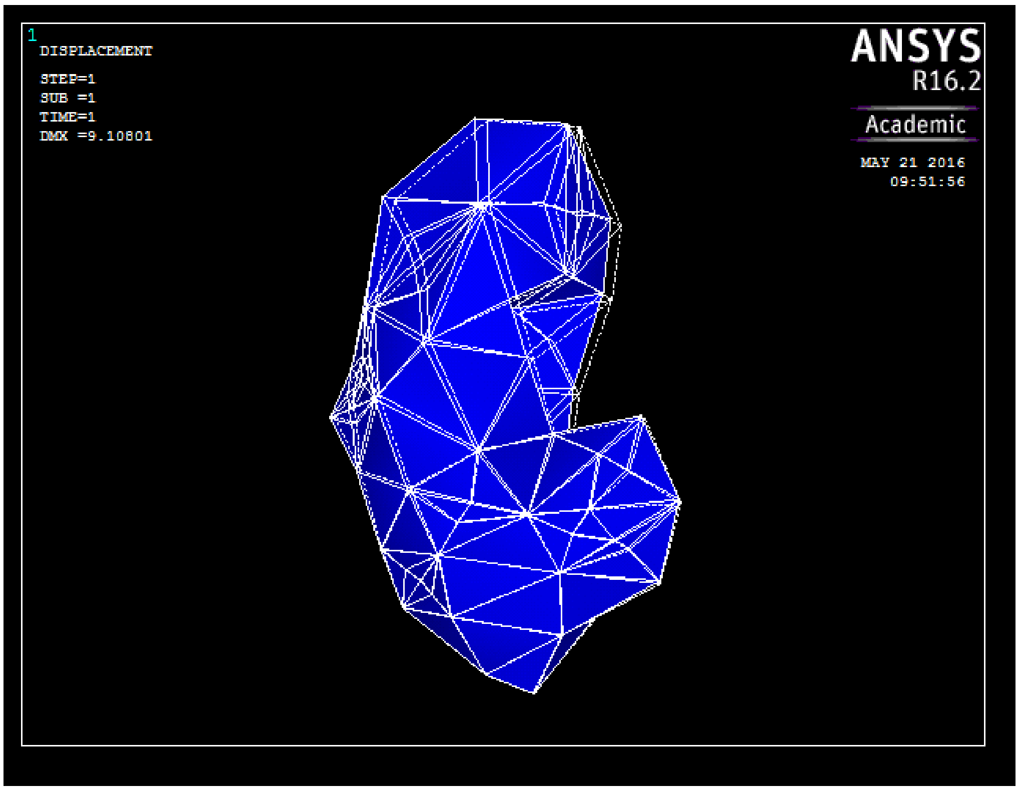

Figure 4.6 Undeformed and Deformed Geometry for Problem 3 (Linear Analysis)

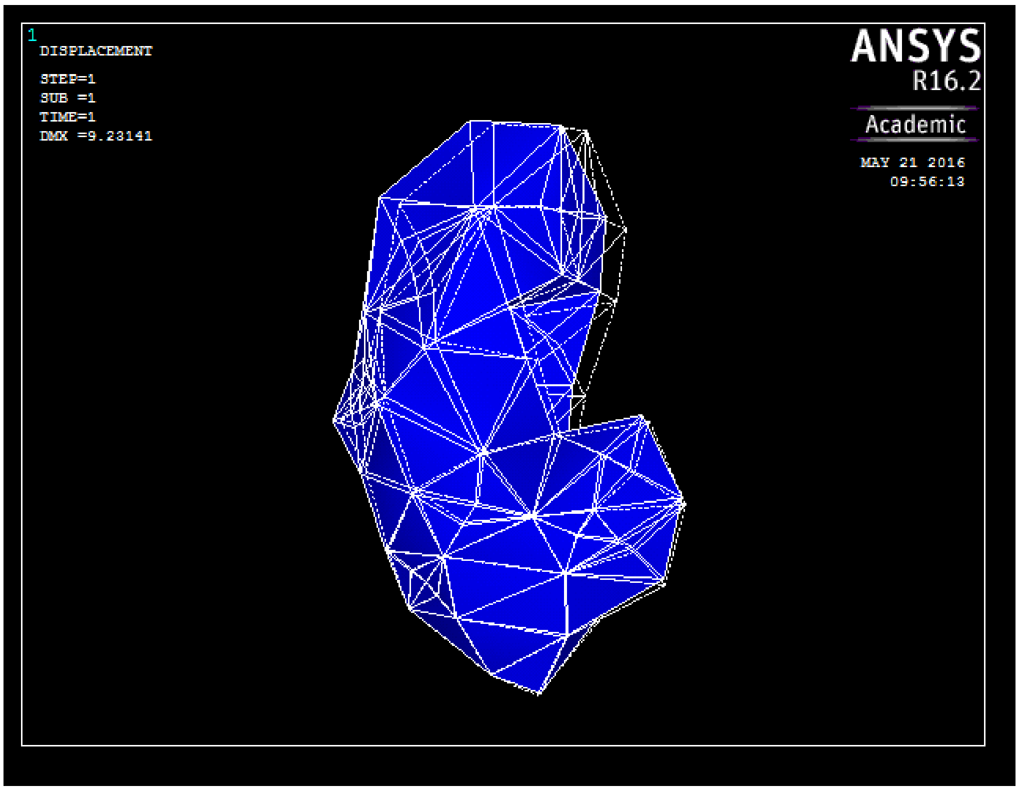

Figure 4.7 Undeformed and Deformed Geometry for Problem 3 (Nonlinear Analysis)

It is easier to compare the difference between the results from linear and nonlinear analyses if the actual values of the displacements are tabulated. Hence, for each of the analyses above, the values of the displacement vector sum at eleven distinct nodes is noted down. These values are tabulated in Table 4.1 to Table 4.3. The eleven nodes are selected such that they are not from a certain part of the geometry only (i.e., nodes are scattered throughout the geometry). The node numbers of the chosen nodes are: 41, 43, 50, 49, 34, 15, 11, 4, 18, 246, 20.

Table 4.1 Percentage Errors for Problem 1 (Linear FEM)

Node Number	Displacement Vector Sum for the Large Deformation Analysis *(mm)*	Displacement Vector Sum for the Small Deformation Analysis *(mm)*	Percentage Error
41	4.176	5.011	19.995
43	5.485	5.983	9.079
50	5.161	5.854	13.428
49	5.177	5.210	0.637
34	2.200	2.300	4.545
15	0.000	0.000	0.000
11	0.728	0.659	-9.478
4	0.750	0.982	30.933
18	1.384	1.375	-0.650
246	1.551	1.583	2.063
20	1.870	1.763	-5.722

Table 4.2 Percentage Errors for Problem 2 (Linear FEM)

Node Number	**Displacement Vector Sum for the Large Deformation Analysis *(mm)***	**Displacement Vector Sum for the Small Deformation Analysis *(mm)***	**Percentage Error**
41	5.043	5.011	-0.635
43	6.048	5.983	-1.075
50	5.932	5.854	-1.315
49	5.251	5.210	-0.781
34	2.323	2.300	-0.990
15	0.000	0.000	0.000
11	0.661	0.659	-0.303
4	0.984	0.982	-0.203
18	1.373	1.375	0.146
246	1.584	1.583	-0.063
20	1.767	1.763	-0.226

Table 4.3 Percentage Errors for Problem 3 (Linear FEM)

Node Number	**Displacement Vector Sum for the Large Deformation Analysis *(mm)***	**Displacement Vector Sum for the Small Deformation Analysis *(mm)***	**Percentage Error**
41	7.551	7.260	-3.854
43	8.922	8.726	-2.197
50	9.231	9.108	-1.332
49	8.446	8.467	0.249
34	3.146	3.090	-1.780
15	0.000	0.000	0.000
11	0.948	0.910	-4.008
4	1.488	1.371	-7.863
18	1.610	1.459	-9.379
246	1.776	1.556	-12.387
20	2.051	1.822	-11.165

4.1.4 Comparison between the Results from Nonlinear Analysis (using ANSYS) and Linear Analysis (using this Author's BEM Code)

Instead of comparing the results from nonlinear analysis using ANSYS with the results from linear analysis using ANSYS (as done in the last section), it is better if the linear analysis is conducted using this author's BEM code (instead of ANSYS). However, here one needs to compare the results at the same location (node) of the problem domain, and it is difficult to do this since FEM and BEM use different discretizations and hence it is difficult to have nodes at the same locations for both the discretizations. Still a comparison is done in this section by manually (visually) locating the nodes in the BEM discretization, which may be located approximately at the same location as the corresponding nodes in the finite element model (ANSYS). Of course, it may be noted that the nodes in the BEM discretization are not located exactly at the same location as the corresponding nodes in the FEM discretization, and this itself can be a cause of some amount of error.

The same Problem 1, Problem 2, and Problem 3 chosen for the simulations for the last section are chosen for the simulations for this section also. Of course, for each of Problem 1, Problem 2, and Problem 3 here, geometry, loads and boundary conditions, and material properties (both for the linear analysis and the nonlinear analysis) used here are the same as the ones mentioned in the last section.

The results from linear and nonlinear analyses are compared by tabulating the actual values of the displacement vector sum at eleven distinct points (tables similar to those in the last section). The values are tabulated in Table 4.4 to Table 4.6. The eleven nodes in the FEM model are the same as those chosen in the last section, i.e., 41, 43, 50, 49, 34, 15, 11, 4, 18, 246, 20. The eleven nodes in the BEM model are chosen such that they are located approximately at the same location as the corresponding FEM nodes, and the node numbers of the corresponding BEM nodes are: 84, 87, 96, 83, 68, 24, 13, 11, 22, 25, 50.

Table 4.4 Percentage Errors for Problem 1 (Linear BEM)

Node Number	**Displacement Vector Sum for the Large Deformation Analysis *(mm)***	**Displacement Vector Sum for the Small Deformation Analysis *(mm)***	**Percentage Error**
84	4.176	3.911	-6.346
87	5.485	4.513	-17.721
96	5.161	4.932	-4.437
83	5.177	4.133	-20.166
68	2.200	1.904	-13.455
24	0.000	0.000	0.000
13	0.728	0.852	17.033
11	0.750	0.933	24.400
22	1.384	1.549	11.922
25	1.551	1.625	4.771
50	1.870	1.723	-7.861

Table 4.5 Percentage Errors for Problem 2 (Linear BEM)

Node Number	**Displacement Vector Sum for the Large Deformation Analysis *(mm)***	**Displacement Vector Sum for the Small Deformation Analysis *(mm)***	**Percentage Error**
84	5.043	3.025	-40.016
87	6.048	3.826	-36.739
96	5.932	4.322	-27.141
83	5.251	4.115	-21.634
68	2.323	2.404	3.487
24	0.000	0.000	0.000
13	0.661	0.150	-77.307
11	0.984	0.162	-83.537
22	1.373	0.389	-71.668
25	1.584	0.546	-65.530
50	1.767	0.839	-52.518

Table 4.6 Percentage Errors for Problem 3 (Linear BEM)

Node Number	**Displacement Vector Sum for the Large Deformation Analysis *(mm)***	**Displacement Vector Sum for the Small Deformation Analysis *(mm)***	**Percentage Error**
84	7.551	3.707	-50.907
87	8.922	3.775	-57.689
96	9.231	3.838	-58.423
83	8.446	3.389	-59.874
68	3.146	1.717	-45.423
24	0.000	0.000	0.000
13	0.948	0.747	-21.203
11	1.488	0.792	-46.774
22	1.610	0.999	-37.950
25	1.776	1.096	-38.288
50	2.051	1.183	-42.321

4.1.5 A Note on the Difference between the Results from Linear and Nonlinear Analysis

From the tables and the figures in the last two sections, one may see that there is not too much difference between the results obtained by linear analysis and nonlinear analysis for many of the cases, especially when ANSYS is used both for linear analysis and nonlinear analysis although the percentage errors might be significant when this author's code is used for the linear analysis. However, it is difficult to definitively say how much error is allowed. Only surgeons can say whether a simulation is useful or not. In fact, validating a numerical model by taking feedback from many surgeons, many surgical procedures, and many trials could itself be a research topic. As of present, there is no clarity on what is the allowable error in a simulation, and the subject is a research topic which has not been explored well.

This author's stand is that it is good to stick to linear elastostatic behaviour at present, and as more powerful hardware (together with relevant software) becomes readily available in the future, it may be good to incorporate nonlinearity. Of course, the relevant technology (e.g., developing nonlinear BEM codes) may be developed right now, and possibly the results could be used with benefit in cases where there is no need for the simulations to be strictly real-time (e.g., surgery planning).

4.2 Conclusions

In Chapter 1, literature review is presented, research problem is defined, and contributions from this thesis are listed.

In Chapter 2, a novel procedure that requires only a few free software packages to obtain the geometry of biological organs from 2D image sequences is presented. The procedure makes use of free software packages only. The geometry of a pig liver is extracted from CT scan images for illustration purpose. Next, the three dimensional geometry of human kidney (left and right kidney of male, and left and right kidney of female) is obtained from the Visible Human Dataset (VHD). The novel procedure presented in this work can be used to obtain patient specific organ geometry from patient specific images, without requiring any of the many commercial software packages that can readily do the job. The software packages used here to reconstruct biological organs are quite established and it is reasonable to assume that they do not produce erroneous results. Although it is possible to perform the same reconstruction (as the one carried out in Chapter 2) using some well established commercial software and compare the biological organs thus reconstructed with the one obtained here to establish the validity of the procedure followed here, this step may not be required since the present work just uses a few software packages all of which must have undergone testing prior to their release.

In Chapter 3, a BEM code that can solve 3D linear elastostatic problems without accounting for body forces, developed from scratch by this author, is explained. This code is used to carry out studies on the viability of BEM for the real-time simulation of biological organs, as explained in the later part of the chapter. The code explained is the first BEM source code for 3D elasticity that is available for open access. The code assumes further significance because it is released under the very permissive

MIT License, which eliminates the ambiguity on how the code can be used by others. The code can be of use to those who need a BEM source code for 3D linear elasticity, especially because the present author could not find any suitable BEM source code for 3D linear elastostatics either as free and open source software or as paid software. In the later part of the chapter, the code is used to solve problems involving human liver and human kidneys, and BEM is found to give accurate solutions. The later part of the chapter also demonstrates that it is possible to simulate linear elastostatic behaviour in real-time using BEM, without resorting to any type of precomputations. This conclusion is arrived at by using a computer cluster, by fully parallelizing the simulations, and by performing simulations on different number of processors and for different block sizes (by sampling the whole range of possible block sizes, while block-cyclically distributing matrices among processors). One can note that all the earlier works use some or the other type/types of precomputations to achieve the real-time performance with BEM, although some of the earlier works recommend the present approach as their future work. The result has a lot of implications because once it is found that all computations can be performed in real-time, there is no need to separately prove that every type of cutting, suturing etc. can be simulated in real-time (since it is possible to get a complete solution in real-time). This statement holds true because only real-time graphics (not real-time haptics) has been achieved in the present work, and whenever haptics is ignored, cutting/suturing amounts to nothing more than a change in the geometry/topology and a change in the boundary conditions. One can see that there is no need to calculate the modified 'stiffness' matrix from the original 'stiffness' matrix when there is a change in the geometry because of a surgical cut, if the present approach is followed. While using the approach followed in the present work, one is guaranteed to get the real-time performance irrespective of whether or not there is a change in the geometry (because of cutting, say) and irrespective of whether the change in the geometry is small or large, once it is found that the simulation is real-time for the given problem (or the given problem size). It is important to note that this is not the case if one follows any of the approaches that are followed by any of the earlier works that use BEM to achieve the real-time performance during the simulation of biological organs. One can arrive at these conclusions just by qualitative reasoning, and hence quantitative justification (e.g., a demonstration of cutting of biological organs in real-time) is not needed in support of these arguments. Of course, present author cannot claim that the present implementation is the best possible BEM implementation, or cannot claim that the present implementation is the speediest possible BEM implementation. However, since real-time performance has already been obtained using the present implementation, it would only be a beneficial thing if someone else can show that the simulations can run even faster (that would not prove that the simulations are not real-time).

In addition to having their own individual merits, Chapter 2 and Chapter 3 could together represent a step towards building a surgical simulator that uses only free and open source software (operating system, image processing software, BEM based simulation software).

In Chapter 4, thesis is summarized and scope for future work is indicated.

Based on the above, the following conclusions can be drawn:

(i) It is possible to reconstruct the geometry of biological organs by making use of only free and open source software, by making use of the new procedure presented.

(ii) It is possible to simulate biological organs in real-time (only real-time graphics) if the organs are assumed to follow linear elastostatic behavior, by making use of the proposed approach. As opposed to any of the earlier literature, the proposed approach does not use any precomputations. The feature of not using any precomputations in the proposed approach is one of the contributions of this thesis.

(iii) The use of a GPU has speeded up the computations by an order of magnitude.

(iv) Use of a computer cluster was useful for speeding up the computations. Very good scalability was obtained until the number of processors was increased up to 16.

(v) It will be difficult to simulate biological organs in real-time (even real-time graphics) if the organs are assumed to follow non-linear behavior, if the novel approach is employed. One may note that the other approaches that use the BEM to achieve real-time simulations have not been able to incorporate nonlinearity also.

(vi) It is difficult to simulate biological organs in real-time if real-time haptics is required and if the novel approach is employed, even if the organs are assumed to follow linear elastostatic behaviour. One may note that the other approaches that use the BEM to achieve real-time simulations too have not been able to incorporate haptics, even by assuming that the organs follow the linear elastostatic behaviour.

4.3 Scope for Future Work

BEM is a promising numerical technique. However, source codes are not readily available for BEM. Hence the future work could be to develop a BEM library; the library should support highly nonlinear analyses; parallelized versions of the source codes (which can execute on a computer cluster or a supercomputer) should also be developed. Optionally, the source codes could be made freely available to anyone for any purpose. Next, soft biological organs can be simulated on the necessary hardware using BEM; target is to achieve both realistic and (nearly) real-time performance. Finally, a high quality BEM based surgical simulator needs to be built (e.g., a simulator that can simulate laparoscopic surgery). One can include rendering, include interaction with other organs, enable cutting, and include physiology (like flow of blood during cutting etc.). However, one may note that these tasks would require enormous amount of time, resources, and expertise.

Alternatively, as future work, one could try to use soft computing techniques to achieve the real-time performance even with nonlinear models and while using just a desktop computer. Author has done some preliminary work on using Support Vector Machines (SVM) to achieve realistic simulations in real-time [Kirana Kumara P, 2013], and future work could be to continue the work. However, one should note that although it may be possible to simulate nonlinear material behaviour in real-time by utilizing just an ordinary desktop computer if one makes use of SVM, there are serious limitations for this approach. The Support Vector Machines need to be trained before using them for the simulations, and the training requires huge amount of training data; also, training is a cumbersome process. Whenever there is change in the geometry (e.g., because of cutting), the SVM has to be trained again, using the training data generated for the changed geometry. Hence it is not possible to use SVM in those cases to perform real-time simulations. One can note that the Boundary Element Method, although computationally intensive when compared to SVM, can handle that type of problems and hence may be used to obtain realistic simulations for that type of problems also, at least with nearly real-time performance if not with real-time performance.

References

3D-DOCTOR, n.d., FDA 510K Cleared, vector-based 3D imaging, modeling and measurement software, http://www.ablesw.com/3d-doctor/ (accessed July 14, 2012)

A. C. M. Dumay, G. J. Jense, 1995, Endoscopic Surgery Simulation in a Virtual Environment, Comput. Biol. Med. Vol. 25, No. 2. pp. 139-148

Abramoff M.D., Magelhaes P.J., Ram S.J., 2004, Image Processing with ImageJ, Biophotonics International, volume 11, issue 7, pp. 36-42

Adam Wittek, Trent Hawkins, Karol Miller, 2009, On the unimportance of constitutive models in computing brain deformation for image-guided surgery, Biomech Model Mechanobiol. 2009 Feb; 8(1): 77–84, doi: 10.1007/s10237-008-0118-1

Aimee Sergovich, Marjorie Johnson, Timothy D. Wilson, 2010, Explorable Three-Dimensional Digital Model of the Female Pelvis, Pelvic Contents, and Perineum for Anatomical Education, Anatomical Sciences Education, 2010, 3:127–133

Al Aho, Jeff Ullman, 1992, Foundations of Computer Science, Available from: http://infolab.stanford.edu/~ullman/focs.html (accessed July 3, 2014)

Alastair McKinstry, Alin Elena, Ruairi Nestor, n.d., An Introduction to Fortran, Available from: http://www.ichec.ie/support/tutorials/fortran.pdf (accessed October 22, 2013)

Alex Rhomberg, 2001, Real-time finite elements: A parallel computer application, Doctoral Thesis, Swiss Federal Institue of Technology, Zurich, Publisher: Shaker, http://dx.doi.org/10.3929/ethz-a-004089534

Amira, n.d., Amira 3D Software for Life Sciences, http://www.amira.com/ (accessed July 14, 2012)

Amy Elizabeth Kerdok, 2006, Characterizing the Nonlinear Mechanical Response of Liver to Surgical Manipulation, Ph.D Thesis, Harvard University

Ang K.C., 2008, Introducing the Boundary Element Method with MATLAB, International Journal of Mathematical Education in Science and Technology, Vol. 39, No. 4, pp. 505-519

Ang W. T., 2007, A Beginner's Course in Boundary Element Methods, Universal Publishers, Boca Raton, USA

ANSYS, n.d., ANSYS - Simulation Driven Product Development, http://www.ansys.com/ (accessed April 28, 2014)

Biorobotics, n.d., Harvard Biorobotics Laboratory, http://biorobotics.harvard.edu (accessed April 28, 2014)

Blaise Barney, n.d., Message Passing Interface (MPI), Available from: https://computing.llnl.gov/tutorials/mpi/ (accessed October 22, 2013)

Bo Zhu, Lixu Gu, 2012, A hybrid deformable model for real-time surgical simulation, Computerized Medical Imaging and Graphics, 36(2012), pp. 356-365

Brebbia C. A., 1978, The Boundary Element Method for Engineers, London/New York: Pentech Press/Halstead Press

Brebbia C. A., Dominguez J., n.d., Boundary Elements: An Introductory Course, Second Edition, WIT Press, Available from: http://www.boundaryelements.com/index.php?option=com_content&view=category&layout=blog&id=908&Itemid=43 (accessed December 27, 2012)

C. Mendoza, C. Laugier, 2003, Simulating Soft Tissue Cutting using Finite Element Models, Proceedings ofthe 2003 IEEE International Conference on Robotics & Automation, Taipei, Taiwan, September 14-19, 2003

C. Monserrat, U. Meier, M. Alcaniz, F. Chinesta, M.C. Juan, 2001, A new approach for the real-time simulation of tissue deformations in surgery simulation, Computer Methods and Programs in Biomedicine, 64 (2001), pp. 77–85

CAELinux, n.d., Open-source powered engineering, http://www.caelinux.com (accessed 25th August 2012)

Chen G, Li XC, Wu GQ, Zhang SX, Xiong XF, Tan LW, Yang RG, Li K, Yang SZ, Dong JH., 2010, Three-dimensional reconstruction of digitized human liver: based on Chinese Visible Human, Chinese Medical Journal, 123(2), p. 146-50

Dong Sun Shin, Jin Seo Park, Byeong-Seok Shin, Min Suk Chung, 2011, Surface models of the male urogenital organs built from the Visible Korean using popular software, Anatomy & Cell Biology, 44(2), p. 151-159

Firat Dogan, M. Serdar Celebi, 2011, Real-time deformation simulation of nonlinear viscoelastic soft tissues, Simulation, Volume 87, Issue 3, March 2011, pp. 179-187, doi:10.1177/0037549710364532

Foster, Timothy,Mark, 2013, Real-time stress analysis of three-dimensional boundary element problems with continuously updating geometry, Doctoral thesis, Durham University

G. Karami, D. Derakhshan, 2001, A field boundary element method for large deformation analysis of hyperelastic problems 8(2), p. 110-122, SCIENTIA IRANICA

Gernot Beer, Ian Moffat Smith, Christian Duenser, 2008, The Boundary Element Method with Programming: For Engineers and Scientists, Springer

GNU, n.d., GNU General Public License, http://www.gnu.org/licenses/gpl.html (accessed October 17, 2014)

H. Pongrac, B. Farber, P. Hinterseer, J. Kammerl, E. Steinbach, 2006, Limitations of human 3D-force discrimination, Human-Centered Robotics Systems

Hammond Simon D., Mudalige Gihan R., Smith J. A., Jarvis Stephen A., Herdman J. A., Vadgama A., 2009, WARPP - A Toolkit for Simulating High-Performance Parallel Scientific Codes, In: 2nd International Conference on Simulation Tools and Techniques (SIMUTools09), Rome, Italy, 2-6 Mar 2009, Published in: SIMUTools '09 2nd International Conference on Simulation Tools and Techniques Article no. 19

Helsinki BEM, n.d., Helsinki BEM Library, http://peili.hut.fi/BEM/ (accessed April 28, 2014)

Henry Gray, 1918, Anatomy of the Human Body, Philadelphia, Lea & Febiger

Herve Delingette, 1998, Toward Realistic Soft-Tissue Modeling in Medical Simulation, Proceedings of the IEEE, Vol. 86, No. 3, March 1998, pp. 512-523

Herve Delingette, Nicholas Ayache, n.d., Soft Tissue Modeling for Surgery Simulation, Online resource from http://www.inria.fr/centre/sophia/ (accessed October 25, 2013)

Hong Z. Tan, Brian Eberman, Mandayam A. Srinivasan, Belinda Cheng, 1994, Human Factors for the Design of Force-reflecting Haptic Interfaces, DSC-Vol.55-1, Dynamic Systems and Control, Editor: Clark J. Radcliffe, Book No. G0909A-1994, The American Society of Mechanical Engineers

Hong-jian Gao, 2007, 3-D Reconstruction of Liver Slice Images Based on MITK Framework, The 1st International Conference on Bioinformatics and Biomedical Engineering (ICBBE 2007), DOI: 10.1109/ICBBE.2007.247

http://www.boundary-element-method.com/, n.d. (accessed April 28, 2014)

http://www.efunda.com, n.d. (accessed October 22, 2013)

Hui-Shen Shen, 2000, Nonlinear bending of simply supported rectangular Reissner–Mindlin plates under transverse and in-plane loads and resting on elastic foundations, Engineering Structures 22(7), p. 847-856

I. Prieto, A.L. Iban, J.A. Garrido, 1999, 2D analysis for geometrically non-linear elastic problems using the BEM, Engineering Analysis with Boundary Elements, 23 (1999), pp. 247–256

ImageJ, n.d., Image Processing and Analysis in Java, http://rsbweb.nih.gov/ij/ (accessed July 14, 2014)

Intel Math Kernel Library Reference Manual, n.d., Available from: http://software.intel.com/sites/products/documentation/hpc/mkl/mklman/index.htm (accessed October 22, 2013)

Issenberg SB, McGaghie WC, Hart IR, Mayer JW, Felner JM, Petrusa ER, Waugh RA, Brown DD, Safford RR, Gessner IH, Gordon DL, Ewy GA., 1999, Simulation Technology for Health Care Professional Skills Training and Assessment, The Journal of the American Medical Association, 282(9), p. 861-6

ITK-SNAP, n.d., ITK-SNAP Home Page, http://www.itksnap.org (accessed May 5, 2014)

Jay T. Bishoff, Louis R. Kavoussi, n.d., LAPAROSCOPIC SURGERY OF THE KIDNEY, Available at: http://www.pkdiet.com/pdf/Laparoscopic%20nephrectomy.pdf (accessed July 14, 2012)

Katia Bertoldi, Michele Brun, Davide Bigoni, 2005, A new boundary element technique without domain integrals for elastoplastic solids, International Journal for Numerical Methods in Engineering 64(7), p. 877-906

Kirana Kumara P, 2011, Reconstructing Solid Model from 2D Scanned Images of Biological Organs for Finite Element Simulation, Google Knol, Available from: http://knol.google.com/k/kirana-kumara-p/reconstructing-solid-model-from-2d/3rc2kfwq179j2/5# (accessed April 28, 2014)

Kirana Kumara P, 2012a, Extracting Three Dimensional Surface Model of Human Kidney from the Visible Human Data Set using Free Software, Leonardo Electronic Journal of Practices and Technologies, 11 (20), pp. 115-126

Kirana Kumara P, 2012b, A MATLAB Code for Three Dimensional Linear Elastostatics using Constant Boundary Elements, International Journal of Advances in Engineering Sciences (IJAES), 2 (3). pp. 9-20

Kirana Kumara P, 2012c, Demonstrating the Usefulness of CAELinux for Computer Aided Engineering using an Example of the Three Dimensional Reconstruction of a Pig Liver, International Journal of Advancements in Technology, Vol 3, No 4, pp. 301-309

Kirana Kumara P, 2013, A study on the usefulness of Support Vector Machines for the realtime computational simulation of soft biological organs, Seventh M.I.T. Conference on Computational Fluid and Solid Mechanics -- Focus: Multiphysics & Multiscale, June 12-14, 2013, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, Available from: http://eprints.iisc.ernet.in/46772/

Kirana Kumara P, 2014a, Codes for solving three dimensional linear elastostatic problems using constant boundary elements while ignoring body forces, Available from: http://eprints.iisc.ernet.in/48088/ (accessed January 10, 2014)

Kirana Kumara P, 2014b, A Study of Speed of the Boundary Element Method as applied to the Realtime Computational Simulation of Biological Organs, Electronic Journal of Boundary Elements, Vol. 12, No. 2, pp. 1-25

Kirana Kumara P, Ashitava Ghosal, 2010, A Procedure for the 3D Reconstruction of Biological Organs from 2D Image Sequences, International Conference on Biomedical Engineering and Assistive Technologies (BEATs-2010), 17-19 December, Dr. B R Ambedkar National Institute of Technology, Jalandhar, India

Kirana Kumara P, Ashitava Ghosal, 2011a, Graphics Processing Units for the Real-time Linear Elastostatic Simulation of Liver, International Conference on Advanced Computing & Communication Technologies (ACCT11), 20-22 January, Rohtak, India

Kirana Kumara P, Ashitava Ghosal, 2011b, Real-time Computer Simulation of Three Dimensional Elastostatics using the Finite Point Method, 2011 The 2nd International Conference on Mechanical, Industrial, and Manufacturing Technologies (MIMT 2011), Feb.26-28, Singapore

Kirana Kumara P, Ashitava Ghosal, 2012, Real-time Computer Simulation of Three Dimensional Elastostatics using the Finite Point Method, Applied Mechanics and Materials, 110-16, pp. 2740-2745

Li Lou, Shu Wei Liu, Zhen Mei Zhao, Pheng Ann Heng, Yu Chun Tang, Zheng Ping Li, Yong Ming Xie, Yim Pan Chui, 2009, Segmentation and reconstruction of hepatic veins and intrahepatic portal vein based on the coronal sectional anatomic dataset, Surgical and Radiologic Anatomy, 31:763-768

M. Brun, D. Capuani, D. Bigoni, 2003, A boundary element technique for incremental, non-linear elasticity Part I: Formulation, Computer Methods in Applied Mechanics and Engineering 192, p. 2461-2479, doi:10.1016/S0045-7825(03)00268-8

M. Guiggiani, A. Gigante, 1990, A General Algorithm for Multidimensional Cauchy Principal Value Integrals in the Boundary Element Method, Journal of Applied Mechanics, Transactions of the ASME, Vol. 57, December 1990, pp. 906-915

M. Guiggiani, n.d., Formulation and numerical treatment of boundary integral equations with hypersingular kernels, Contained in: “Singular Integrals in B. E. Methods, V. Sladek and J. Sladek, eds., 1998”

M.H. Aliabadi, 2006, Boundary element method for shear deformable plates with combined geometric and material nonlinearities, Engineering Analysis with Boundary Elements 30(1), p. 31-42

M. Kreienmeyer, E. Stein, 1995, Parallel implementation of the boundary element method for linear elastic problems on a MIMD parallel computer, Computational Mechanics, 15 (1995), pp. 342-349

Marek Gzik, Wojciech Wolanski, Dawid Larysz, 2011, Cut to the Bone - Simulation reduces surgery duration and patient risk in treating infants with skull disorders, ANSYS Advantage, Volume V, Issue 2, pp. 26-27, Available from: http://www.ansys.com/staticassets/ANSYS/staticassets/resourcelibrary/article/AA-V5-I2-Cut-to-the-Bone.pdf (accessed October 24, 2013)

Mark Adams, James W. Demmel, 1999, Parallel Multigrid Solver for 3D Unstructured Finite Element Problems, ACM 1-58113-091-8/99/0011, Available from: http://www.cs.illinois.edu/~snir/PPP/mesh/multigrid3.pdf (accessed October 30, 2013)

Martin Bayliss, 2003, The Numerical Modelling of Elastomers, PhD Thesis, School of Engineering, Cranfield University

MathWorks, n.d., MATLAB and Simulink for Technical Computing, http://www.mathworks.com/ (accessed April 28, 2014)

MATLAB Central, n.d., BEM Code for 2D Pulsating Cylinder, File Exchange, MATLAB Central, http://www.mathworks.com/matlabcentral/fileexchange/16074-bem-code-for-2d-pulsating-cylinder (accessed April 28, 2014)

Max A. Woodbury, 1950, Inverting modified matrices, Memorandum Rept. 42, Statistical Research Group, Princeton University, Princeton, NJ, 4pp MR 38136

MeshLab, n.d., http://meshlab.sourceforge.net/ (accessed May 5, 2014)

Mimics, n.d., Medical Image Segmentation for Engineering on Anatomy, http://biomedical.materialise.com/mimics (accessed July 14, 2012)

Min Li, Yun-Hui Liu, 2005, Modeling Interactions of Pulpal Tissue with Deformable Tools in Endodontic Simulation, Proceedings of the 2005 IEEE International Conference on Robotics and Automation, Barcelona, Spain, April 2005, pp. 2637-2642

NIH, n.d., National Institutes of Health, http://www.nih.gov/

NLM, n.d., National Library of Medicine, http://www.nlm.nih.gov/ (accessed July 14, 2012)

O. Köhler, G. Kuhn, 2001, The Domain-Boundary Element Method (DBEM) for hyperelastic and elastoplastic finite deformation: axisymmetric and 2D/3D problems, Archive of Applied Mechanics (Ingenieur Archiv) 71(6-7), p. 436-452, doi:10.1007/s004190100153

Olek C Zienkiewicz, Robert L Taylor, J.Z. Zhu, 2013, The Finite Element Method: Its Basis and Fundamentals, Seventh Edition, Butterworth-Heinemann

Ottensmeyer M, Salisbury K, 2001, In Vivo Data Acquisition Instrument for Solid Organ Mechanical Property Measurement, MICCAI: Medical Image Computing and Computer-Assisted Intervention 4th International Conference, Utrecht, The Netherlands, Springer-Verlag

Oztireli, Guennebaud, Gross, 2009, Feature Preserving Point Set Surfaces based on Non-Linear Kernel Regression, Eurographics 2009

P.H. Wen, M.H. Aliabadi, A. Young, 2005, Large deflection analysis of Reissner plate by boundary element method, Computers & Structures 83(10-11), p. 870-879

P.M. Baiz, M.H. Aliabadi, 2007, Buckling analysis of shear deformable shallow shells by the boundary element method, Engineering Analysis with Boundary Elements 31(4), p. 361-372

P Wang, A A Becker, I A Jones, A T Glover, S D Benford, M Vloeberghs, 2009, Real-time surgical simulation for deformable soft-tissue objects with a tumour using Boundary Element techniques, Journal of Physics: Conference Series, Volume 181, Number 1, doi:10.1088/1742-6596/181/1/012016

P. Wang, A.A. Becker, I.A. Jones, A.T. Glover, S.D. Benford, C.M. Greenhalgh, M. Vloeberghs, 2007, Virtual reality simulation of surgery with haptic feedback based on the boundary element method, Computers and Structures, 85(2007), pp. 331-339

Parallel ESSL Guide and Reference, n.d., Available from: http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/ (accessed November 29, 2013)

Paul A. Yushkevich, Joseph Piven, Heather Cody Hazlett, Rachel Gimpel Smith, Sean Ho, James C. Gee, Guido Gerig, 2006, User-guided 3D active contour segmentation of anatomical structures: Significantly improved efficiency and reliability, Neuroimage, 2006 Jul 1;31(3):1116-28

Rasband W.S., n.d., ImageJ, U. S. National Institutes of Health, Bethesda, Maryland, USA, http://rsb.info.nih.gov/ij/, 1997-2009

ReMESH, n.d., ReMESH - Edit and Repair Polygon Meshes, http://remesh.sourceforge.net/

Rhinoceros, n.d., http://www.rhino3d.com (accessed May 5, 2014)

Robert G Aykroyd, Brian A Cattle, Robert M West, n.d., A boundary element approach for real-time monitoring and control from electrical resistance tomographic data, 4th World Congress on Industrial Process Tomography, Aizu, Japan

S. De, J.-W. Hong, K. J. Bathe, 2003, On the method of finite spheres in applications: towards the use with ADINA and in a surgical simulator, Computational Mechanics, 31 (2003), pp. 27–37, DOI 10.1007/s00466-002-0390-3

Salisbury, n.d., Real-Time Finite Element Analysis (FEA) in Haptic Surgical Simulation, http://web.stanford.edu/group/sailsbury_robotx/cgi-bin/salisbury_lab/?page_id=343 (accessed July 13, 2014)

ScaLAPACK - Scalable Linear Algebra PACKage, n.d., Available from: http://www.netlib.org/scalapack/ (accessed October 22, 2013)

ScaLAPACK Example Programs, n.d., Available from: http://www.netlib.org/scalapack/examples/ (accessed October 22, 2013)

SensAble technologies, 2005, OPENHAPTICS™ TOOLKIT version 2.0, PROGRAMMER'S GUIDE

Simulation Acceleration using System Objects, MATLAB Coder and Parallel Computing Toolbox, n.d., Documentation Center, MathWorks, Available from: http://www.mathworks.in/help/comm/examples/simulation-acceleration-using-system-objects-matlab-coder-and-parallel-computing-toolbox.html (accessed October 30, 2013)

Sonya Allin, Yoky Matsuoka, Roberta Klatzky, 2002, Measuring Just Noticeable Differences For Haptic Force Feedback: Implications for Rehabilitation, Proceedings of the 10th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (HAPTICS'02), IEEE Computer Society

Stephane Bordas, 2008, A Ph.D.+MSc Position on Brain Surgery Simulation by XFEM and FleXFEM, http://imechanica.org/node/3239 (accessed July 13, 2014)

Stephen A. Jarvis, Daniel P. Spooner, Helene N. Lim Choi Keung, Junwei Cao, Subhash Saini, Graham R. Nudd, 2006, Performance prediction and its use in parallel and distributed computing systems, Future Generation Computer Systems, 22 (2006), pp. 745–754

"STLA Files - ASCII stereolithography files", n.d., "liver.stl", the ASCII STL file, a human liver using 38142 triangular faces and 19073 nodes, Available from: http://people.sc.fsu.edu/~jburkardt/data/stla/stla.html (Accessed March 14, 2015)

Suvranu De, Jung Kim, Yi-Je Lim, Mandayam A. Srinivasan, 2005, The point collocation-based method of finite spheres (PCMFS) for real time surgery simulation, Computers and Structures, 83 (2005), pp. 1515–1525, doi:10.1016/j.compstruc.2004.12.003

T. Dirgantara, M.H. Aliabadi, 2006, A boundary element formulation for geometrically nonlinear analysis of shear deformable shells, Computer Methods in Applied Mechanics and Engineering 195(33-36), p. 4635-4654

Taylor V.E., Chen J., Disz T.L., Papka M.E., Stevens R., 1996, Interactive virtual reality in simulations: exploring lag time, Computational Science & Engineering, IEEE (Volume: 3, Issue: 4), pp. 46 - 54, Winter 1996

Text Mechanic, n.d., A collection of simple, single task, browser based, text manipulation tools, Available from: http://textmechanic.com/ (accessed December 2, 2014)

Trevor G. Davies, Xiao-Wei Gao, 2006, Three-dimensional elasto-plastic analysis via the boundary element method, Computers and Geotechnics 33(3), p. 145-154

TurboSquid, n.d., 3D Models for Professionals, http://www.turbosquid.com (accessed July 14, 2012)

U. Meier, O. Lopez, C. Monserrat, M. C. Juan, M. Alcaniz, 2005, Real-time deformable models for surgery simulation: a survey, Computer Methods and Programs in Biomedicine, 77(2005), pp. 183-197, doi:10.1016/j.cmpb.2004.11.002

Ullrich Meier, Carlos Monserrat, Nils-Christian Parr, Francisco Javier Garcia, Jose Antonio Gil, 2001, Real-Time Simulation of Minimally-Invasive Surgery with Cutting Based on Boundary Element Methods, Medical Image Computing and Computer-Assisted Intervention - MICCAI 2001, Lecture Notes in Computer Science, Volume 2208, pp. 1263-1264

Valerie E. Taylor, Milana Huang, Thomas Canfield, Rick Stevens, Daniel Reed, Stephen Lamm, 1996, Performance Modeling of Interactive, Immersive Virtual Environments for Finite Element Simulations, International Journal of High Performance Computing Applications, June 1996, vol. 10, no. 2-3, pp. 145-156, doi: 10.1177/109434209601000203

VHD, n.d., The Visible Human Project - Getting the Data, http://www.nlm.nih.gov/research/visible/getting_data.html

VHP, n.d., Visible Human Project, http://www.nlm.nih.gov/research/visible/visible_human.html (accessed July 14, 2012)

Victor W. Lee, Changkyu Kim, Jatin Chhugani, Michael Deisher, Daehyun Kim, Anthony D. Nguyen, Nadathur Satish, Mikhail Smelyanskiy, Srinivas Chennupaty, Per Hammarlund, Ronak Singhal, Pradeep Dubey, 2010, Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU, ACM SIGARCH Computer Architecture News - ISCA '10, Volume 38 Issue 3, June 2010, pp. 451-460

VOXEL-MAN, n.d., Example Scene: Correlation of CT with 3D and Cross-Sectional Anatomy, VOXEL-MAN 3D-Navigator: Inner Organs, http://www.voxel-man.de/vm/io_correlation.en.html (accessed July 14, 2012)

W. T. Ang, 2007, A Beginner's Course in Boundary Element Methods, Universal Publishers, Boca Raton, USA

Warren C. Young, Richard G. Budynas, 2002, Roark's Formulas for Stress and Strain, Seventh Edition, McGraw-Hill

Wei-Zhe Feng, Xiao-Wei Gao, Jian Liu, Kai Yang, 2015, A new BEM for solving 2D and 3D elastoplastic problems without initial stresses/strains, Engineering Analysis with Boundary Elements 61, p. 134-144

Xiao-Wei Gao, Trevor G. Davies, 2000, An effective boundary element algorithm for 2D and 3D elastoplastic problems, International Journal of Solids and Structures 37(36), p. 4987-5008

Yi-Je Lim, Suvranu De, 2007, Real time simulation of nonlinear tissue response in virtual surgery using the point collocation-based method of finite spheres, Comput. Methods Appl. Mech. Engrg., 196 (2007), pp. 3011–3024

Yijun Liu, n.d., Fast Multipole Boundary Element Method (FastBEM) Software for Education, Research and Further Development, CAE Research Lab, University of Cincinnati, Ohio, USA, Available from: http://urbana.mie.uc.edu/yliu/Software/ (accessed April 28, 2014)

Youssef F. Rashed, n.d., Available from: http://www.boundaryelements.com/ (accessed October 22, 2013)

List of Publications

Journals

[1] Kirana Kumara P, Ashitava Ghosal, "Real-time Computer Simulation of Three Dimensional Elastostatics using the Finite Point Method," Applied Mechanics and Materials, 2012, 110-16, pp. 2740-2745.

[2] Kirana Kumara P, "Extracting Three Dimensional Surface Model of Human Kidney from the Visible Human Data Set using Free Software," Leonardo Electronic Journal of Practices and Technologies, 2012, 11 (20), pp. 115-126.

[3] Kirana Kumara P, "A MATLAB Code for Three Dimensional Linear Elastostatics using Constant Boundary Elements," International Journal of Advances in Engineering Sciences (IJAES), 2012, 2 (3). pp. 9-20.

[4] Kirana Kumara P, "Demonstrating the Usefulness of CAELinux for Computer Aided Engineering using an Example of the Three Dimensional Reconstruction of a Pig Liver," International Journal of Advancements in Technology, 2012, Vol 3, No 4, pp. 301-309.

[5] Kirana Kumara P, "A Study of Speed of the Boundary Element Method as applied to the Realtime Computational Simulation of Biological Organs," Electronic Journal of Boundary Elements, 2014, Vol. 12, No. 2, pp. 1-25.

Conferences

[1] Kirana Kumara P, Ashitava Ghosal, "A Procedure for the 3D Reconstruction of Biological Organs from 2D Image Sequences," International Conference on Biomedical Engineering and Assistive Technologies (BEATs-2010), 17-19 December, 2010, Dr. B R Ambedkar National Institute of Technology, Jalandhar, India.

[2] Kirana Kumara P, Ashitava Ghosal, "Graphics Processing Units for the Real-time Linear Elastostatic Simulation of Liver," International Conference on Advanced Computing & Communication Technologies (ACCT11), 20-22 January, 2011, Rohtak, India.

[3] Kirana Kumara P, Ashitava Ghosal, “Real-time Computer Simulation of Three Dimensional Elastostatics using the Finite Point Method,” 2011 The 2nd International Conference on Mechanical, Industrial, and Manufacturing Technologies (MIMT 2011), Feb.26-28, 2011, Singapore.

[4] Kirana Kumara P, “A Study on the Usefulness of Support Vector Machines for the Realtime Computational Simulation of Soft Biological Organs,” Seventh M.I.T. Conference on Computational Fluid and Solid Mechanics - Focus: Multiphysics & Multiscale, June I2-14, 2013, Massachusetts Institute of Technology, USA.

Codes

[1] Kirana Kumara P, 2014, “Codes for solving three dimensional linear elastostatic problems using constant boundary elements while ignoring body forces,” Available from: http://eprints.iisc.ernet.in/48088/ (accessed January 10, 2014)

Preprints

[1] Kirana Kumara P, 2015, Simulations using meshfree methods, Available from: http://arxiv.org/abs/1506.02808 (accessed June 19, 2015)

[2] Kirana Kumara P, 2015, A literature survey on the real-time computational simulation of biological organs, Available from: http://vixra.org/abs/1506.0122 (accessed June 19, 2015)

Blog

[1] Kirana Kumara P, 2011, “Reconstructing Solid Model from 2D Scanned Images of Biological Organs for Finite Element Simulation,” Google Knol, Available from: http://knol.google.com/k/kirana-kumara-p/reconstructing-solid-model-from-2d/3rc2kfwq179j2/5# (accessed April 28, 2014)

Appendix: Explanation of Some Terminologies Related to Parallel Computing

BLACS: The BLACS (Basic Linear Algebra Communication Subprograms) is a collection of routines that may be used to support a linear algebra oriented message passing interface. The advantage of having BLACS is that it makes it possible to implement efficient and uniform linear algebra oriented message passing interfaces, across many different distributed memory computers.

blacs_barrier: The 'blacs_barrier' is a routine that is used to sychronize the processes. The routine achieves the synchronization by holding up the execution of the subsequent portion of the program, for all the processes, until all the processes finish calling this routine.

Block Distribution, and Block-Cyclic Distribution: The two types of data distribution are explained here with reference to a one dimensional array. The explanations presented here for the vector in one dimension can be applied independently over two dimensions (for the rows and columns of a matrix) and that results in a matrix being distributed among processes as per the concerned distribution. The following explanations (including formulae) for Block Distribution and Block-Cyclic Distribution are copied/adapted from the website of the IBM Knowledge Center. Block Distribution assigns blocks of size r of the global vector of M data objects over P processes. For Block Distribution, the mapping $m \longrightarrow (p, i)$ is defined as: $m \longrightarrow (floor(m/L), m \bmod L)$, where $L = ceiling (M/P)$, m is the global index of a data object $(0 \leq m < M)$, p $(0 \leq p < P)$ specifies the process to which the data object is mapped, and i specifies its location in the local array. In the Block-Cyclic Distribution, blocks of r consecutive data objects are distributed cyclically over the P processes. This can be described by a mapping of the global index m to an index triplet (p,b,i), where b is the block number in process p, and other symbols have the same meaning as earlier. Hence, for Block-Cyclic distribution, the mapping $m \longrightarrow (p, b, i)$ is defined as: $m \longrightarrow (floor((m \bmod T)/r), floor(m/T), m \bmod r)$, where $T = rP$.

Block size: When a matrix or a vector is to be distributed among the available processes, one needs to divide the matrix or the vector into pieces. The number of rows in a piece is called the 'row block size' while the number of columns in the piece is called the 'column block size'.

CUDA: CUDA (Compute Unified Device Architecture) is a parallel computing platform by NVIDIA (NVIDIA produces GPUs). CUDA is very useful when GPUs are used as processors (accelerators in fact).

maxNumCompThreads: The MATLAB command 'maxNumCompThreads' specifies the number of threads that MATLAB should use (while running MATLAB). The maximum value that 'maxNumCompThreads' can take is equal to the number of computational cores (on any machine).

MPI: Message Passing Interface (MPI) is a standardized and portable message-passing system. It can be used to program many of the parallel computers.

Parallel Computing Toolbox: The MATLAB Parallel Computing Toolbox makes it possible for some of the MATLAB codes and scripts to be executed on a GPU (only a subset of the MATLAB features is supported), right from within the MATLAB.

parallel.gpu.gpuArray.zeros: The 'parallel.gpu.gpuArray.zeros' is a command available through the MATLAB Parallel Computing Toolbox. The command is used to initialize an array in the GPU memory with zeros.

parfor, matlabpool, MATLAB client, MATLAB lab, MATLAB worker, matlabpool close: These terminologies are related to the MATLAB Parallel Computing Toolbox, specifically when one wants to utilize multiple cores available in computers. The 'parfor' loop is a replacement for the standard MATLAB 'for' loop when one wants to run the statements inside the 'for' loop on a number of cores in parallel. The command 'matlabpool' enables the parallel features in the MATLAB. The process which starts the parallel environment is called 'MATLAB client' while the other processes in the parallel environment are called 'MATLAB labs'. MATLAB labs are also known as 'MATLAB workers'. The command 'matlabpool close' disables the parallel features in the MATLAB language.

Process grid: If the one-dimensional array of processes of a parallel machine is mapped into a two-dimensional rectangular grid, then the grid is called the process grid. The number of rows in a process grid when multiplied by the number of columns in the process grid gives the total number of processes.

ScaLAPACK: ScaLAPACK (Scalable Linear Algebra PACKage) is a library of linear algebra routines for parallel computers.