SIMULATION OF TETRAHEDRAL MESH BASED ORGAN DEFORMATION USING PARAMETER OPTIMIZED SPRING CONSTANTS

by

KOYEL MUKHERJEE

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2008

ACKNOWLEDGEMENTS

I am very grateful to my thesis supervisor Dr. Venkat Devarajan for his constant supervision, encouragement and valuable guidance throughout the course of my Master's degree and thesis. He has been a constant source of inspiration and advice and has always been there to help me out whenever I got stuck in my work. I especially appreciate his time and effort in guiding me throughout the thesis writing process.

I am very grateful to Dr. Michael T. Manry and Dr. W. Alan Davis for agreeing to be in my thesis supervising committee and for all their help, cooperation and guidance.

I am indebted to Xiuzhong for his constant help and guidance all throughout my work.

I also wish to thank Mr. Prashant Chopra and Dr. Joerg Meyer for having responded very kindly and quickly to my questions regarding their work and having explained everything clearly which helped me to make further progress in my work. I also want to extend my thanks to Mr. James D. Fleming of HPC systems in UTA for responding quickly to all my doubts and queries which helped me get my work done.

I am extremely grateful to my labmate, Dibbesh Adhikari for having helped me get an initial foothold on this area of research to which I was new, and having helped out in other ways too. I am also very grateful to labmates and friends, Kriti, Ajay, Rupin and other friends for having been constant sources of support and help.

I thank my family for having borne with me at my times of frustration and having kept me going with all their help and support. I thank GOD for having blessed me and my family to help us achieve what we have so far and I have faith on his continued blessings on us.

June 26, 2008

ABSTRACT

SIMULATION OF TETRAHEDRAL MESH BASED ORGAN DEFORMATION USING PARAMETER OPTIMIZED SPRING CONSTANTS

Koyel Mukherjee, M.S.
The University of Texas at Arlington, 2008

Supervising Professor: Venkat Devarajan

Virtual reality based surgical simulators are becoming increasingly popular for training doctors on minimally invasive surgery. For these simulators, it is necessary to generate 3D organ models and create virtual environments depicting visually realistic deformation response by tissues and organs to manipulations by surgical tools. For creating a 3D model, several different approaches have been used by the computer graphics community, among which polygonal/polyhedral meshes have gained popularity for real time virtual reality applications.

This work investigates two problems in 3D organ modeling for virtual reality simulators. Firstly, we consider the use of unstructured 3D meshes for generation of organ models. We investigate a class of algorithms for simplification of meshes, that is, for reduction in the number of elements, and also for improving the quality of generated mesh. We propose a new algorithm for mesh simplification by building up on existing approaches. This algorithm has better and quicker simplification performance. At the same time we ensure that the quality of simplified mesh elements is not degraded beyond a certain user-specified limit. We compare the results obtained by our algorithm to existing rapid procedures.

The second problem investigated in this work deals with mass-spring-damper models, as applied to the tetrahedral mesh for modeling the physically deformable nature of the organs. We extend the work reported by Wang and Devarajan in 2007 [1] and apply the suggested parameter optimization to the more general case of irregularly shaped organs with a higher deformation tendency. We optimize the Hooke's constant for the springs used in the Mass-Spring-Damper (MSD) model in the mesh based on some constraints to generate a more realistic and visually appealing physically-based deformation response. We set up a stand-alone framework for modeling the deformation response of MSD model based organs and determine all relevant environment parameters for a stable simulation.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 3D Modeling

In 3D computer graphics, 3D modeling is the process of developing a mathematical representation of any three-dimensional object (either inanimate or living) via specialized software. A 3D model can be displayed as a two-dimensional image through a process called rendering or used in a computer simulation of physical phenomena.

3D models have a variety of applications. Detailed organ modeling is used in virtual medical applications. Modeling of characters and objects for animation and real-life motion pictures is common in the film industry. Figure 1.1 is an example of 3D model creation for animation in a film. 3D models are widely used in computer and video games. Other applications include detailed modeling of chemical compounds, designing new devices, vehicles and structures, buildings and landscapes, geological modeling and a host of other areas.

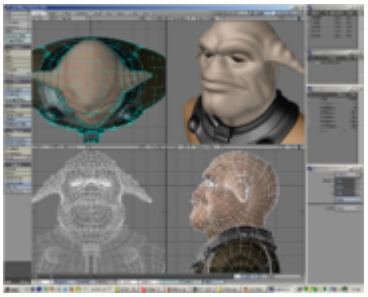


Figure 1.1 A 3D model of a “Mangalore” from the film The Fifth Element in the 3D modeler LightWave, shown from different perspectives

1.1.1 Modeling Process

Creation of a 3D model requires specification of the 2D surfaces or surface modeling, as well as representation of volumes surrounded by surfaces, known as solid modeling. Figure 1.2 shows

the well known model of the Utah Tea Pot, widely used in education as an example of an object that requires modeling of smooth curved surfaces.

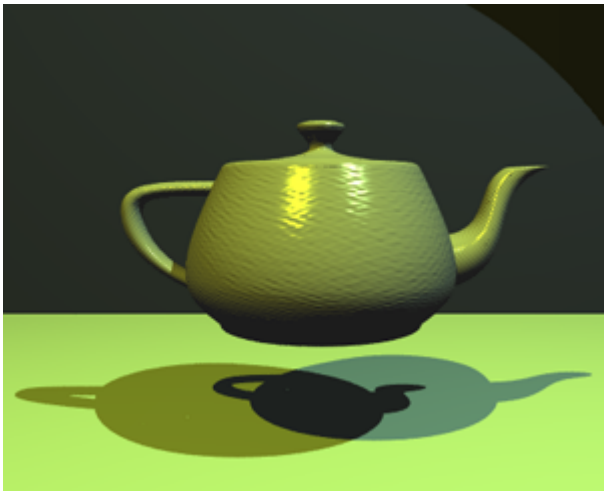


Figure 1.2 The Utah Teapot

Popularly used techniques for surface modeling are as follows [19] :

- Polygon Mesh - It is a set of connected edges, vertices and polygons, where each edge connects two vertices, and is shared at most by two polygons, and each polygon is a closed sequence of edges. Examples are triangular meshes, quadrilateral meshes etc.

- Parametric Polynomial Curves – Points on a 3D curve are specified by three polynomials defined in terms of a variable parameter, one for each dimension x, y and z. Commonly used Curves are Hermite Curves, Bezier Curves, various forms of B-Splines (curve segments whose local behavior is controlled by 'weighted' control points) like Uniform Non-rational B-Splines, Non-Uniform Non-Rational B-Splines and Non-Uniform Rational B-Splines (also known as NURBS).

- Parametric Bivariate Polynomial Surface Patches – Coordinates of points on curved surfaces are defined by three bivariate polynomials, one for each of x, y and z. The boundaries of the patches are parametric curves (if both are cubic, then they are known

as Bicubic Surfaces). Examples are Hermite Surfaces, Bezier Surfaces, B-Spline Patches.

- Implicit Surfaces – In this case, surfaces are defined implicitly by an equation *f(x,y,z) = 0.* If *f* is a quadric polynomial in x, y and z, it is referred to as a Quadric Surface. They are used for representations of spheres, cylinders etc.

  Commonly used solid modeling techniques are as follows [19, 21]:

  - Primitive Instancing – A set of primitive 3D solid shapes are defined that are relevant to the application area. These primitives are typically parameterized to define a 'family' of parts, with similar behavior and small variations.

  - Sweeps – A new object is defined by sweeping an object through a trajectory in space. Translation Sweep or Extrusion is defined by a 2D area swept along a linear path normal to the plane of the area to create a volume. Rotational Sweeps are defined by rotating areas about axes. General Sweeps follow arbitrary curved trajectories.

  - Boundary Representations (B-rep) – This involves description of an object in terms of its surface boundaries: vertices, edges and faces. Some B-reps are restricted to planar, polygonal boundaries. Polyhedral meshes, e.g., tetrahedral meshes (used in this work) fall under this category, and are discussed in detail in later sections.

  - Spatial Partitioning Representations – A solid is decomposed into a collection of adjoining, non-intersecting solids which are more primitive than the original solid and may not be the same type as the original solid. Commonly used approaches are Cell Decomposition, Spatial Occupancy Enumeration (voxel based method), Octrees and Binary Space-Partitioning (BSP) trees.

- Constructive Solid Geometry (CSG) – Simple primitives are combined by means of regularized Boolean set operators. An object is stored as a tree with operators at the internal nodes and the simple primitives at the leaves.

Modeling can be performed by means of a dedicated program (e.g., Maya, 3DS Max, Blender) or an application component (Shaper, Lofter in 3DS Max) or some scene description language (as in POV-Ray).

1.1.2 Illumination, Shading and Texture

Illumination can be modeled as having three components [19]: Ambient, Diffuse and Specular. While ambient light models a diffuse, non-directional source of light, diffuse and specular lighting mainly model lighting effects due to a point source of light and are responsible for varying light reflections and shininess across object surfaces, depending on the light and object positions and object material. Figure 1.3 illustrates the illumination types.

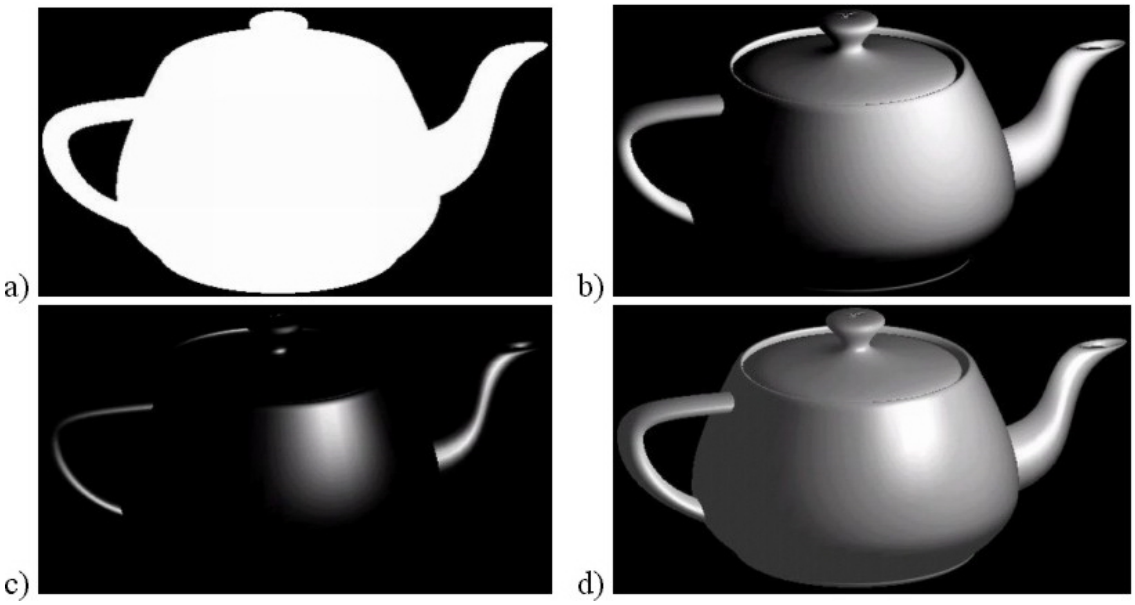


Figure 1.3 Lighting [a) Only Ambient Light, b) Only Diffuse Light, c) Only Specular Light, d) All three types of Lighting present] [Source: http://www.naturewizard.com/tutorial0107.html]

Shading refers to the method used for calculating lighting effects as applied on a particular object. Commonly used models are flat shading, Gouraud Shading and Phong Shading.

The method of applying 2D texture image to the model's surface to create a photorealistic effect is called texture mapping. Texture images are normal digital images which are assigned special information called texture coordinates or UV coordinates to indicate which parts of the texture image map to which parts of the 3D model's surface. Figure 1.4 is an example of a 3D model with texture mapping, lighting and shading effects.

Figure 1.4 A fully textured rendering of a 3d model with lighting effects

One of the main applications of 3D models is in virtual reality (VR) based surgical simulators for the purpose of training surgeons. The work presented in this thesis is mainly applicable in this area. The following sections provide some background information on this area and the motivation behind this work.

1.2 Application of 3D models in VR based Training Systems

1.2.1 Importance of VR based Surgical Training Systems

Traditionally, surgical residents receive hands-on training by observing experienced surgeons and progressively performing surgical steps in the course of patient care [29]. Using real patients as resources for practice might result in potential harm to them due to mistakes of the beginners. Also, experienced surgeons might need training with new and upcoming surgical

procedures. Especially, in minimally invasive surgery (MIS), where a camera and surgical instruments are inserted in a patient's body through small incisions, surgeons are handicapped by the limitations of current technology, like limited visualization, difficult hand-eye coordination, limited haptic (sense of touch) feedback and restricted translational movements.

MIS, which includes laparoscopic (Figure 1.5) and endoscopic surgery, is gaining in popularity because it does not require large, painful incisions. The pain and scarring associated with surgical procedure are considerably less and the patient can return to normal activities much faster [25, 29]. However, due to the very limited operational space, difficult manipulation and lack of depth information from the two-dimensional image, surgeons usually have to be extensively trained to avoid accidents and reduce complications and recurrence rate [25, 29]. Training on cadaver and live animals incurs huge costs and criticism on ethics [25]. Plastic models or mannequins do not provide realistic or dynamic experience. Therefore, in order to provide realistic visual and haptic cues, it is becoming increasingly important to build virtual reality (VR) based surgical simulators.

Figure 1.5 Scene from an actual Laparoscopic Surgery [22]

1.2.2 Steps in developing a VR based Surgical Simulator [22,29]

A virtual reality based surgical simulator lets the trainee touch, feel, and manipulate virtual tissues and organs through the same surgical tool handles used in actual MIS while viewing images of tool-tissue interactions on a monitor as in real laparoscopic procedures [29] These images are created to be as realistic as possible and to mimic the look and feel of a patient's anatomy.

Figure 1.6 shows the main processing blocks in developing a VR based surgical simulator. The very first step is the generation of 3D anatomical models of organs from medical images, labeled Offline Processing. This is generally accomplished using segmentation and reconstruction techniques of computer vision and computer graphics. Next, the deformation responses of soft tissues and organs are modeled.

The deformation models created are used for both graphic visualization as well as haptic modeling. Collision detection and response techniques have to be developed to simulate the real-time interactions of simulated surgical instruments and the manipulated organs. In order to realistically portray the response of tissues and organs to such manipulations, both the graphical display as well as force feedback are important, and are implemented through effective deformable modeling.

Finally, the real time controls and synchronizations have to be set in place. Rendering with texture mapping and special effects generates the visual display. For enabling haptic feedback, the feedback force needs to be calculated and the haptic devices need to be interfaced with the software simulation environment. This is necessary for the tactile response.

In this work, we are mainly concerned with the offline processing module, more specifically, 3D organ modeling, framework set up, and deformation response modeling.

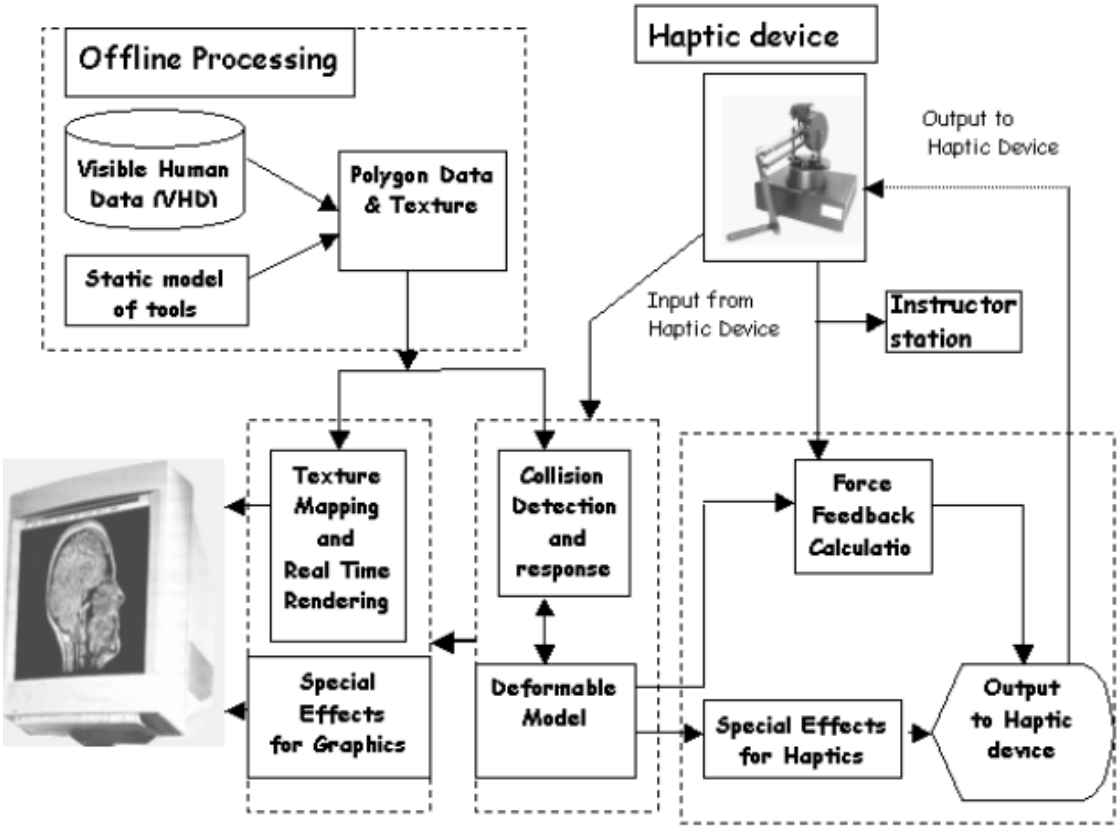


Figure 1.6 Steps in Developing a VR based Surgical Simulator [25]

1.2.3 Motivation and Problem Statements

In this work, we mainly concentrate on the first two steps of building a VR based simulator, or in other words, developing a 3D model of a human body organ with a more realistic real time deformation response than currently used real time models.

For this work, we have generated a 3D model of an organ using tetrahedral meshes. Mass spring damper (MSD) model has been used to simulate the physically deformable nature of the organ. Since real time performance is important, we used polygonal modeling for 2D surface modeling and a polyhedral mesh for 3D volume modeling. The tetrahedral mesh used to model the organ is generated from an existing 2D triangular surface mesh. Also, we have used

the MSD model to generate deformation response in real time, in place of other more accurate but computationally intensive models.

Our objective is to develop a 3D unstructured organ model and then use parameter optimization on an existing 3D MSD model to generate a better visual response. We extend the work by Wang and Devarajan [1], in which they have applied parameter optimization to 3D models in regular symmetric isotropic bodies with a small number of mesh elements. Here, we apply the same parameter optimization method for unstructured meshes to the modeling of irregularly shaped human organs, with different elastic and material properties compared to regular solids, and which require a very large number of mesh primitives for faithful reproduction of the irregular shape. We accomplished this by generating a tetrahedral mesh, simulating a virtual MSD based model, optimizing its parameters and updating them as needed, determining the values of other relevant parameters for a stable virtual organ simulation, and rendering the organ in real time.

However, in the course of performing this task, we encountered another problem. The mesh size was becoming prohibitively large for processing the parameters without using high performance computing resources. Though we did use a high computing facility for performing the parameter optimization calculations, this motivated us to explore the highly researched field of mesh simplification and quality improvement. We investigated some of the existing algorithms and proposed, developed and tested a new algorithm for mesh simplification, or reduction of the number of mesh elements without further deterioration of the quality of the mesh.

The problem statements for the two main problems investigated in this work can thus be summarized as follows:

Problem Statement 1: Development of an efficient and rapid mesh simplification and quality improvement algorithm, and its implementation.

Problem Statement 2: Simulation of a 3D organ model using unstructured tetrahedral meshes and MSD principles and the calculation of optimized MSD parameters and other relevant

parameters for the realization of a stable virtual reality based system, exhibiting realistic visual deformation response to applied displacements and force.

1.3 Organization of the Thesis

Chapter 1 gives an overview of the main aspects and commonly used techniques of 3D modeling and also the main problems investigated in this work. Chapter 2 deals with mesh simplification and mesh quality improvement. Mesh construction is explained and previous work on mesh simplification is investigated. A novel mesh simplification and quality improvement algorithm is proposed as an improvement over the existing approaches. Chapter 3 explains the workings of some physically based deformable models, with special emphasis on MSD systems and outlines previous work done in the optimization of a MSD model. The parameter based spring constant optimization method, originally suggested in [1, 25] and which we have extended in this work for simulating deformation response of a 3D organ model is explained. Chapter 4 discusses the implementation details of both the mesh simplification algorithm as well as the optimized mass spring damper system and presents the results obtained. It also suggests possible future work in both these areas.

CHAPTER 2

TETRAHEDRAL MESH SIMPLIFICATION AND QUALITY IMPROVEMENT

Triangles and quads (four sided polygons) are the most common shapes used in polygonal modeling. A group of polygons connected together by shared vertices and edges in a non-self intersecting manner is referred to as a mesh. We provide some background information on the generation of triangular and tetrahedral meshes in Appendix A.

2.1 Polyhedral Mesh Simplification – Previous Work

Three dimensional meshes consisting of tetrahedral elements are widely used in most computer graphics and animation applications. They are also used in finite element analysis, surgical simulators, three dimensional image reconstructions etc. In order to generate realistic models, often these meshes are highly detailed. As a result, three dimensional tetrahedral meshes commonly used in engineering and research applications generally have a very large number of mesh primitives, namely vertices, faces, edges and tetrahedra. The data volume becomes too large at times for realistic possibility of further processing or handling of the data. In such situations, it is necessary to reduce the number of tetrahedral elements to generate a mesh with smaller number of primitives, which can then be processed in systems with limited memory and processing capabilities. At the same time, it is sometimes necessary to improve the quality of meshes generated. In fact, for many of the critical applications of three dimensional meshes, it is necessary to ensure that the quality of tetrahedral elements meets a certain criteria.

Mesh simplification is commonly applied to both triangular and tetrahedral elements as a preprocessing step to reduce their geometric complexity. In the following section we outline some of the previous research [2-16] in mesh simplification, both triangular and tetrahedral, that is relevant to our work.

For 2D meshes, using triangular surface elements, retriangulation into smaller triangles has been suggested by some approaches [8-9]. 2D meshes have been simplified using surface fitting approach too, as suggested by Turk [10] and to some extent by Kalvin et al [8]. Schroeder et al [9] suggest making multiple passes through the mesh and removing vertices selectively. Retriangulation is thereafter used to patch up holes left by vertex removals. Kalvin et al [8] present an algorithm to approximate the original mesh within a specified error tolerance.

For simplification of tetrahedral meshes, the most common operation is edge collapse [6-7] (Figure 2.1). Other operations commonly used for three dimensional mesh simplifications are edge split, edge swap, vertex split (the reverse of edge collapse) and vertex merge. Cignoni et al [5] provide a comprehensive study of mesh simplification operations. They also propose an iterative edge-collapse based method for tetrahedral mesh simplification in [4]. In most of the 'edge collapse'-based decimation methods, one of the main issues explored were prevention of mesh inconsistencies due to simplification and decimation operations [4, 7,13]. While many methods have been developed for accurate error evaluation metrics for the simplified mesh [4,13], most of these algorithms are slow and hence become inapplicable to high volume data sets.

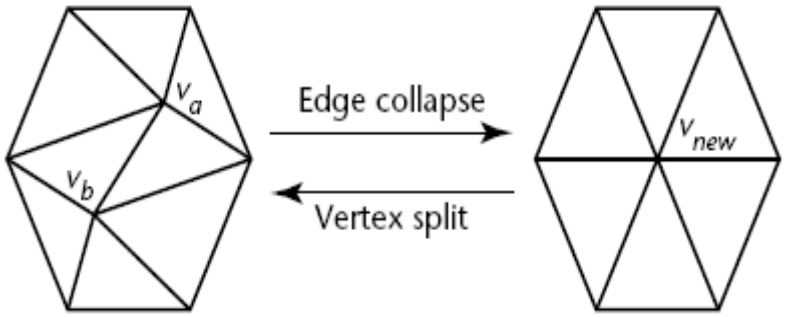


Figure 2.1 Diagram illustrating Edge Collapse [65]

M. Garland [16] in his survey report on multi-resolution modeling presents a detailed description of decimation strategies used in polygonal meshes. A tetrahedral collapse operation has been suggested by Trotts et al. [12] as a tetrahedral mesh reduction technique. They define a tetrahedral collapse operation as a sequence of three edge collapses, while keeping the overall error (based on a unique spline-segment representation of each tetrahedron) below a tolerance range. However, due to the use of 'edge-collapse' operations, the time and space complexities for this algorithm are high.

'Edge-collapse' has been used as the basic mesh reduction operation by Staadt and Gross [7] in their work on progressive tetrahedralization as well as by Hoppe in his seminal work [6]. The algorithm stated in [7] preserves the topological and geometric features and handles problems like flipping (negative tetrahedra) and tetrahedron-boundary intersections at concave interiors. However, this algorithm is also time consuming.

Trotts et al. [13] in extension of their earlier work on tetrahedral mesh simplification [12] revert to a single edge collapse as the atomic decimation operation.

Error evaluation techniques have been explored by Cignoni et al. in [4]. Local accumulation, gradient difference, and brute force strategies are used to evaluate error introduced in a tetrahedral mesh due to 'edge-collapse' based operations.

2.2 Our Approach to Mesh Simplification

Most of the earlier work deals with mesh simplification, instead of mesh quality improvement. The simplification process might itself introduce deterioration in mesh quality, which is generally ignored. Also, the computational overload and time complexity of most of the earlier approaches are quite daunting.

We propose an algorithm to simplify and improve the quality of tetrahedral meshes using a systematic and controlled approach. A new kind of data structure is proposed to store the connectivity information of the meshes. This structure allows us to employ a step-by-step simplification algorithm which controls the quality of mesh generated, at the same time.

We found that Tetfusion suggested by Chopra and Meyer [2] in 2002 was the most efficient method for removal of tetrahedral primitives. We also found that for evaluating the quality of each tetrahedron the approach suggested in [3] by Cutler et al was a good baseline approach. Our approach modifies the approach in [2] and uses the quality definition in [3] to create a novel hybrid method.

Our proposed algorithm is a robust and systematic procedure which achieves rapid tetrahedral deletions, and simultaneously tries to maintain a good enough mesh quality. It tries to overcome the extremely localized greedy nature of the approach in [2], while reducing the computational complexity and time complexity of the approach suggested in [3], where various other operations are used for tetrahedral deletions (instead of fusion). Also, our algorithm introduces a systematic and hierarchical processing based on the mesh connectivity which would achieve better performance than random selection as suggested in [3] by ensuring a constant check on the tetrahedral quality.

An additional novelty in our approach lies in the unique representation of mesh connectivity information, which helps us to proceed through the mesh in a controlled manner, deleting tetrahedra based on quality considerations and at the same time ensuring that the overall mesh quality does not get impaired too much by the simplification process.

2.3 Proposed Algorithm

We propose building a conceptually ‘tree’ like data structure to help us in processing the mesh in a systematic and hierarchical manner. We associate a 'level' (or depth) information with each tetrahedral element that forms a part of the mesh. The 'level' information is determined based on the mesh connectivity, as well as visual perception. We mention visual perception here to emphasize the importance that our approach associates with tetrahedral elements forming a part of the surface of the model, or in other words, tetrahedra which determine the boundary of the 3D model.

2.3.1. Preprocessing

The 'level' information increases progressively inwards from the boundary. Specifically, the tetrahedral elements which form the boundary, that is, which contain any vertex which lie on the surface of the model, are assigned a level value of '0'. Then, from the connectivity of the mesh, we determine the neighboring tetrahedral elements for each boundary tetrahedron. We then conduct a Breadth-First Search of depth one on every tetrahedron of the current level, (that is, the ones with a 'level' value of 0). Now, for each neighbor visited, if it is not a boundary tetrahedron, its level is then assigned equal to 1. The same process is then repeated for each of these level 1 tetrahedra and each of their neighbors, if not visited already (that is, if its level is not set already), is assigned a level of 2. The process is continued iteratively, namely, the neighboring tetrahedral elements are determined for each tetrahedron assigned a level in the previous step, and if these neighbors do not have a level assigned, they are assigned the next higher level from their immediate parent set. We keep a counter, which keeps track of the total number of tetrahedra with levels already assigned at any given stage of the processing. When this counter value becomes equal to the total tetrahedral count in the mesh, the process is terminated. Figure 2.2 illustrates this diagrammatically.

To store this level information, we have employed a combination of doubly linked list and dynamic array (Figure 2.3). Each node in the linked list represents one level in our proposed tree-like data structure, with the level numbers increasing from the first node onwards. Each node again contains a dynamic array, which lists all the tetrahedral elements belonging to the same level. The dynamic array grows as the number of tetrahedra in each level increases. The linked list is made doubly linked to facilitate the implementation of the second part of our proposed processing algorithm.

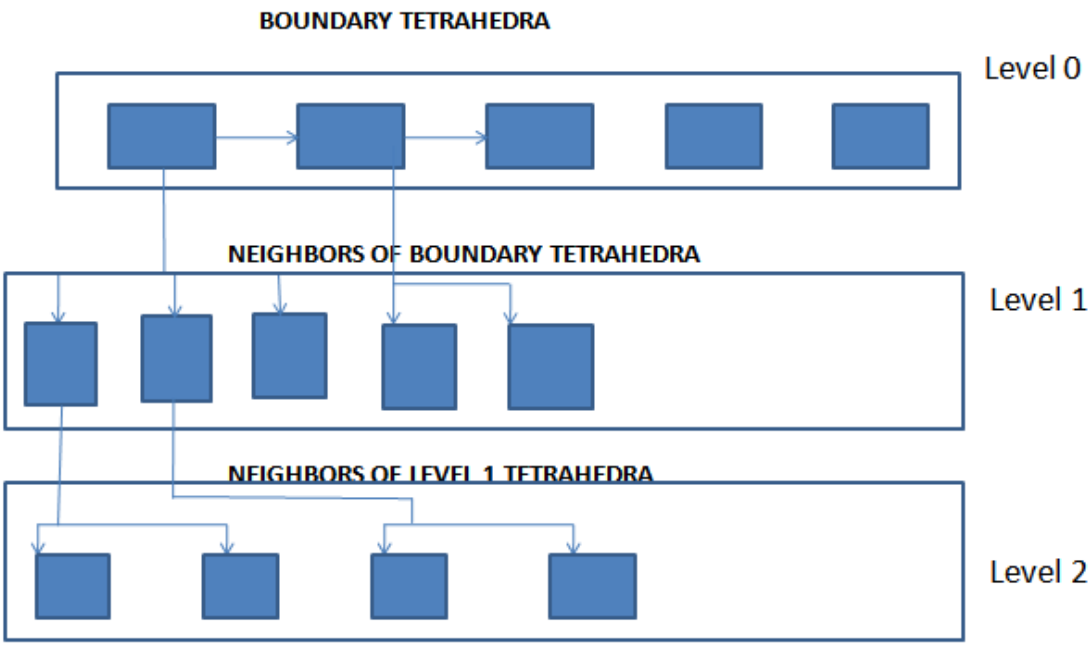


Figure 2.2 Diagrammatic Representation of proposed Data Structure

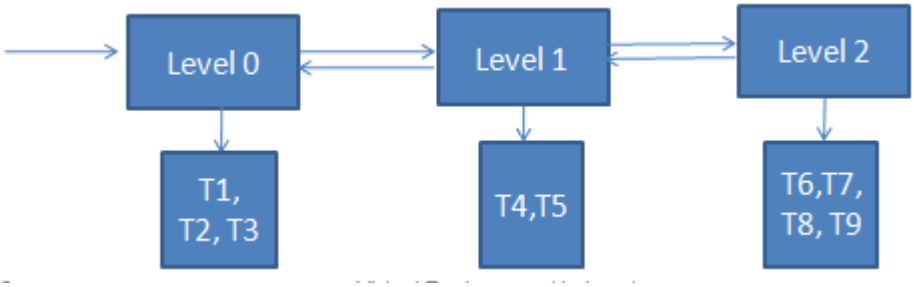


Figure 2.3 Data Structure Implementation

2.3.2. Quality Determination

At each level, we do a second stage of processing to determine the quality values of the tetrahedral elements in each stage. The quality is determined based on approaches suggested in [3], namely by using a geometric mean approach. It is evaluated as the geometric mean of several quality factors (discussed later). The minimum quality factor for each 'level' is determined, and associated with that level as one of its properties. This helps us to detect the worst shaped tetrahedral elements in every stage, that is, sliver tetrahedra, needle tetrahedra, wedge tetrahedra etc. (Figure 2.4). The final data structure form has been shown in Figure 2.5.

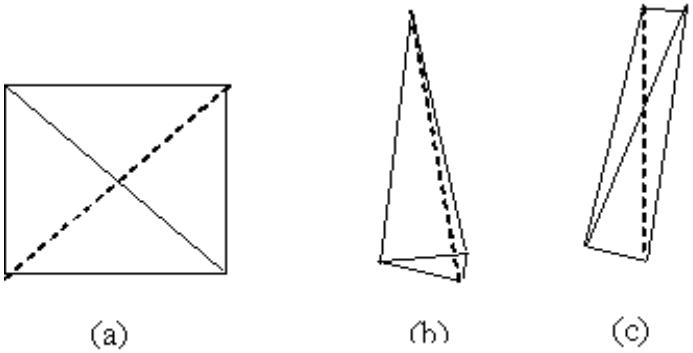


Figure 2.4 Different undesirable types of tetrahedra (poor quality factors) (a) Sliver Tetrahedra (b) Needle Tetrahedra (c) Wedge Tetrahedra

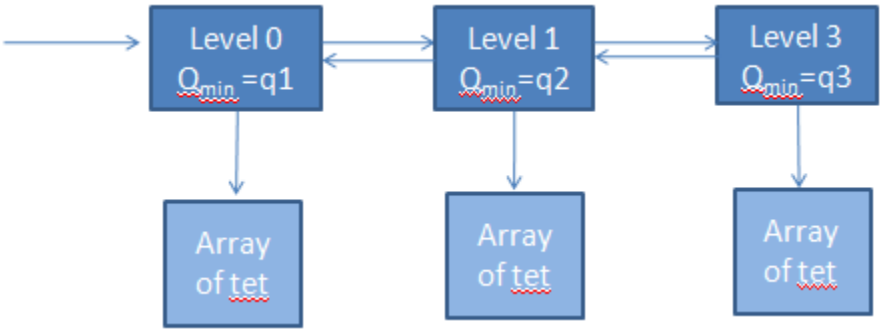


Figure 2.5 Block Diagram of Suggested Data Structure

2.3.3 Systematic Mesh Simplification

We start the simplification from the highest 'level' (or from the interior most tetrahedral elements). Hence, we start from the end node or last node of the proposed doubly linked list and progressively move backwards.

At each level, we first determine if every tetrahedral element, if not deleted already, is a possible candidate for deletion or not. We employ the technique of TetFusion illustrated in [2] for deletion of tetrahedral elements. Specifically, when every tetrahedral element is deleted, it is replaced by its barycenter, and all tetrahedral elements sharing more than one vertex with this deleted tetrahedron get deleted as well.

Before we delete any tetrahedra, we first evaluate its suitability for deletion. Firstly, we determine if this tetrahedron was deleted, how it would affect the quality of the tetrahedra in the

previous (immediate parent) level. The resultant quality of every affected tetrahedron (the ones sharing one vertex with the candidate for deletion) is measured.

The new quality factor of every affected tetrahedron should be greater than or equal to some set value. If the lowest quality value of the parent level was qmin, then the new quality factor can be restricted to be at least equal to qmin/100 or some such metric to limit the quality degradation. However, if the resultant quality factor falls below this set metric, then the tetrahedron under consideration is marked as unsuitable for deletion in the current pass.

Secondly, deletion of any tetrahedron should not result in flipping of connected tetrahedra, that is, it should not result in negative volume. This is again determined by the dot product of the normals to the unaffected surface of the affected tetrahedra before and after the prospective deletion.

If a tetrahedron passes the above two tests in the current level, its quality is determined. Then among all these suitable candidates for deletion, the worst tetrahedron (in terms of the quality metric) is chosen for deletion. Actual deletion of a tetrahedron has been implemented as suggested in [2]. The four vertices of the deleted tetrahedron are replaced by a single vertex at its barycenter. The tetrahedra sharing an edge or a face with this tetrahedron get deleted as well, because now their total number of valid vertices becomes less than four. The tetrahedra sharing one vertex only with the deleted ones get updated with the new vertex position (barycenter vertex) accordingly.

After the worst shaped tetrahedron among the current ones have been deleted, we repeat the procedure again. Now, the worst shaped tetrahedron among the remaining ones is deleted and so on. A counter is maintained such that when the number of tetrahedral elements for the current level falls to zero, then we proceed to the next lower (parent level: our level values decrease inwards from the boundary) and repeat this process. The transition to the next parent level may be done earlier too if the current level runs out of tetrahedral elements which are suitable candidates for deletion.

This process is continued for every node having a 'level' value of 2 or more. We exclude the tetrahedral elements forming the boundary and their immediate neighbors from this process since we are using tetrahedral fusion [2] as our tetrahedral element deletion mechanism, and we would like to preserve the original boundaries.

2.3.4. Hierarchical Mesh Quality Improvement

The approach outlined above simplifies and improves the mesh quality hierarchically, or in a level-by-level manner. We start from processing the inner most elements and progressively proceed towards the boundary. In every step, we delete the tetrahedra which are the worst shaped and hence, of the poorest quality. This way, we ensure that even if the tetrahedral elements belonging to the immediate parent level of the deleted tetrahedra get diminished in quality, the worst affected of them have a high chance of getting deleted themselves when the processing proceeds to its own level. It is because we do the quality comparison for deletion at every level.

This 'level-by-level' or hierarchical approach helps our algorithm to come quite close to generating the globally optimum solution for a simplified and improved quality mesh. Though at every level our algorithm proceeds in a locally greedy manner by choosing the worst of the available ones, we try to approach the global optimum by the systematic approach facilitated by our proposed data structure. Also, at every level, our data representation ensures that our algorithm is allowed to detect the worst shaped one among the current set for deletion, instead of being assigned the first available one while proceeding in a sequence as in [2].

2.3.5. Discussion of Quality Evaluation Procedure

We evaluate the quality factor for every tetrahedron based on the approach suggested in [3]. Our aim is to preserve the better shaped tetrahedra, while getting rid of the poorer shaped ones as much as possible.

Our metric for the shape of tetrahedra is based on solid angle measurement, edge length measurement and volume measurement [3]. The tetrahedra which are close to being a regular

tetrahedron are considered better shaped. Needle, wedge and sliver tetrahedra should be reduced and the volume of the entire model should be more regularly distributed among all the constituent tetrahedra. If the user would like to add any other metric for measurement of better shaped tetrahedra, the definition of quality factor in the implementation of our algorithm needs to be altered accordingly. It is because our work is not directed at determining what properties constitute a good tetrahedron, but at finding the most optimal solution or method to simplification and improvement (based on the application or the user's criteria) of tetrahedral meshes. Here we have followed the approach suggested previously and we elaborate here on the definition of quality factor that we have adopted following [3]. Our quality metric is measured as the geometric mean of the following three factors:

1. Minimum Solid Angle: The ratio of the absolute value of the minimum solid angle of a tetrahedron to the ideal value of 0.55 steradians => $\frac{\min(solid\ angle)}{0.55}$
2. Edge length Ratio: The ratio of the magnitudes of the shortest and the longest edge is considered => $\frac{shortest\ edge}{longest\ edge}$
3. Volume: The ratio of the tetrahedral volume to the ideal volume (given by the total volume of the 3D model divided by the total number of tetrahedra) clamped to 1 =>

$$clamp\left(\frac{Volume\ of\ Tetrahedron}{Total\ Volume / No.of\ Tetrahedra}\right)$$

Based on the above definition of quality factor, it is always going to be less than or equal to The implementation details have been presented in Chapter 4.

2.3.6. Discussion of Tetrahedral Deletion Process

We remove tetrahedra by deleting them as a whole and replacing them by a single new vertex at the barycenter. Here we follow to some extent the approach suggested by Chopra and Meyer in [2], namely Tetfusion (Figure 2.6). We discuss the tetrahedral fusion process briefly here.

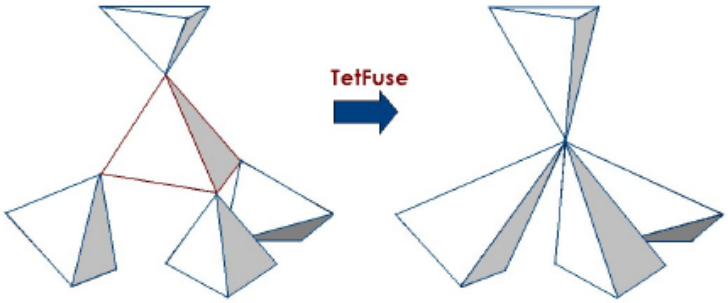


Figure 2.6 TetFusion or Tetrahedral Fusion [2]

Once a particular tetrahedron has been selected for deletion based on quality considerations, we replace its four vertices by a single vertex at its barycenter. The selected tetrahedron is of course deleted and removed from the mesh. Along with it, the tetrahedra which shared two or more vertices with the deleted one (that is, the ones which either shared an edge or a face with it), also become degenerate and are deleted. However, all of the vertices of these neighboring degenerate tetrahedra are not removed from the mesh.

The tetrahedra which shared only one vertex with the selected one (replaced by barycenter) get updated, that is, their affected vertex gets replaced by the new vertex at the barycenter. This might result in over-stretching (low solid angle or poor ratio of edge lengths) or flipping (negative volume) of the affected tetrahedra (Figure 2.7). However, we ensure by pre-checks that the affected tetrahedra will not fall too low in quality measurement. Even if they do, our hierarchical approach ensures that these affected tetrahedra themselves have a high chance of getting deleted if their quality falls below accepted values. The neighbors of the tetrahedra, who got degenerate by default due to the fusion process, do not get deleted themselves, but their shape might get altered due to change in vertex position.

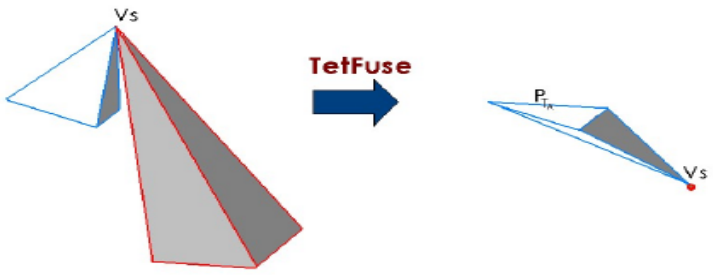


Figure 2.7 Flipping due to Tetrahedral Fusion [2]

Our algorithm employs the techniques suggested in sections 2.3.5 and 2.3.6 and attempts to find the best possible way to ensure simplified and better quality meshes. The implementation of tetrahedral deletion procedure is presented in Chapter 4.

2.3.7. Salient Features of Our Algorithm

The proposed algorithm has the following properties which make it a very good mesh simplification procedure to adopt.

1. Robustness: Our algorithm is a robust approach towards mesh simplification and improvement of mesh quality regardless of the mesh structure.

2. More Optimal Solution: It proceeds in a level by level fashion based on the mesh connectivity and at every level it picks the worst shaped tetrahedra and deletes them. This is continued for every level until a limit to tetrahedral removal is reached. Thus, we try to find a more optimal mesh structure upon simplification or tetrahedral reduction, than obtained by previous approaches. This approach however does not find the absolute global optimum (which would have entailed high computational and processing overhead). It is not restricted to proceed sequentially and pick up the first available tetrahedra, and hence avoids the entirely locally greedy nature of previous approaches. Also, it does not pick up any random tetrahedra element from the mesh, but proceeds

with some kind of regularity to ensure better overall mesh generation by removing worst affected ones as well.

3. Flexibility of being Application Specific: We do not propose any fixed metric of quality of tetrahedra. The tetrahedral quality metric definition can be altered as needed. This way our approach can be used to generate optimum simplification as required by different specifications.

4. Efficient and Rapid: Since we employ the tetrahedral fusion, our algorithm gives a rapid mesh simplification, because of the high rate of decimation inherent in the tetrahedral fusion approach.

5. Boundary Preservation: We do not process boundary tetrahedra and also those tetrahedra which are immediate neighbors of boundary tetrahedra. This ensures that the overall contour or shape of the desired 3-D model is preserved.

2.3.8. Proposed Algorithm Steps

The following steps illustrate the implementation of our algorithm.

Step 1: Build the Tree type data structure from mesh connectivity information.

- Get information about the neighboring tetrahedral elements for every tetrahedron in the mesh.

  We conduct a BFS on every tetrahedral element to determine its immediate neighbors.

- Create and initialize the proposed Data Structure.

  The linked list node form that we use is depicted below

```
Node{
        Array Tetrahedra
        Pointer to Node of next level
        Pointer to Node of previous (parent) level
        Minimum quality for this level
    }
```

For doing the above we implement the following pseudo code:

LLEV is the current value of 'level' being processed. Also, we set a flag, flag_end, as 1 when the 'level' information has been set for all the tetrahedra in the mesh.

```
Set initial value of 'level' =  LLEV = 0
Set flag_end = 0
Initialize the first Node of the linked list
Number of Tetrahedra with Level Information Set =  cnt_lev = 0
Let total number of tetrahedra in mesh = N

        For all tetrahedra T in the mesh M
            If T is a boundary tetrahedron
                Set the 'level' of T = LLEV
                Add T to the Array of first node in the list
                Increment cnt_lev by 1
                If cnt_lev =  N
                    flag_end = 1
                    stop
                End
            Else
                Continue
            End
        End
        While flag_end is not set

            Increment LLEV by 1
            Initialize a new node of the linked list and make it the current node

            For all tetrahedra T in the previous level

                Visit each neighbor Nb of T
                    If Nb has 'level' set
                        continue to next Nb
                    Set 'level' of Nb = LLEV
                    Add Nb to the array of current node
                    Increment cnt_lev by 1
                    If cnt_lev = N
                        flag_end = 1
                        stop
                    End
                    If flag_end = 1
                        stop
                    Else
                        continue to next Nb
                End
            End
        End
```

Step 2: Get the quality information

For implementing the above step, the following pseudo code is used:

```
Traverse the linked list from the start node to the end
For every node:
        Evaluate quality_factor for every tetrahedron in the Array of the node

        Find the minimum value of quality_factor among all the tetrahedra of one 'level' = qmin

       Store qmin as the minimum quality data field for the node
```

Step 3. Simplify the mesh

- Proceeding from the end of the linked list, we progressively move backwards in this step, deleting tetrahedra and simplifying the mesh step by step in each level or node of the list and we stop at 'level' 2. The pseudo-code is illustrated below:

```
'curr_level' : Maximum value of 'level' set

While 'curr_level' > 1
  n : Size of the array of 'curr_level' node

      While n>0
        For all Tetrahedra T in the array of curr_level:
            If T is not deleted:
                If T is a suitable candidate for deletion:
                    Evaluate and store the quality_factor of current T
                End
            End
        End
        If there are no T suitable for deletion:
            stop
            decrement curr_level by 1
            go to previous 'level'
       End
       Find the Tetrahedron with the lowest quality_factor = T_target

       Delete T_target and other tetrahedra sharing edge or face with T_target. Update the vertex
      of information of other affected (but not deleted) tetrahedra by the barycenter of T_target.

      Decrement n by the number of tetrahedra deleted along with T_target

    End
```

Decrement curr_lev by 1. Go to previous level

End

Since the quality evaluation and tetrahedral deletion processes are already suggested by previous work [2, 3], we do not elaborate on their pseudo-code implementations here.

- Build a new mesh from the connectivity information in the non-deleted, existing tetrahedral elements. This step can be implemented in whatever method is most efficient for the given application. So we do not suggest any pseudo code for this step.

The flowchart for the simplification part of the algorithm has been shown in Figure 2.8.

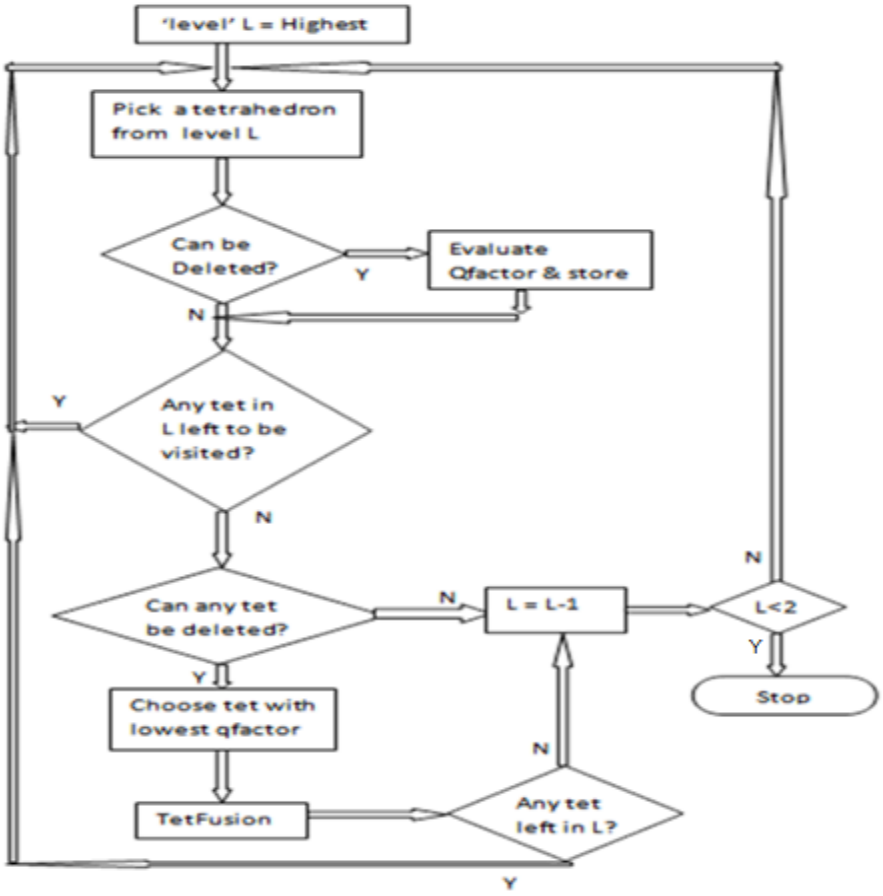


Figure 2.8 Flowchart of the Suggested Mesh Simplification Algorithm

2.3.9. Comparison to Existing Approaches

Our algorithm ensures that the mesh is progressively simplified, retaining only the tetrahedra possessing the best possible quality in each step. In this way we strive to reach the globally optimum solution for mesh simplification with better quality of mesh elements (tetrahedra). Our approach differs in quite a few aspects from the previous approaches. For example, we do not put any restriction on deletion of tetrahedra affected by prior deletions. This allows us to not only have a higher rate of mesh reduction, but also helps us to control the quality of mesh elements much more compared to existing approaches. Our processing time is somewhat higher of course, because we try to reach some approximation to the global optimum by local approximations instead of following a completely locally greedy approach.

The systematic hierarchical level by level approach gives a better quality performance compared to the sequential approach followed in [2]. It is because it allows our algorithm the luxury of choice, for picking the worst quality tetrahedron among suitable ones starting from the innermost level.

Our algorithm has considerably shorter running time compared to some of the existing approaches for simultaneous mesh simplification and improvement. Since tetrahedral fusion is much more rapid compared to other approaches like edge collapse, vertex split etc, our algorithm takes significantly less processing time for achieving the same number of tetrahedral reduction. For example, if we consider ‘edge-collapse’ as a simplification operation, then we would require at least 6 applications of ‘edge-collapse’ procedure for deleting a single tetrahedron in a mesh, since a tetrahedron has 6 edges. Compared to that, a single application of tetrahedral fusion results in the deletion of at least 11 tetrahedra from any given mesh.

Finally, we do not start with some initial target tetrahedral count (as in some previous approaches [3]), but let our algorithm simplify and refine the mesh unrestricted till it reaches the limit.

Previous work also mentions randomly picking any tetrahedron for deletion and measuring its quality [3]. However, the systematic restriction that we suggest likely helps achieve better mesh quality than would be possible in a random approach. In other words, our algorithm strikes a good balance between the sequential and random approach, and efficiently combines both performance improvement in terms of tetrahedral reduction in a short time and maintaining a better quality of resultant simplified meshes.

The results obtained by the application of the proposed algorithm have been presented in Chapter 4.

CHAPTER 3

PARAMETER OPTIMIZATION OF MASS SPRING DAMPER MODEL

3.1 Physically Based Deformable Models

Deformable models are used in Computer Aided Design (CAD) to simulate the deformation of industrial materials and tissues. In image analysis, deformable models are used for fitting curved surfaces, boundary smoothing, registration and image segmentation [27]. Deformable models have been used in computer graphics for the animation of clothing, facial expressions, and human and animal characters [28]. The modeling of deformable soft tissue and organs is very important for medical imaging, virtual surgery environments and computer assisted surgery (CAS) where interaction with virtual objects are required for generating physically realistic simulation of complex tissue and organ mechanics.

Existing modeling approaches can be physical or non-physical. Models based on purely geometrical techniques, rather than physical principles are known as non-physical models, eg. B-Splines etc.[28]. Models based on solving continuum mechanics problems under consideration of material properties and other environmental constraints are called physical models [27].

Since the seminal paper by Terzopoulos et al on elastically deformable models [26], many deformable models have been proposed. The most popular ones are the MSD model [25, 30, 40-44, 46-48], FEM [25, 33, 34], the method of finite spheres (MFS) [25, 31, 32], the elasticity theory method [37], the tensor-mass model [35], the quasi-static elastic model [35], the long element method (LEM) [36] etc. Some of these methods are explained below. The MSD or Mass Spring Damper model has been used in this work.

3.1.1 Mass-Spring-Damper Systems [25]

In the MSD model, an object is represented as a collection of point masses connected by springs in a lattice structure (Figure 3.1). It is generally used for polygonal/polyhedral objects where every vertex is assigned a mass and every edge is assigned a spring (interconnecting the mass points by the springs). Both linear and nonlinear springs can be used to model deformable objects such as human organs that exhibit nonlinear behavior. The springs exert forces on neighboring points, when a mass is displaced from its rest positions.

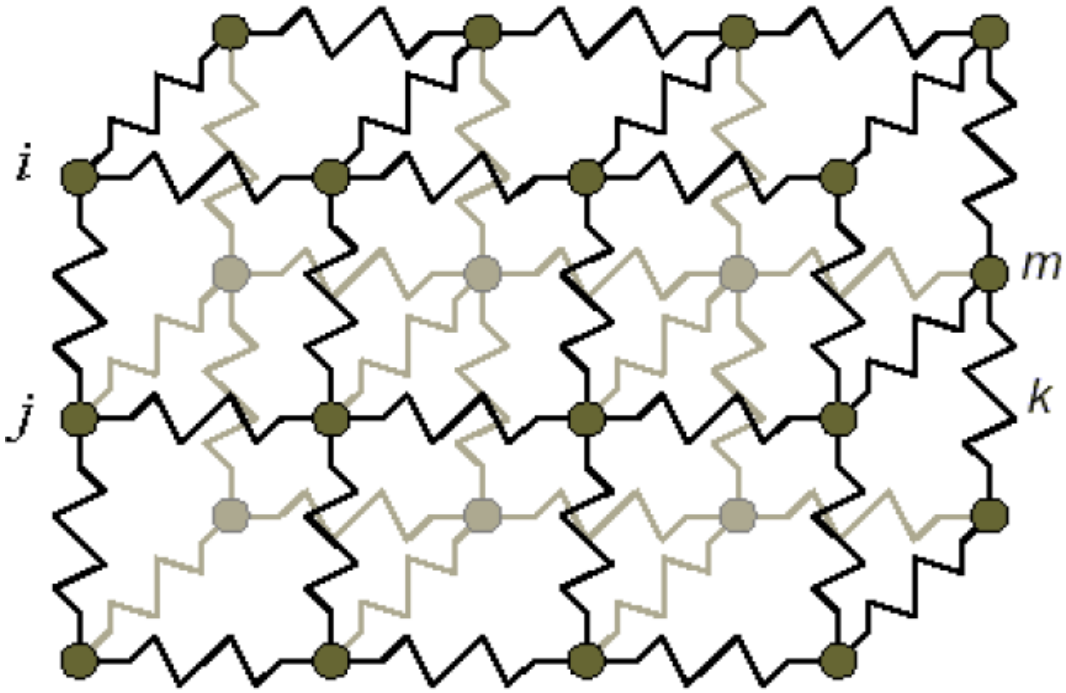


Figure 3.1 The MSD system structure [25]

In a dynamic MSD system, Newton's Second Law governs the motion of each mass point in the lattice structure:

$$m_i \ddot{\vec{r}}_i = -\gamma_i \dot{\vec{r}}_i + \sum_j \vec{f}_{ij} + \vec{f}_e \quad \text{.... (3.1)}$$

where m_i is the mass of the i[th] particle, $\vec{r}_i \in R^3$ is its position, $\vec{f}ij$ is the force exerted on it by the spring between mass particles i and j, $\vec{f}_e$ is the sum of external forces (e.g. gravity or forces applied by the user) acting on particle i, and γ_i is the damping coefficient for the resistance from the environment against motion of the mass. Vector $\dot{\vec{r}}_i$ represents the velocity vector $\vec{v}_i$ (or first derivative of position vector with respect to time) and $\ddot{\vec{r}}_i$ represents the acceleration vector $\vec{a}_i$ (or second derivative of position vector) of m_i .

Modeling the springs as linear and considering linear damping resistance for the relative motion between mass i and j, we have

$$\vec{f}_{ij} = k_{ij}\left(\left\|\vec{r}_i - \vec{r}_j\right\| - L_{ij}\right)\frac{\vec{r}_i - \vec{r}_j}{\left\|\vec{r}_i - \vec{r}_j\right\|} + \lambda_{ij}\left(\dot{\vec{r}}_i - \dot{\vec{r}}_j\right) \quad \text{..... (3.2)}$$

where L_{ij} is the rest length of the spring between mass i and j, k_{ij} is the Hooke's constant of the spring, λ_{ij} is the damping coefficient for the resistance against relative motion between mass particles i and j, which is used to model the internal (frictional) energy loss during deformation of the continuum object.

From (3.1) and (3.2), we can have

$$\ddot{\vec{r}}_i + \left(\frac{\gamma_i}{m_i} + \sum_j \frac{\lambda_{ij}}{m_i}\right)\dot{\vec{r}}_i - \sum_j \frac{\lambda_{ij}}{m_i}\dot{\vec{r}}_j - \sum_j \frac{k_{ij}}{m_i}\left(\vec{r}_i - \vec{r}_j\right) + \sum_j \frac{k_{ij}}{m_i} L_{ij}\frac{\vec{r}_i - \vec{r}_j}{\left\|\vec{r}_i - \vec{r}_j\right\|} = \frac{\vec{f}_e}{m_i} \quad \text{... (3.3)}$$

Thus for a system with N mass particles, the equation of motion can be written as:

$$\ddot{\vec{R}} + D\dot{\vec{R}} + K\left(\vec{R}\right)\vec{R} = \vec{A}_e \quad \text{..... (3.4)}$$

where D and $K(\vec{R})$ are the 3N x 3N damping matrix and stiffness matrix respectively, $\vec{R}$ is a column vector of the positions of the N masses, $\vec{A}_e$ is a column vector of the acceleration of the N masses due to external forces. D and $K(\vec{R})$ are symmetric matrices.

The system described by Eq. (3.4) is a nonlinear system since $K(\vec{R})$ is a function of $\vec{R}$. The second-order equation system above can be converted to a first-order equation system for the convenience of analysis or integration:

$$\dot{\vec{V}} = -D\vec{V} - K(\vec{R})\vec{R} + \vec{A}_e \quad \text{.... (3.5)}$$

$$\dot{\vec{R}} = \vec{V} \quad \text{.... (3.6)}$$

The above differential equation system is numerically solved using discrete time steps to approximate the continuous integration solution. That is, the acceleration, velocity and position of each mass particle are only updated at discrete time points spaced by certain time step. Many discrete numerical integration methods exist which can be used to solve the differential equation system above numerically, among which, the Euler's method is the simplest and which has been implemented in this work. An explanation of Euler's method is given in Appendix C.

Using the Euler's method and a time step of T, we have

$$\vec{V}(n+1) = (I - TD)\vec{V}(n) - TK\left(\vec{R}(n)\right)\vec{R}(n) + T\vec{A}_e(n) \quad \text{.... (3.7)}$$

$$\vec{R}(n+1) = \vec{R}(n) + \vec{V}(n)T \quad \text{.... (3.8)}$$

Advantages of MSD model:

- It is simple with easily understood dynamics
- It has a small computation burden
- It is very suitable for real-time applications with limited computing resources
- Since the MSD model has a simple discrete structure, all kinds of operations including cut and suture in the surgery can be handled easily.

The MSD model has been used to simulate facial animation (both static and dynamic, two dimensional and simplified three dimensional models [46-48]), animating fire, clouds and water [49], animation of animals [50, 51], cloth draping [52, 53], and recently in surgical simulation [54, 55].

Much research has been carried out to improve the MSD model. Research directed at improving the accuracy of the simulation involved refining the model adaptively [56] and updating Hooke's constants after refinement, controlling the isotropy or anisotropy of the material [57] etc. Many research efforts have also been directed towards improving the speed of simulation [54, 58, 59], eliminating the super elasticity phenomena [42], and handling post-buckling instability for stable but responsive simulation [60].

The two major drawbacks of the MSD model are:

- Exact deformation of real organs is not possible using the MSD model. That is, it is difficult to specify the correct values of parameters for the model.

- It is difficult to achieve fast and stable simulation. In fact, use of discrete integration schemes lead to numerical instability if a small enough value of time constant is not specified.

In the MSD model, continuous material is modeled by discrete lumps of mass and discrete springs are used to model the distributed interactions between them. Thus, it is the simplest and easiest physically based deformable model to implement, and it can handle many user interactions. However, it is not very accurate and it is difficult to transform the constraints for the continuous system to those for the corresponding discrete system.

3.1.2 Finite Element Method

FEM [34] is the most accurate method devised for modeling physically based deformation subject to certain boundary conditions. The object being modeled is decomposed into small polygonal or polyhedral meshes. The deformation field for each mesh is expressed separately by a polynomial interpolated by the displacements of the vertices of the mesh [25].

Thus a set of equations, applying the principles of continuum mechanics is obtained for each mesh with displacements and external forces set as unknowns. The resultant matrix equation is condensed to solve for the unknowns to reduce computation [33]. The integration over the mesh is calculated using Gauss product rules. Since the interpolation functions (shape functions) are polynomials, even a small number of integration points result in accurate integration. Usually, the integration is reduced to a small number of multiplications and additions [25]. The accuracy of FEM depends on the type and size of polygon or polyhedron and the number of interpolation points used. The number of interpolation points is not necessarily the number of vertices. Methods to increase the accuracy include increasing the number of meshes (h-refinement) and/or the number of interpolation points of each mesh (p-refinement). Triangular meshes are used most frequently in two dimensions (2D) and tetrahedral meshes in three dimensions (3D) and for accurate computation of deformation, usually several thousand meshes are necessary for an uncomplicated object under a simple boundary condition. As a result, the computation burden associated with FEM is too high to achieve accurate deformations in real-time. Hence, FEM is generally not used for real-time applications although in recent times some researchers have attempted to dilute the rigorous FEM approach to get it close to real time speeds.

3.1.3 Method of Finite Spheres

The method of finite spheres (MFS) [31, 32] was developed by S. De and K. J. Bathe to overcome the meshing burden for methods like FEM. It is a meshless method and uses a set of points instead of meshes to solve the governing equations. The main approach is to add some local points around the point of deformation, for example, where a surgical tool tip touches a tissue and at each of these local points a finite radius sphere is located for calculating the force deformations. Shape functions [34], similar to FEM, are used to approximate deformation fields. However, the functions chosen have to be rational, instead of polynomial. Rational functions, even if chosen carefully to minimize the computation burden, require more interpolation points than polynomial functions and hence result in higher computational load in the integration part.

For real-time application, the computation needs to be further reduced and capability for handling interactions other than point interactions needs to be introduced.

3.1.4 Tensor Mass Model

In the tensor-mass model [35] organs are meshed with conformal tetrahedrons. The mass of the object is discretized to lumped mass on the mesh points $P_i \ (i=1,2,\ldots,N)$ similar to MSD model. The governing equation for the motion of the mesh points is also based on the Newtonian Law:

$$m_i \frac{d^2 \vec{P}_i}{d t^2} = \gamma_i \frac{d\vec{P}_i}{d t} + \vec{F}_i \quad \text{..... (3.9)}$$

However, unlike the MSD model, here $\vec{F}_i$ is obtained through the energy-based finite element method. The computation of this linear elastic force can be decomposed into four steps [25] stated below:

1. The interpolation equation (shape functions) that gives the displacement vector at any point inside a tetrahedron T_k is defined as a function of the four displacement vectors at each vertex.
2. The elastic energy of a tetrahedron is expressed as a function of these four displacement vectors
3. The elastic force produced by tetrahedron *Tk* is computed and applied to vertex P_i
4. The forces $\vec{F}_{iT_k}$ produced by all the tetrahedrons connected to vertex P_i are added together to obtain $\vec{F}_i$.

$\vec{F}_i$ is computed locally, since it is only related to the tetrahedrons connected to vertex P_i and hence this method can handle cut operation and suture operation with ease, just like the MSD model. However, the tensor-mass model computes force by continuum mechanics and

therefore is independent of the mesh topology whereas the MSD model is sensitive to the mesh topology). When there is cut operation or suture operation, the tensor-mass model can provide more realistic interactions. But the major drawback of the tensor-mass method is that it is only accurate for small displacements. Because the $\vec{F}_i$'s are computed locally they won't be zero under pure rigid transformation without deformation, giving incorrect behavior in such cases.

As has been already mentioned, due to spatial discretization, the MSD model can only be made to approximate any continuum object in some aspects. In fact, A.V. Gelder [38] had demonstrated that an exact simulation using the MSD model is impossible after comparing the MSD model with finite element method. Although it is not possible to use the MSD model to realize an exact simulation, the simulation can definitely be improved by optimizing the parameter set of the given model and simulation under consideration. However, the criterion for better simulation appears ambiguous and hard to define, and realistic visual effects play an important role. One approach is to choose the parameter values on a trial and error basis, but it becomes very tedious and time-consuming. Especially for very large models (or mesh), it would become almost impossible to find optimum parameters based on a trial and error procedure. In such cases, it is necessary to follow an algorithmic approach.

3.2 Past Work

A lot of past work has been directed at estimating the parameter set from outer appearances of the MSD model. In [41], Jojic and Huang proposed to estimate cloth draping parameters from range data for cloth animation. In this work, the best fit for the cloth model was determined by comparing the drape of the model with the range data. In [39], Bhat et al proposed to estimate the parameters for cloth animation from a video based on matching between folds. Though this approach can be made to work for soft cloth by careful experimentation, it fails for linen cloth, which is more rigid, and for which the estimated parameter values turn out to be sensitive to the size of the cloth, an undesirable property.

Some work has also been directed at methods to calculate parameters for the MSD model based on the material properties of the real object. In [38], Gelder derived an approximate formula to calculate the spring constants based on the constant strain assumption in the triangular/tetrahedral mesh. In [40], Maciel et al devised a method of calculation of the spring constants based on the Hooke's law, in which the main drawback is that the simulated system obtained can pass the tensile force test in a certain direction only, but fails in other directions and other tests, like the shear test.

Most research on the MSD model parameter optimization focuses on the application of the 2D model to cloth simulation. In [42], Provot proposed a procedure to reduce the elongation of "super-elongated" springs to overcome the super-elasticity phenomenon. This method limits the deformation rate to a critical value by applying a dynamic inverse procedure to the two ends of the spring when the deformation rate becomes greater than the critical rate, thus restricting the deformation to the critical rate. Vassilev et al developed further on this method by applying a velocity directional modification approach to eliminate the super-elasticity phenomenon [43]. In [44], Bridson et al proposed a method of calculating the bending resistance according to the bending mode motion instead of using bending springs in order to model the folds and wrinkles in clothing simulation. In [45], Grinspun et al proposed modeling the bending energy as the function of the difference of the dihedral angle between adjacent triangle meshes before and after deformation in order to model curved undeformed configurations such as hats, leaves and aluminum cans. The resistant force calculation becomes complicated and the computation burden is much heavier than the pure MSD model.

Recently, Wang and Devarajan [30] have proven that triangular meshes are better for mass-spring model than rectangular meshes since the triangular mesh gives a better response in the case where there is bending force. They have showed that the results with preload are much better than the one without preloaded springs.

3.3 Three Dimensional Mass Spring Damper Model for Structured and Unstructured Mesh

Generally, for the structured case rectangular meshes are adopted for the 2D model and hexahedral meshes for the 3D model. For such cases, most of the parameters of the structured MSD model can be calculated according to the material properties, except Hooke's constant of the shear spring (needed for modeling the shearing resistance), and the flexion spring which is needed for the modeling of bending resistance in the 2D MSD model [61].

Parameter assignment for the structured MSD model has been somewhat investigated and solved to some extent [40, 61]. However, in order to model a deformable object with an irregular boundary, an unstructured MSD model is required. Parameter assignment for the unstructured MSD model is an unsolved problem. The tetrahedron is the most common 3D unstructured mesh. A new method to optimize the parameters of the 3D MSD model with tetrahedral meshes based on continuum mechanics theory has been proposed by Wang and Devarajan [1]. They have also shown that with a minor modification, the proposed method can be applied to the 3D structured MSD model to obtain the parameters of the model in an explicit form in terms of material properties and mesh geometry, giving a more complete solution than existing approaches [40, 61].

In this work, the original proposed method for regular symmetric and isotropic objects has been applied to unstructured tetrahedral mesh for a deformable model of a human organ, which is highly irregular in shape, and has a much larger number of mesh elements. The deformation response for such a model has been successfully simulated in a real time virtual environment. The following section describes the optimization procedure presented in [1].

3.4 Optimizing the 3D Unstructured MSD Model based on Continuum Mechanics [1]

Figure 3.2 shows the tetrahedral meshes of a circular shaft, which necessitates the use of unstructured meshes for efficient geometric representation. The 3D unstructured MSD model of the object is constructed on its tetrahedral meshes by assigning masses to all the vertices (nodes) and applying springs for all the edges of the tetrahedra.

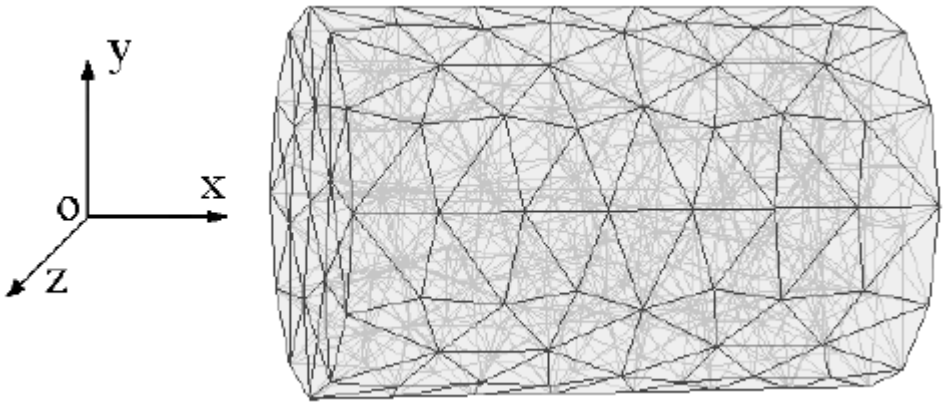


Figure 3.2 The 3D MSD model with tetrahedral meshes of a circular shaft [1]

Let $\vec{e}_{mi}$ be the unit vector emanating from node *m* along its *i*th incident spring. Let the angle between vector $\vec{e}_{mi}$ and the *z*-axis be α_{mi}, and the angle between the projection of vector $\vec{e}_{mi}$ on the *x*-*y* plane and the *x*-axis be β_{mi}. Then $\vec{e}_{mi}$ can be expressed as:

$$\vec{e}_{mi} = \left(\sin\alpha_{mi}\cos\beta_{mi}, \sin\alpha_{mi}\sin\beta_{mi}, \cos\alpha_{mi}\right)$$

It is assumed that the *i*th spring incident to node *m* is at rest when the model is in its natural rest state. Let its Hooke's constant be k_{mi}, its rest length be U_{mi}, and its displacement be u_{mi} when the model deforms. Then the energy function of the spring is

$$E_{mi}\left(u_{mi}\right) = \frac{1}{2} k_{mi} u_{mi}^2 \quad \text{.... (3.10)}$$

A physically accurate MSD model should allow stretching in the same way as the object in arbitrary directions. The model is assumed to be evenly stretched by strain, ε, in the direction of vector $\vec{n}$. It should contract transversely to minimize the potential energy. It is assumed that the material of the real object is isotropic and this transversal strain is μ, which should be a function of the tensile strain, ε. To have regular Poisson effects, μ should be small when ε is small. Then it can be assumed:

$$\mu = g(\varepsilon) = \lambda_1 \varepsilon + \lambda_2 \varepsilon^2 + o\left(\varepsilon^2\right) \quad \text{... (3.11)}$$

Let $\vec{n} = (\sin\alpha\cos\beta, \sin\alpha\sin\beta, \cos\alpha)$, where α is the angle between vector $\vec{n}$ and the *z*-axis, and β is the angle between the projection of this vector in the *x*-*y* plane and the *x*-axis. Let the angle between vector $\vec{e}_{mi}$ and vector $\vec{n}$ be ϕ_{mi} .Then,

$$\cos\phi_{mi} = \cos\alpha\ \cos\alpha_{mi} + \sin\alpha\ \cos\beta\ \sin\alpha_{mi}\ \cos\beta_{mi} + \sin\alpha\ \sin\beta\ \sin\alpha_{mi}\ \sin\beta_{mi} \quad (3.12)$$

$$u_{mi} = U_{mi}\ \sqrt{(1+\varepsilon)^2\ \cos^2\phi_{mi} + (1+\mu)^2\ \sin^2\phi_{mi} - U_{mi}} \qquad \ldots.(3.13)$$

Each spring of the MSD model is shared by two nodes. Thus, the energy increment of the model associated with node *m* under stretching is

$$W_m(\alpha, \beta, \varepsilon, \mu) = \frac{1}{2}\sum_{i=1}^{N_m} E_{mi}\ (u_{mi}) \qquad \ldots. (3.14)$$

The transversal strain minimizes the potential energy, i.e.,

$$\partial W_m(\alpha,\beta,\varepsilon,\mu) \Big/ \partial\mu = 0$$

Thus, from Eqs. (3.10), (3.11) (3.13) and (3.14), we obtain

$$\sum_{i=1}^{N_m} E_{mi}\left[\lambda_1 + (1-2\lambda_1)\cos^2\phi_{mi} + (\lambda_1 - 1)\cos^4\phi_{mi}\right] = 0 \quad \ldots (3.15)$$

According to Eq. (3.12), Eq. (3.15) can be expressed by Fourier series of α and β . To ensure that Eq. (3.15) holds for any α and β, we obtain $\lambda_1 = -1/4$, which means the only value of the Poisson constant can be achieved is -1/4. Besides, fifteen other independent constraints on the Hooke's constants of the springs incident to node *m* can be obtained.

From Eqs. (3.10), (3.11), (3.13) and (3.14), we obtain

$$\frac{dW_m(\alpha,\beta,\varepsilon,\mu)}{d\varepsilon} = \sum_{i=1}^{N_m} E_{mi}\left(\cos^2\phi_{mi} + \lambda_1\sin^2\phi_{mi}\right)^2\varepsilon + o(\varepsilon) \qquad \ldots.(3.16)$$

Let $\Delta_m = \sum_{i=1}^{N_m} E_{mi} \left(\cos^2 \phi_{mi} + \lambda_1 \sin^2 \phi_{mi} \right)^2$, which can also be expressed as a Fourier series of α and β. Since the MSD model should have the same stiffness in different directions, Δ_m should be the same for any α and β, which leads to two additional independent constraints.

Further, if all those constraints are satisfied, the stretching resistance of the MSD model associated with node *m* can be obtained as:

$$\frac{d W_m(\alpha, \beta, \varepsilon, \mu)}{d \varepsilon} = \frac{1}{6} \sum_{i=1}^{N_m} E_{mi} \varepsilon + o(\varepsilon) \qquad \text{.....(3.17)}$$

For the MSD model to be physically accurate, the stiffness of the MSD model at each node should be the same as the stiffness of the real object in the corresponding influence volume of the node, from which another independent constraint can be obtained. Thus, at each node, eighteen independent equality constraints on the Hooke's constants of the incident springs can be obtained. The equality constraints have been outlined in Appendix B. For a MSD system with *N* nodes, 18*N* constraints are obtained, which usually outnumber the springs. Under those constraints, together with the nonnegativity constraint for the stability of the model, the Hooke's constants of the springs can be solved by the constrained linear least square optimization method.

We have applied the above described optimization process to an unstructured 3D tetrahedral mesh model of a human organ using MSD model of deformation. The implementation details and results have been presented in Chapter 4.

CHAPTER 4

IMPLEMENTATION AND RESULTS OBTAINED

4.1 Mesh Generation

We used a triangular surface model of a human organ (kidney) (courtesy Dr. Yunhe Shen of Virtual Environment Laboratory) to generate the tetrahedral mesh. First, we converted the file to a .stl (Certificate Trust List) format by using 3D Studio Max (a 3D graphics application software developed by Autodesk Media and Entertainment). Then we gave this file as an input to GMSH [62] (an automatic 3D finite element mesh generator, primarily Delaunay with built-in CAD and post-processing facilities), and programmed it to generate the tetrahedral mesh from the existing triangular mesh by extruding the surface triangles inwards. The algorithm chosen for the 3D meshing was Netgen for some cases and a combination of TetGen and Delauney for other cases.

- NETGEN: NETGEN is an automatic 3D tetrahedral mesh generator. It accepts input from constructive solid geometry (CSG) or boundary representation (B-Rep) from the STL file format. NETGEN contains modules for mesh optimization and hierarchical mesh refinement. It is open source based on the LGPL license and available for Unix/Linux and Windows.
- TETGEN: TetGen generates the Delaunay tetrahedralization, Voronoi diagram, and convex hull for three-dimensional point sets, generates the constrained Delaunay tetrahedralizations and quality tetrahedral meshes for three-dimensional domains with a piecewise linear boundary.

After generation, we stored the mesh information in .mesh format, which had information regarding vertices, triangles and tetrahedra. The triangles were part of the surface mesh and were extruded to form the tetrahedral elements. We used this file as the main input to our

simulation to enable rendering of the organ mesh and build a deformation model based on MSD principles. We implemented our simulation in Microsoft Visual Studio .NET 2003 using primarily VC++, and MFC (Microsoft Foundation Classes) with OpenGL (freely available standard application programming interfaces or APIs for developing 2D and 3D computer graphics).

4.2 Object Oriented Design Framework

We designed several (C++) classes to set up the tetrahedral mesh MSD framework. For setting up a complete stand-alone framework using tetrahedral mesh and Mass-Spring-Damper model we followed an object oriented approach.

4.2.1 Mesh Formation and Processing

For implementing a mesh structure, we designed the following classes and created their objects. In building the mesh framework, we followed the approach illustrated in [63] to some extent, especially in the nomenclature of members. We now describe each of the classes, their contents and as well as functionalities in the following paragraphs:

CVertex3D:

This class implements the basic Vertex primitive of the tetrahedral mesh to be rendered. It consists of the 3D coordinate information representing the vertex in the right-handed coordinate system used for rendering, a 3D vector member m_normal which is later used to store the normal per vertex and neighbor information. Neighbor information is stored for not only other vertices it is connected to by the tetrahedron network, but also for the neighboring faces. It also has array fields to store the lengths of the springs incident on it, array of numbers identifying each such spring, and the angles alpha and beta for every such spring (which are needed to optimize parameters for the model later on). Each vertex object also has fields to store the current 3D velocity of the point mass which it represents in the mass spring damper model, as well as the current 3D force acting on it due to virtual interactions and deformable modeling. It also has a field storing the influence volume of each vertex (that is, each mass particle in the deformable MSD model). Also, it has member functions to access, modify and do

other required operations (like creating arrays based on a standard template) on these data members.

CFace3D:

This class is used to implement the Triangle (or Face) primitive of the tetrahedral mesh. It consists of data members such as an array of the CVertex3D class objects (which are its constituent vertices), an array of neighboring faces (array of CFace3D class objects), a 3D normal vector calculated for shading and lighting purposes, as well as member functions to access, modify and create these members.

CTetrehedra3D:

This class represents each constituent tetrahedral element forming the mesh. It consists of data members like an array of CVertex3D class objects representing the constituent vertices making up the tetrahedron, the barycenter of the tetrahedron, the volume etc. It also has member functions for creating, accessing and modifying these members.

CSpring3D:

This class models each spring element in the MSD model. The springs are considered to be linear and have associated spring constants (Hooke's constant). The Hooke's constant for each spring is determined by the parameter optimization method (ref: Chapter 3) and correspondingly assigned to the spring. Each spring is uniquely identified by a number in a list and has associated alpha and beta values. The damping parameter γ (gamma) is considered the same for all springs.

CMesh3D:

This class implements the complete mesh. It contains arrays of the vertices (CVertex3D), faces (CFace3D), tetrahedra (CTetrahedra3D) and springs (CSpring3D). Since each primitive array is actually implemented as an array of pointers to the storage allocated for the realization of each primitive, not much extra space is needed for implementing the CMesh3D class object (which becomes the main mesh object for rendering and modeling in our

program). It has member functions for creating, accessing and modifying each such array of primitives, and also functions to calculate normals per face and per vertex for shading and texture mapping purposes.

CVector3D:

This class implements any three dimensional vector quantities. It has data members corresponding to the three vector coordinates and member functions to create, access and modify their values. Since it implements a vector, it has member functions providing such functionalities as calculating the Inner Product and Outer Product with another vector (another object of CVector3D class), L2 norm of the vector (magnitude squared) and normalizing the vector.

4.2.2 Creation of the Mesh Structure and Springs

In the following paragraphs we explain the program flow for creating the mesh structure in the simulation.

The mesh file in .mesh format (which is basically in plain text format) is read in by our implementation program. Every new vertex read is added to the increasing vertex array for the mesh. The vertex positions are normalized to the range [-1,1]. After that the triangle information is read and the corresponding face array is created. At the time of reading in the 'face' information, an array of face neighbors for each corresponding vertex (forming the triangle) is also created. This information is required later in calculation of normals per vertex (to implement shading and texture mapping). After this, the tetrahedra information is read in and corresponding tetrahedral array is built incrementally. Along with that, for every constituent vertex of each tetrahedron, the other three constituent vertices are added to the vertex neighbor array of the former. This vertex neighbor information is necessary for the spring addition, as a spring object needs to be added between every vertex pair of each tetrahedron. So, for every new vertex neighbor added for a given vertex, a new spring object is added, its rest length is computed as the Euclidean distance between the two vertices involved in their rest position, the

corresponding alpha and beta angles are calculated from the length vector of the new spring and the x, y and z axes vectors. This information is required later in the optimization of the spring constants.

Once the original mesh information is read in from the file and storage has been allocated for it, it is simplified by the mesh simplification algorithm and a new simplified mesh is generated. Once it has been simplified, the influence volume is calculated for each node in the new simplified mesh.

4.2.3 Mesh Simplification

We have already explained the algorithm implemented in this part in detail in Chapter 2. Only the details of quality evaluation and tetrahedral deletion procedures are elaborated here.

4.2.3.1 Solid Angle of Tetrahedon

Let OABC be the vertices of a tetrahedron (Figure 4.1) with an origin at O subtended by the triangular face ABC where $\vec{a}, \vec{b}, \vec{c}$ are the vector positions of the vertices A, B and C. The vertex angle θ_a is defined to be the angle BOC and θ_b and θ_c are defined correspondingly. Let ϕ_{ab} be the dihedral angle between the planes that contain the tetrahedral faces OAC and OBC and ϕ_{bc} and ϕ_{ac} correspondingly.

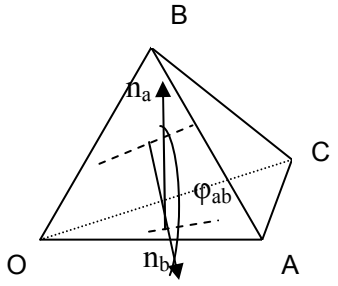


Figure 4.1 Tetrahedron with vertices OABC

[In geometry, the angle between two planes is called their dihedral or torsion angle. The dihedral angle of two planes can be seen by looking at the planes "edge on", i.e., along their line of intersection. The dihedral angle φ_{AB} between two planes denoted A and B is the angle between their two normal unit vectors $\vec{n}_A$ and $\vec{n}_B$:

$\cos\phi_{AB} = \vec{n}_A \bullet \vec{n}_B$ … (4.1)]

The solid angle at O subtended by the triangular surface ABC is given by

$$\Omega = \phi_{ab} + \phi_{bc} + \phi_{ac} - \pi \quad \text{.....} \quad (4.2)$$

For calculating the solid angle of a tetrahedron from the tetrahedral mesh, the following is done:

First, the normal for every face of the tetrahedron whose quality is being evaluated is calculated. Then, the dihedral angle for every pair of faces in the tetrahedron (6 such pairs) is calculated according to Eq. (4.1). This is done by calculating the dot product of the normal pair, dividing it by the product of the magnitudes of the normal vectors, and evaluating the arccosine of the result. The dihedral angles are then used to calculate the four solid angle values according to Eq. (4.2) (the three dihedral angles for calculating each solid angle value are chosen according to the orientation of the faces they are calculated from, as defined by the solid angle and dihedral angle illustrated above). After all four solid angles have been evaluated, the minimum among them is determined, and its magnitude divided by 0.55 is assigned to q1. 0.55 steradians is the ideal solid angle value since it is the solid angle in a regular tetrahedron.

4.2.3.2 Length of Edges

The lengths of all six edges of the tetrahedron are calculated. The longest and the shortest among them are determined. The ratio of the shortest to the longest edge is then assigned to q2.

4.2.3.3 Volume of Tetrahedron

The volume of the tetrahedron being evaluated is determined. The entire volume of the mesh is calculated and divided by the total number of tetrahedron present to determine the ideal volume per tetrahedron. The ratio of the actual volume to the ideal volume is assigned to q3.

4.2.3.4 Quality Factor

Finally the quality factor of a tetrahedron is calculated as the geometric mean of q1, q2 and q3. That is, the cube root of the product of q1, q2 and q3 is taken as the value of the quality factor for the given tetrahedron.

4.2.3.5 Tetrahedral Deletion Procedure

The barycenter of the tetrahedron marked for deletion is calculated. A new vertex (data structure) is created with the barycentric coordinates and storage is allocated for it. After this all the vertices in the tetrahedron marked for deletion are labeled as 'invalid' and the tetrahedron itself is labeled as 'deleted'. At the same time the counter for deleted tetrehedra is incremented. Then all the neighbors of the deleted tetrahedron are checked for the number of vertices they share with the latter. If they were sharing only one vertex, then this affected vertex is replaced by the new vertex created at the barycenter of the deleted tetrahedron. If the number of vertices shared was more than one, then again these degenerate tetrahedra are marked as 'deleted' and the counter of deleted tetrahedra is incremented. However, the vertices of this degenerate tetrahedra are not marked as 'invalid'. Again, those tetrahedra among the neighbors of the currently degenerate one, which have one vertex 'invalid', have their invalid vertex replaced by newly created vertex at the barycenter of the originally deleted tetrahedron.

In order to be marked for deletion, a tetrahedron first has to pass the following tests:

1) **Not a Boundary tetrahedron:** This is checked by ensuring that none of the vertices forming the tetrahedron are part of the face or surface triangle mesh, that is, for all of them, the number of face neighbors is zero.

2) **Not the Neighbor of a Boundary tetrahedron:** This is checked by running the above check on all the neighboring tetrahedra of the tetrahedron under consideration.

3) **Neighbors will not be diminished in quality beyond a certain measure:** This is considered for only those neighbors who share one vertex with the tetrahedron to be deleted. The quality factor of such neighbors is determined before and after deletion (if deletion will occur). Their ratios are evaluated and compared against the metric of 1/100 times the former quality factor.

4) **Flipping does not occur:** The normal to the base triangle of a neighbor tetrahedron sharing only one vertex with the tetrahedron being considered is calculated. Its inner product with the base vector Rb1 (vector from the base triangle to the affected vertex) is taken. Again, the inner product of Rb2 (vector from the base triangle to the newly created vertex at barycenter) is determined with the face normal. If their product is negative, flipping will occur, and hence the tetrahedron being considered for deletion is rejected.

4.2.4 Setting the Influence Volume for Each Node:

For implementing the parameter optimization process outlined above, we need to determine the influence volume of each node. It is calculated as ¼ of the sum of the volumes of all tetrahedra incident on a node (that is, all the tetrahedra of which the vertex is a part). Since we are dealing with irregular tetrahedra, the volume calculation is done by the following formula:

Specifying the tetrahedron by the three polyhedron edge vectors $\vec{a}$, $\vec{b}$ and $\vec{c}$ from a given polyhedron vertex, the volume is:

$$V = \frac{1}{3!}\left|\vec{a}\cdot\left(\vec{b}\times\vec{c}\right)\right| \qquad \text{.... (4.3)}$$

Thus, for each node or vertex, of each of the incident tetrahedron, the three incident edge vectors are calculated, and the corresponding volume determined by the inner and outer products as specified above.

4.2.5 Calculation of Spring Constants

We calculate the spring constants for each spring by the parameter optimization process outlined above. This is an offline process.

For the tetrahedral mesh used in the implementation of our simulation, the number of vertices was originally 1503 and after mesh simplification 1242, and the number of springs was initially around 9998, and after simplification 7959. For such a mesh size, the matrix sizes to be used in the simplification function would be 1242 X 7959 per constraint and there are 18 such constraints. For constrained least squares simplification, we wrote a MATLAB .m file using the built-in function lsqlin(), and another built-in function optimset() was used to set the conditions of solution, that is, the maximum number of iterations allowed for the constrained solution and the tolerance level for error.

We set the maximum number of iterations at 100,000,000 and the tolerance level at 1.0e-300. The processing was done using the High Performance Computing System (HPC) available as a computing resource in UT Arlington.

[HPC: UTA's High Performance Computing environment combines multiple independent systems connected via a private high-speed network to solve complex problems in numerous disciplines. The servers, collectively "the system", operate with Intel IA64 and Intel EM64T processor architectures and use Red Hat Enterprise Linux as their operating environments. The system is based on a Client/Server model architecture where users on a client (the root nodes) request that jobs be performed on a server (the compute nodes).]

The parameter optimization calculation is done on HPC and the end results are stored in a text file, which is read in during the actual execution of the program (carried out in an Intel Core Due Processor T2300@1.66Ghz and 1GB RAM) before starting the rendering and

execution in real time mode. This essentially means reading and associating optimized spring constants with the correct spring objects in the MSD tetrahedral mesh, identifying every spring by a unique number and associated alpha and beta angles.

4.2.6 Calculation of Volume and Center

The volume of the entire mesh is calculated as the sum of the volumes all the tetrahedra forming the mesh. The geometric center is calculated as the average of the barycenters of all the tetrahedra forming the mesh. Both of these measurements are associated with the mesh and used later in realizing stability in the implementation of the mass-spring-damper model.

4.2.7 Scene Set-up

The OpenGL context is created and the pixel format is set up. The initial geometry settings are also specified along with the view angle and viewing coordinates. These are preliminary steps to make rendering possible and for the organ to be visible at a certain desired angle. The lighting and shading parameters are also set up. All three lighting, namely, ambient, diffuse and specular lighting conditions are specified.

4.2.8 Rendering Details

The rendering is done in the OnPaint() function, which is called by the Windows message WM_PAINT generated whenever the rendering area/window is invalidated (that is, refreshed and drawn again). Two Timer threads are set up to control the rate of updating and rendering of the vertices. The Timer thread INTEGRATE_CHANGE is set up to run at intervals of 5 ms and it controls the rate of updating the positions and velocities of the vertices due to forces applied (according to MSD model and discrete integration scheme used). The Timer thread FIG_CHANGE is set up to run at intervals of 20 ms and it controls the rate of refreshing the screen and rendering the organ with the currently updated vertex positions. This ensures that the frame rate is more than the required rate of 33 Hz (for real time visual rendering).

For rendering the surface triangle mesh is used to make it faster (since the number of triangles to be rendered is much less than the number of tetrahedra). However, for implementing the MSD system and the discrete integration schemes the original tetrahedral mesh is used. Since the vertices have been stored as pointers in the array referred to by both the triangle surface mesh as well as the tetrahedral 3D volume mesh, the rendering by either mesh will represent the currently updated vertex position and hence the current organ position and form. So, we choose the triangular mesh for rendering because of the lesser number of primitives to be rendered and hence at a higher frame rate.

We have used the Open GL numeric constant GL_TRIANGLE_STRIP for rendering the list of triangles which makes it faster and more efficient than a triangle by traingle rendering (that is, rendering each triangle separately using GL_TRIANGLES). Before rendering, the current normals per face and per vertex are calculated and updated, and used for texture mapping coupled with lighting and shading properties.

For texture mapping, we chose a 32x32 .bmp image. We used a linear filtering method for both magnification and minification filtering, and implemented 2D texture mapping by wrapping across both S and T dimensions. The shading model chosen is smooth and back face culling is enabled to increase efficiency of rendering. Modulation of texture with given lighting conditions is also enabled along with normal mapping, so as to make the curves and deformations more clearly visible. The image for texture mapping was obtained from the database of VRBase simulator, designed by Dr. Yunhe Shen et al of the Virtual Environment Laboratory at UT Arlington.

The shading model implemented is Gouraud Shading, as the normals used for calculation of lighting effects are calculated per vertex, by averaging the normals of all incident faces (which had been calculated per face).

4.2.9 Discrete Integration Scheme

Euler's method is used to implement the numerical integration scheme for simulating the MSD model. The integration and numerical update for all the vertices in the system takes place every time the Timer event INTEGRATE_CHANGE takes place (that is, every 5 ms).

First, the force is calculated by processing every spring. For every spring in the data structure, the two end vertices are retrieved. The force on either of them (same magnitude but oppositely directed) due to the current condition of the spring is calculated from the definition of the mass spring damper model. The spring constant values are obtained by the parameter optimization method, and the spring damping constant, gamma, is set equal to 500.0. This value has been obtained by trial and error for a stable realization of the MSD model in the given context.

Secondly, for every vertex, force due to kinetic friction damping (inversely proportional to the current velocity) is calculated. The damping coefficient for this case has been set equal to kd = 800. The mass per vertex is set at a value of unity.

Thirdly, elastic force acting on each vertex is calculated based on the change in volume in the current state from the original volume and the shift in the position of current barycenter from the original one. The new volume is determined and the difference in volume from the original volume is calculated. The current barycenter or center of mass for the entire organ is also calculated. Then, on each vertex, a force of magnitude m_kv * vol_diff acts in the direction opposite to the direction of a vector from vertex to the current barycenter, where m_kv is a constant, whose value has been set equal to 100.0 (again chosen by a trial and error procedure for a stable realization of the current model and the given context), and vol_diff is set equal to the value of the volume difference.

After determining the total force acting on each vertex at the current time instant, by the above procedures, the current velocity and position for every vertex is updated. The velocity for the i^{th} vertex is updated according to the Euler method as follows:

$$velocity_{i\_new} = velocity_{i\_old} + force_i \bullet delta\_time / mass \quad \text{.... (4.4)}$$

where, the delta_time is the time constant used for the Euler method of numeric integration.

Since the Euler method tends to become unstable because of the crude linear approximation for future value estimation, we have used a very small value for the time constant delta_time = 0.3ms. We have also restricted the maximum allowable magnitude of velocity to 20 by normalizing the resultant velocity accordingly. This is to ensure stability of the model so that the floating point errors and errors which build up due to the numerical approximation of integration do not cause the model to become unstable.

After the velocity has been updated, the current position is updated for each vertex again in accordance with the Euler method as follows:

$$position_{i\_new} = position_{i\_old} + velocity_{i\_new} \bullet delta\_time \quad \text{.... (4.5)}$$

As can be seen from Eq. (4.5) the new estimated velocity is used in the integration, in accordance with the Implicit Solution method of Differential Equations [24]. The position, velocity and force for each vertex are represented as 3D vectors, having x, y and z fields.

The values of different parameters that we have used in our simulation have been listed in Table 4.1. These values have been either obtained algorithmically or after extensive trial-error experimentation.

Table 4.1 Values of Parameters Used in Simulation

Parameters for Stable Simulation	**Values (Obtained Algorithmically or on a trial-error basis)**
Mass per Vertex	1.0
Spring Constant (Hooke's) ks	100.0
Spring Damping Constant γ	500.0
Per Vertex Motion Friction Damping Constant kd	800.0
Constant for Elastic Force Calculation m_kv	100.0
Value of Time Constant for Numeric Integration δt	0.3 ms
Updating rate of v and r	5ms
Screen Refresh rate (rate of rendering)	20ms

4.3 Results for Mesh Simplification Algorithm

Our algorithm has been tested on both regular geometric figures as well as irregular ones. We have compared the quality of resultant tetrahedral mesh as opposed to the quality of tetrahedra simplified by TetFusion for same number of tetrahedral reduction.

In Figure 4.2 and 4.3, we present the results obtained by the application of our algorithm on an irregular model (kidney mesh referred to earlier) having a large number of primitives, since the results obtained for such cases are more relevant than those obtained for meshes with small number of elements. After running our simplification algorithm as well as Tetfusion [1] on the given dataset or mesh, we generated new meshes with reduced number of elements as obtained in either case. Then we did a quality check on these resultant new meshes, that is, we determined the quality factor for every tetrahedron in each of the two new meshes, according to the definition of quality given in [3]. We plotted the histogram of the tetrahedra against the valid range of quality factor [0, 1.0], using the built-in MATLAB function hist(). The horizontal axis contains the range of allowed quality factor values, divided into 10 sections, and the vertical axis lists the number of tetrahedra belonging to each of these 10 buckets or sections of quality factor values. Figures 4.2 and 4.3 show that the quality factor of tetrahedra resulting from the application of our algorithm tends to be higher than that obtained for sequential tetrahedral fusion. The percentage of tetrahedra having quality factors higher than 0.6 is much higher in our algorithm than with the sequential approach. (It is desirable that the quality factor should be close to 1 for most tetrahedra in the mesh). Using Tetfusion alone results in 45.71% of tetrahedra having quality factor higher than 0.5, while using our approach results in 63.33% of such tetrahedra. Thus our proposed systematic algorithm clearly results in a better quality of mesh for the same degree of simplification. We have also presented a wire frame rendering of the simplified tetrahedral mesh structure (Figure 4.5) along with a wire frame rendering of the original mesh for comparison (Figure 4.4). It can be seen from Figure 4.4 and 4.5 4.6 that the

overall contour and mesh structure of the resultant simplified mesh after the application of our algorithm are well preserved compared to the original mesh.

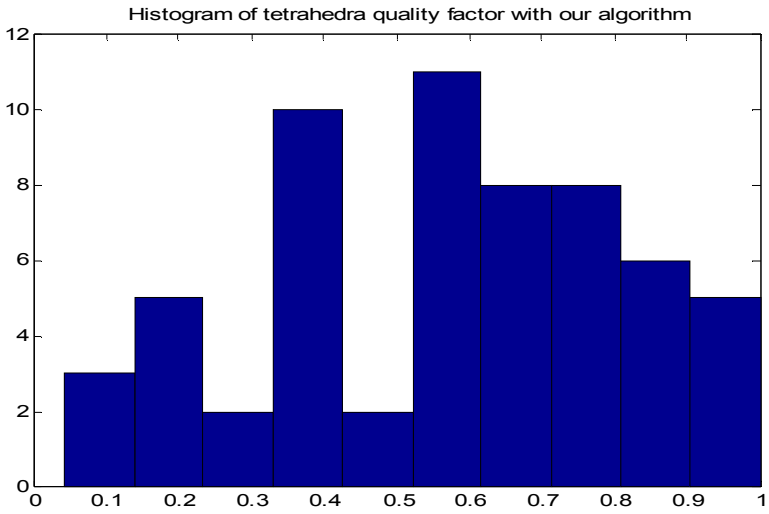


Figure 4.2 Histogram with our algorithm: Percentage of tetrahedra having a quality factor more than 0.5 is 63.33%

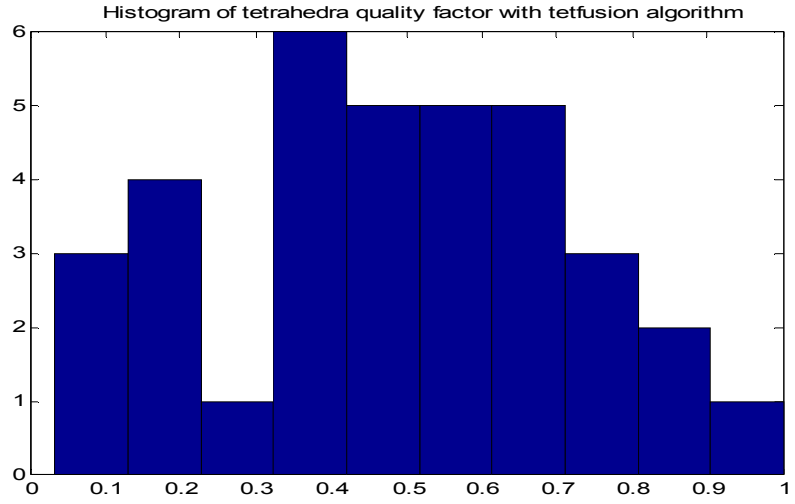


Figure 4.3: Histogram with sequential tetrahedra deletion. Percentage of tetrahedra having a quality factor more than 0.5 is 45.71%.

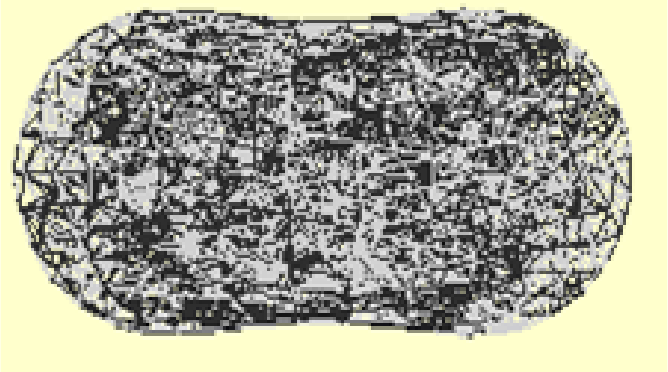


Figure 4.4 Original mesh of Kidney 3D model

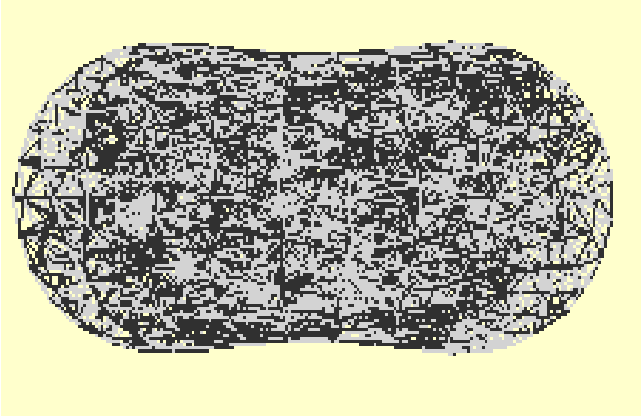


Figure 4.5 Mesh of the Kidney 3D model Simplified by Proposed Algorithm

4.3.1 Conclusion and Future Work for Mesh Simplification

We present a robust and systematic algorithm which can be applied for tetrahedral mesh simplification and quality improvement. It is quite general in its approach towards quality measurement and quite rapid and efficient in its implementation. The approach is very systematic and hence overcomes some of the shortcomings of previous approaches. It tries to achieve an optimal solution and works for both structured and unstructured meshes.

Our algorithm tries to approximate the most optimal solution, but the solution obtained is not the absolute globally optimum one. Future studies can be directed at further improving the optimality with minimal computation overhead. Also our algorithm does not simplify the boundary tetrahedra, where other primitive and time consuming operations like edge collapse or

vertex unification may be necessary in order to reduce the number of elements. Future work can be directed at this direction as well.

4.4 Results for Simulation of Deformation using Parameter Optimized MSD model

The deformations are realized by arbitrarily displacing some vertices from their original position in the tetrahedral mesh mass-spring-damper model. The force application (or vertex displacements) has been done in the form of palpation by instrument to simulate something close to a virtual medical environment. We have presented the screen shots of the organ before and after deformation, and at various stages of intermediate deformations till it returns to the original stable state below. As can be seen from Figures 4.6-4.14, we have been successful in determining the parameter optimized spring constants (following the approach in [1]) and the other necessary parameters for mass spring damper model correctly enough to give a visually realistic and stable deformation response.

A video of the entire deformation process has also been created and some of its screen shots have been provided in Figures 4.15-4.17. The video has been generated using the classes CGLMovie and CAviFile obtained from [64]. The video is filmed at 10 fps and has not been compressed with any codec.

The results as obtained in the simulation environment have been presented below:

Figure 4.6 Undeformed Organ

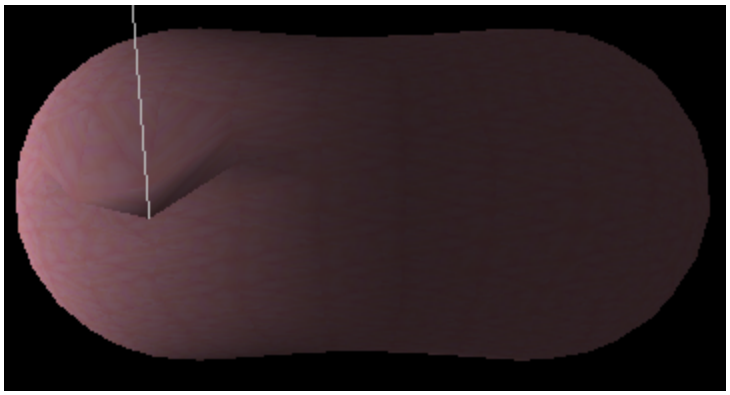


Figure 4.7 After Displacing Vertices (application of force)

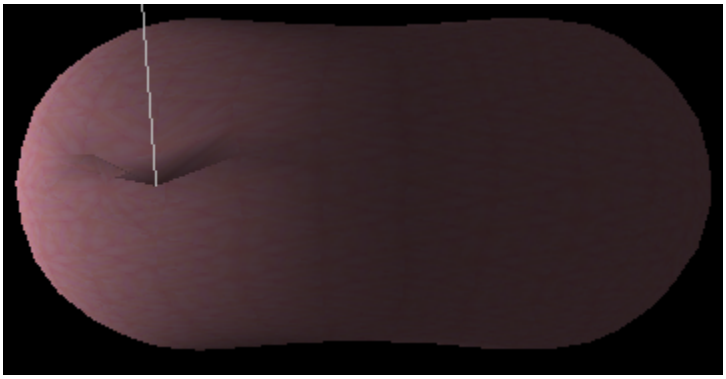


Figure 4.8 Response to Deformation Applied

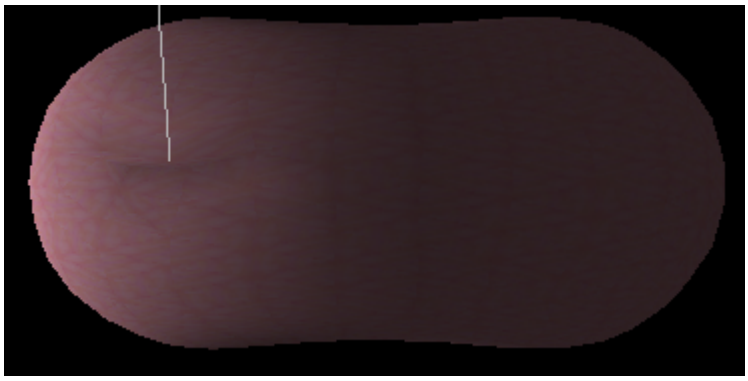


Figure 4.9 Regaining Original State

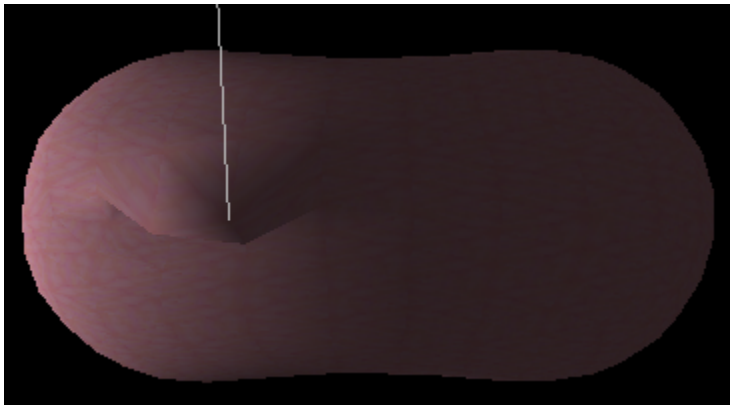


Figure 4.10 Second Deformation Applied

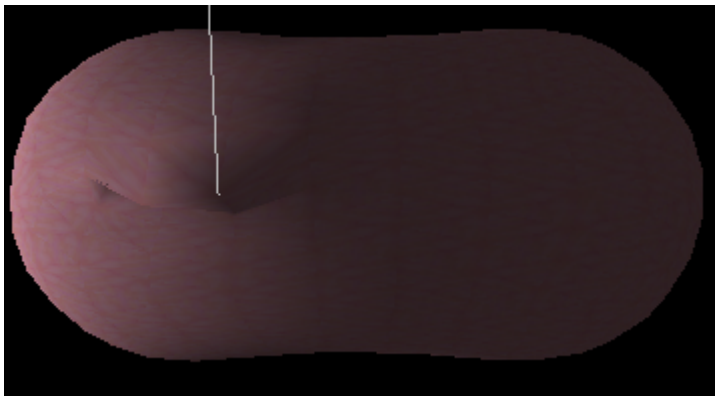


Fig 4.11 Response to Second Deformation Applied

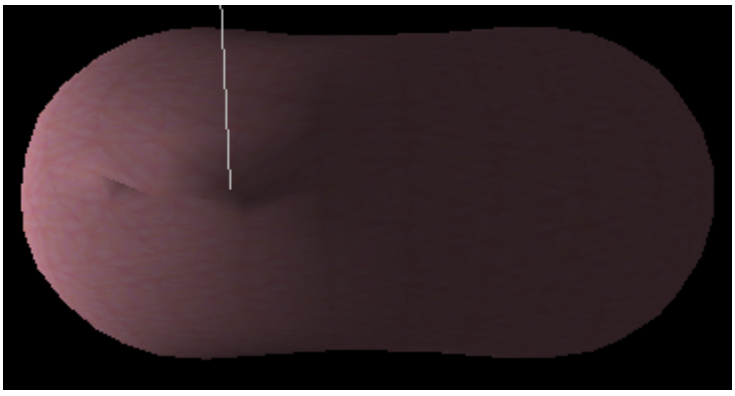


Figure 4.12 Response (contd.)

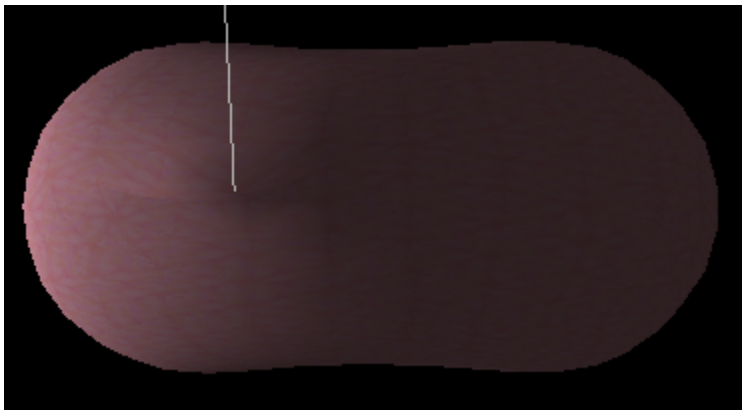


Figure 4.13 Coming Back to Original Form

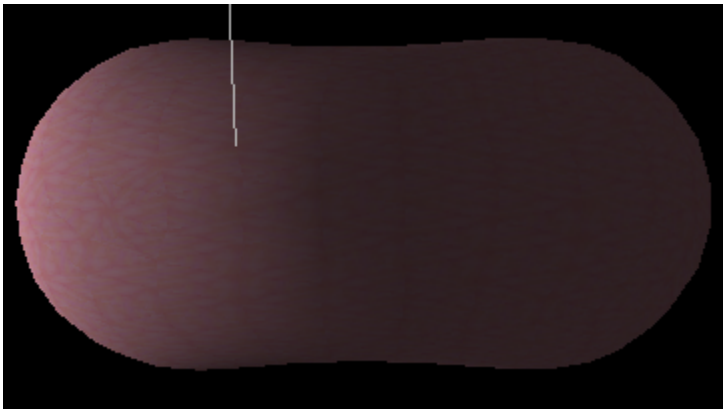


Figure 4.14 Back to Original Undeformed Condition

Screen shots of the video created from the simulation:

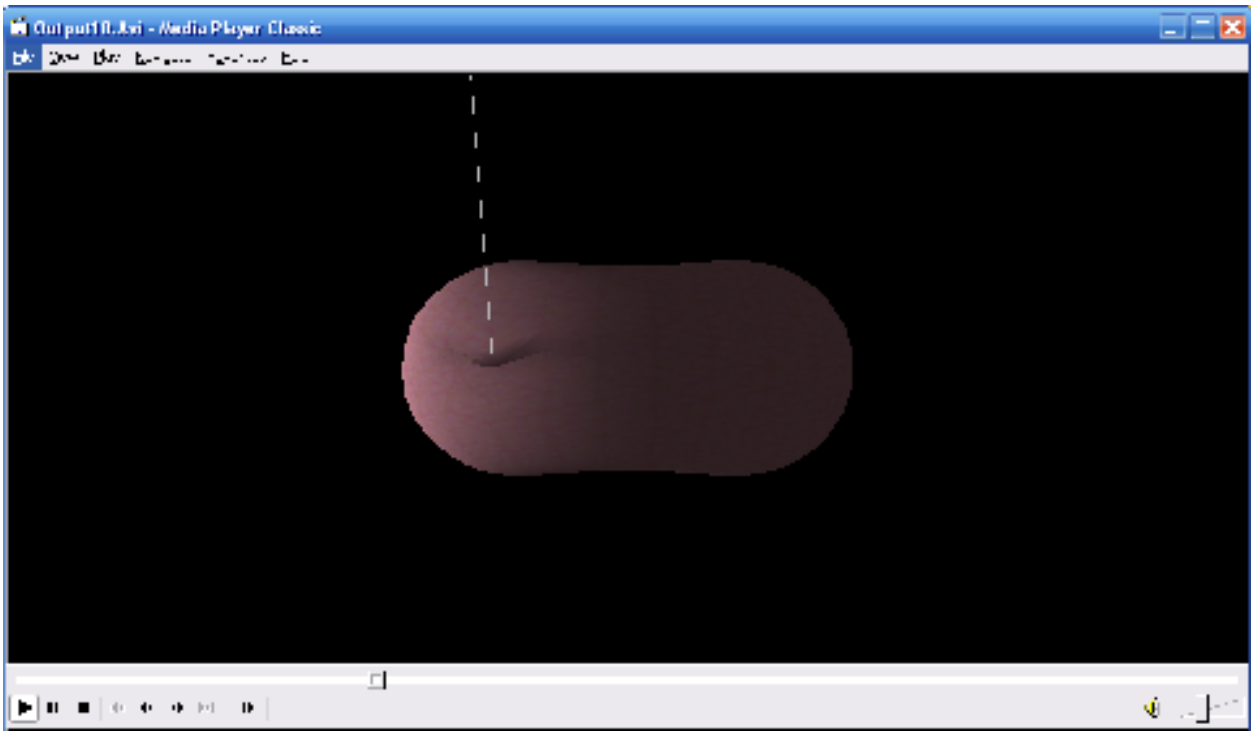


Figure 4.15 Screenshot 1

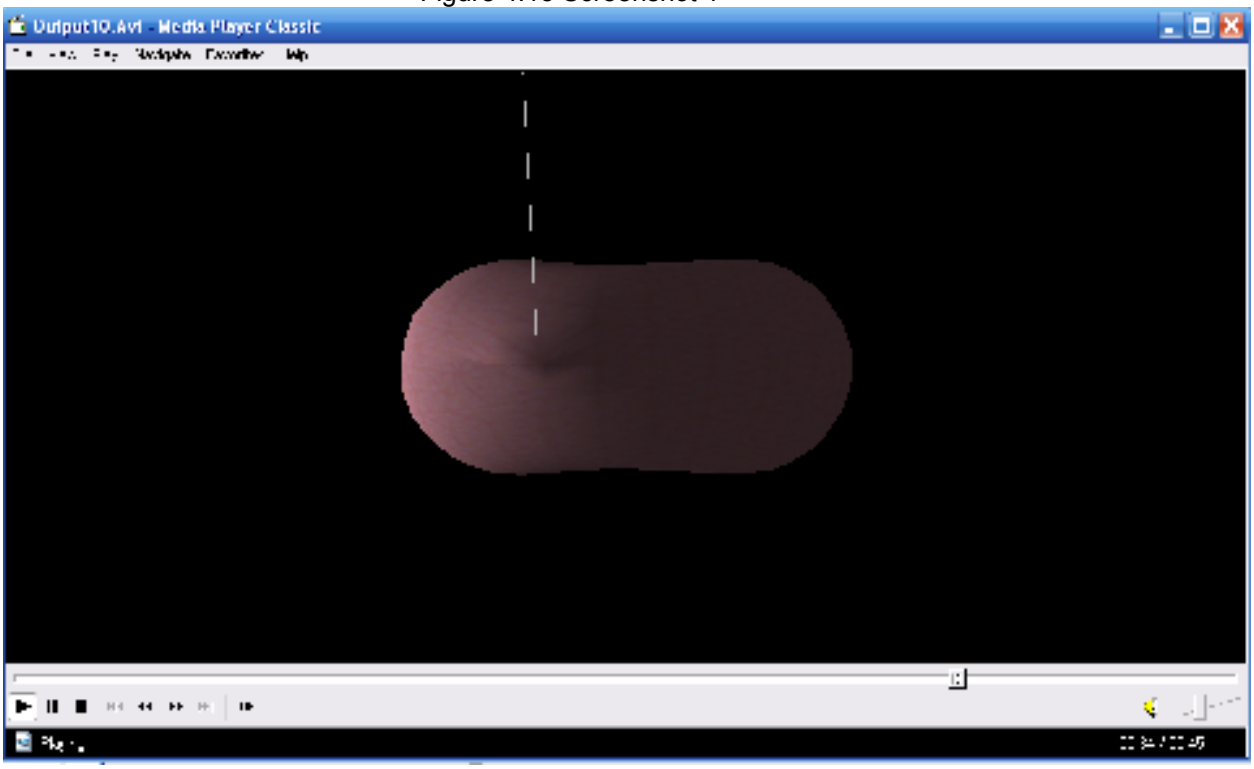


Figure 4.18 Screenshot 2

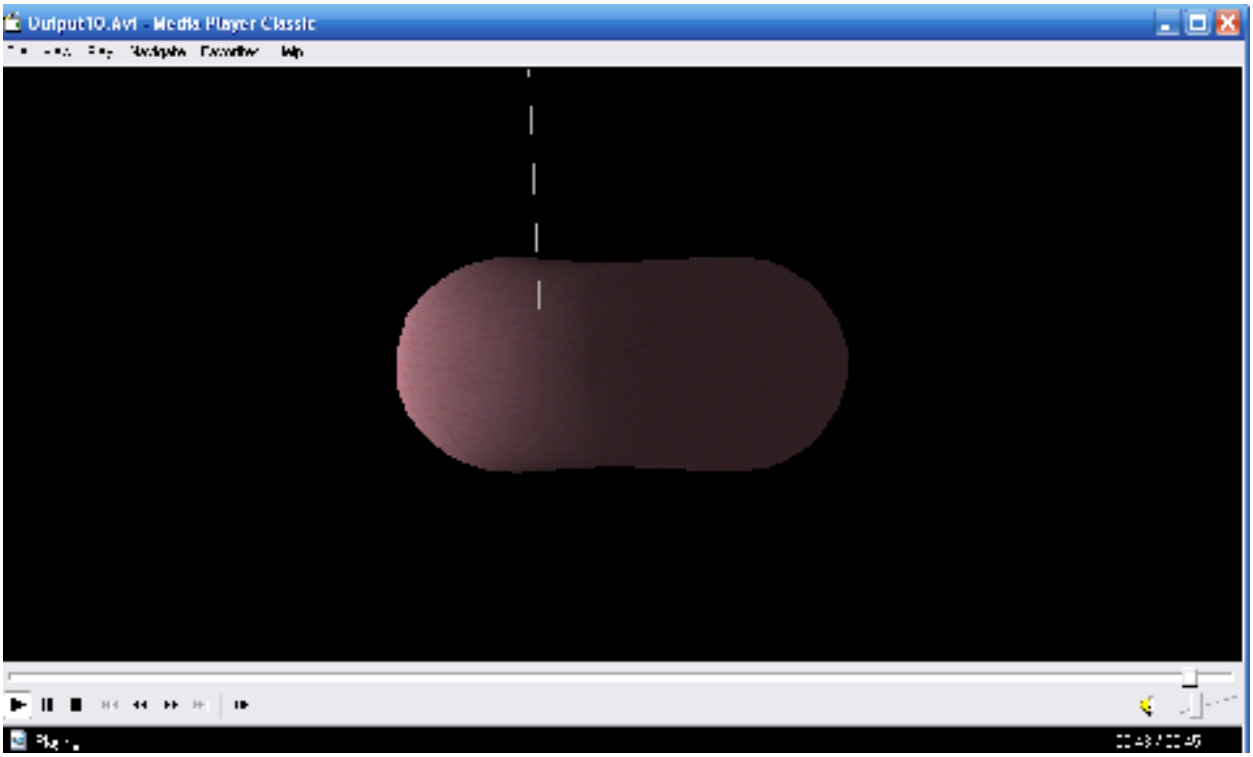


Figure 4.17 Screenshot 3

4.4.1 Conclusion and Future Work on Parameter Optimization

This work can be further exploited by using the optimized parameters in an haptic simulation such as the one at the Virtual Environment Lab at UTA. The elastic and mechanical properties of the human organs and tissues will need to be considered in order to calculate a realistic force feedback for haptic purposes from the virtual organ on deformation.

APPENDIX A

TRIANGULAR AND TETRAHEDRAL MESH GENERATION

Triangulation of a Point Set

Many tetrahedral mesh generators such as advancing front method [18], require a surface mesh of triangles to be generated first. Furthermore, providing a refined, coarsened, or smoothened surface triangulation will prove efficient prior to the generation of the volume mesh [17]. The most common method of triangulation used is Delaunay triangulation. The three-dimensional Delaunay triangulation is a special type of tetrahedralization. For a given point set with n points, the number of triangles in any triangulation grows with O(n), but the number of Delaunay tetrahedra in a tetrahedralization can grow with $O(n^2)$ [17]. Thus, generally tetrahedral mesh for the same solid will have many more elements than a triangle surface mesh for the same solid.

Delaunay Triangulation

Delaunay triangulation for a set P of points in the plane is a triangulation DT(P) such that no point in P is inside the circumcircle of any triangle in DT(P). Figure A.1 illustrates the Delaunay triangulation of a given point set. Delaunay triangulations maximize the minimum angle of all the angles of the triangles in the triangulation, and hence try to avoid generation of "sliver" triangles. The triangulation was invented by Boris Delaunay in 1934.

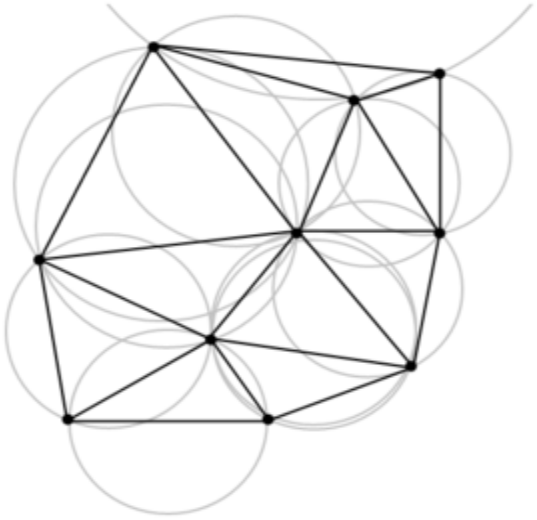


Figure A.1 An example of Delaunay triangulation in the plane with circumcircles shown

Based on Delaunay's definition, the circumcircle of a triangle formed by three points from the original point set is empty if it does not contain vertices other than the three that define it (other points are permitted only on the very perimeter, not inside). The Delaunay condition for bi-dimensional spaces states that a triangle net is a Delaunay triangulation if the circumcircles of all the triangles in the net are empty. It is possible to use it in tridimensional spaces by using a circumscribed sphere in place of the circumcircle.

Generalizations are possible to metrics other than Euclidean. However, in these cases a Delaunay triangulation is not guaranteed to exist or be unique.

The Delaunay triangulation of a discrete point set P corresponds to the dual graph of the Voronoi tessellation for P.

N – Dimensional Delaunay

For a set P of points in the (N-dimensional) Euclidean space, a Delaunay triangulation is a triangulation DT(P) such that no point in P is inside the circum-hypersphere of any simplex in DT(P).

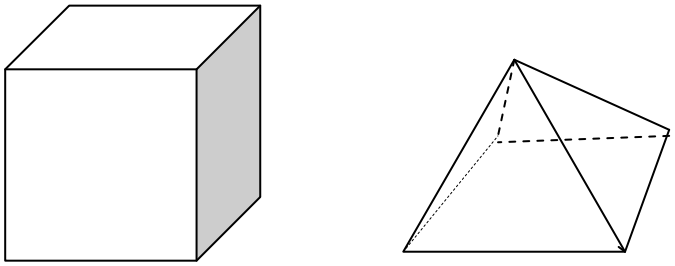
It is known that there exists a unique Delaunay triangulation for P, if P is a set of points in general position; that is, no three points are on the same line and no four are on the same circle, for a two dimensional set of points, or no (n + 1) points are on the same hyperplane and no (n + 2) points are on the same hypersphere, for an n-dimensional set of points.

Tetrahedralization of Point Set

A polyhedron is a solid that is bounded by a set of polygons. The b-rep of a simple polyhedron satisfies Euler's formula [18, 19, 20]:

$$V - E + F = S \qquad \text{.....(A.1)}$$

where V is the number of vertices, E is the number of edges, F is the number of faces and S refers to the number of solids including infinite 'solid' outside the polyhedron. For the case of a single polyhedron, *S = 2.* Figure A.2 illustrates the application of Eq. (A.1) for some common solids.



V = 8
E = 12
F = 6

V = 5
E = 8
F = 5

Figure A.2 Euler's formula illustrated for some simple polyhedra

Euler's formula gives a necessary but not sufficient condition for an object to be a simple polyhedron. In order to guarantee that the object is solid additional constraints need to be satisfied: each edge must connect two vertices and must be shared by exactly two faces, at least three edges must meet at a vertex and faces must not interpenetrate [19].

For a general tetrahedralization, Euler's formula can be written as

$$V - E + t - T = 1 \qquad \text{.....(A.2)}$$

where *t* is the number of triangles and *T* is the number of tetrahedra. Based on this equation simple bounds on the number of tetrahedra can be deduced [17]

$$V - 3 \le T \le \binom{V-1}{2} - n_{hull} + 2 \qquad \text{.....(A.3)}$$

where n_{hull} is the number of points on the convex hull.

APPENDIX B

LIST OF CONSTRAINTS OPTIMIZED FOR PARAMETER OPTIMIZATION [1]

$$\sum_{i=1}^{N_m} k_{mi}\, U_{mi}^{2}\, \left[69 + 251\lambda_1 + (4-68\lambda_1)\cos 2\alpha_{mi} + 9(\lambda_1 - 1)\cos 4\alpha_{mi} = 0\right] \quad \text{(B.1)}$$

$$\sum_{i=1}^{N_m} k_{mi} U_{mi}^{2} \left[(20\lambda_1 - 4)\cos 2\alpha_{mi} \cos 2\beta_{mi} + (1 - 17\lambda_1)\cos 2\beta_{mi} + 3(1-\lambda_1)\cos 4\alpha_{mi} \cos 2\beta_{mi} = 0\right] \quad \text{(B.2)}$$

$$\sum_{i=1}^{N_m} k_{mi} U_{mi}^{2} \left[(20\lambda_1 - 4)\sin 2\alpha_{mi} \cos 2\beta_{mi} + (1 - 17\lambda_1)\cos 2\beta_{mi} + 3(1-\lambda_1)\cos 4\alpha_{mi} \sin 2\beta_{mi} = 0\right] \quad \text{(B.3)}$$

$$\sum_{i=1}^{N_m} k_{mi} U_{mi}^{2} \left[(1 - 17\lambda_1) + (4 - 52\lambda)\cos 2\alpha_{mi} + 5(1-\lambda_1)\cos 4\alpha_{mi} = 0\right] \quad \text{(B.4)}$$

$$\sum_{i=1}^{N_m} k_{mi} U_{mi}^{2} \left[(18\lambda_1 - 2)\sin 2\alpha_{mi} \sin\beta_{mi} + (1-\lambda_1)\sin 4\alpha_{mi} \sin 2\beta_{mi} = 0\right] \quad \text{(B.5)}$$

$$\sum_{i=1}^{N_m} k_{mi} U_{mi}^{2} \left[(18\lambda_1 - 2)\sin 2\alpha_{mi} \cos\beta_{mi} + (1-\lambda_1)\sin 4\alpha_{mi} \cos 2\beta_{mi} = 0\right] \quad \text{(B.6)}$$

$$\sum_{i=1}^{N_m} k_{mi} U_{mi}^{2} \left[(5\lambda_1 - 1)\cos 2\beta_{mi} - 4\lambda_1 \cos 2\alpha_{mi} \cos 2\beta_{mi} + (1-\lambda_1)\cos 4\alpha_{mi} \cos 2\beta_{mi} = 0\right] \quad \text{(B.7)}$$

$$\sum_{i=1}^{N_m} k_{mi} U_{mi}^{2} \left[(5\lambda_1 - 1)\sin 2\beta_{mi} - 4\lambda_1 \cos 2\alpha_{mi} \sin 2\beta_{mi} + (1-\lambda_1)\cos 4\alpha_{mi} \sin 2\beta_{mi} = 0\right] \quad \text{(B.8)}$$

$$\sum_{i=1}^{N_m} k_{mi} U_{mi}^{2} \left[9\cos 4\beta_{mi} - 12\cos 2\alpha_{mi} \cos 4\beta_{mi} + 3\cos 4\alpha_{mi} \cos 4\beta_{mi} = 0\right] \quad \text{(B.9)}$$

$$\sum_{i=1}^{N_m} k_{mi} U_{mi}^{2} \left[9\sin 4\beta_{mi} - 12\cos 2\alpha_{mi} \sin 4\beta_{mi} + 3\cos 4\alpha_{mi} \sin 4\beta_{mi} = 0\right] \quad \text{(B.10)}$$

$$\sum_{i=1}^{N_m} k_{mi} U_{mi}^{2} \left[2\sin 2\alpha_{mi} \sin 3\beta_{mi} - \sin 4\alpha_{mi} \sin 4\beta_{mi} = 0\right] \quad \text{(B.11)}$$

$$\sum_{i=1}^{N_m} k_{mi} U_{mi}^{2} \left[2\sin 2\alpha_{mi} \cos 3\beta_{mi} - \sin 4\alpha_{mi} \cos 4\beta_{mi} = 0\right] \quad \text{(B.12)}$$

$$\sum_{i=1}^{N_m} k_{mi} U_{mi}^{2} \left[35\cos 4\alpha_{mi} + 20\cos 2\alpha_{mi} + 9 = 0\right] \quad \text{(B.13)}$$

$$\sum_{i=1}^{N_m} k_{mi} U_{mi}^{2} \left[2\sin 2\alpha_{mi} \sin\beta_{mi} - 7\sin 4\alpha_{mi} \sin\beta_{mi} = 0\right] \quad \text{(B.14)}$$

$$\sum_{i=1}^{N_m} k_{mi} U_{mi}^{2} \left[2\sin 2\alpha_{mi} \cos\beta_{mi} - 7\sin 4\alpha_{mi} \cos\beta_{mi} = 0\right] \quad \text{(B.15)}$$

$$\sum_{i=1}^{N_m} k_{mi} U_{mi}^2 \left[3\cos 2\beta_{mi} + 4\cos 2\alpha_{mi} \cos 2\beta_{mi} - 7\cos 4\alpha_{mi} \cos 2\beta_{mi} = 0\right] \quad \text{(B.16)}$$

$$\sum_{i=1}^{N_m} k_{mi} U_{mi}^2 \left[3\sin 2\beta_{mi} + 4\cos 2\alpha_{mi} \sin 2\beta_{mi} - 7\cos 4\alpha_{mi} \sin 2\beta_{mi} = 0\right] \quad \text{(B.17)}$$

Substituting $\lambda_1 = -1/4$, the equations (B.1, B.4 – B.8) become:

$$\sum_{i=1}^{N_m} k_{mi} U_{mi}^2 \left[25 + 84\cos 2\alpha_{mi} - 45\cos 4\alpha_{mi} = 0\right] \quad \text{(B.18)}$$

$$\sum_{i=1}^{N_m} k_{mi} U_{mi}^2 \left[21 + 68\cos 2\alpha_{mi} - 25\cos 4\alpha_{mi} = 0\right] \quad \text{(B.19)}$$

$$\sum_{i=1}^{N_m} k_{mi} U_{mi}^2 \left[5\sin 4\alpha_{mi} \sin 2\beta_{mi} - 26\sin 2\alpha_{mi} \sin \beta_{mi} = 0\right] \quad \text{(B.20)}$$

$$\sum_{i=1}^{N_m} k_{mi} U_{mi}^2 \left[5\sin 4\alpha_{mi} \cos 2\beta_{mi} - 26\sin 2\alpha_{mi} \cos \beta_{mi} = 0\right] \quad \text{(B.21)}$$

$$\sum_{i=1}^{N_m} k_{mi} U_{mi}^2 \left[9\cos 2\beta_{mi} - 4\cos 2\alpha_{mi} \cos 2\beta_{mi} - 5\cos 4\alpha_{mi} \cos 2\beta_{mi} = 0\right] \quad \text{(B.22)}$$

$$\sum_{i=1}^{N_m} k_{mi} U_{mi}^2 \left[9\sin 2\beta_{mi} - 4\cos 2\alpha_{mi} \sin 2\beta_{mi} - 5\cos 4\alpha_{mi} \sin 2\beta_{mi} = 0\right] \quad \text{(B.23)}$$

APPENDIX C

EULER'S METHOD OF NUMERICAL INTEGRATION

Euler's Method [23]

In calculating numerical solutions discrete time steps are taken starting with the initial value $x(t_0)$. An approximate change in x, Δx, over a time interval Δt is calculated from using the derivative function f (or $\dot{x}$) and then x is incremented by Δx to obtain the new value.

In a numerical integral solution method, one or more derivative evaluations are performed at each time step. The simplest numerical method is called Euler's method. Let the initial value for x be denoted by $x_0 = x(t_0)$ and the estimate of x at a later time $t_0 + h$ by $x(t_0 + h)$ where h is a stepsize parameter. Euler's method evaluates $x(t_0 + h)$ by the following formula:

$$x(t_0 + h) = x_0 + h\dot{x}(t_0) \quad \text{.... (C.1)}$$

Thus Euler's method will give accurate solution for functions having a constant derivative. However, for any arbitrary function, with variable derivative it is not very accurate. Figure C.1 shows the example of a 2D function f whose integral curves are concentric circles for which Euler's method fails. Moreover, Euler's method can be unstable. For sufficiently small step sizes it gives reasonable behavior. Assuming the function $x(t)$ is smooth, it can be represented by Taylor's series,

$$x(t_0 + h) = x_0 + h\dot{x}(t_0) + \frac{h^2}{2!}\ddot{x}(t_0) + \frac{h^3}{3!}\dddot{x}(t_0) + \ldots + \frac{h^n}{n!}\frac{\partial^n x}{\partial t^n} + \cdots \quad \text{.... (C.2)}$$

Thus the error is $O(h^2)$.

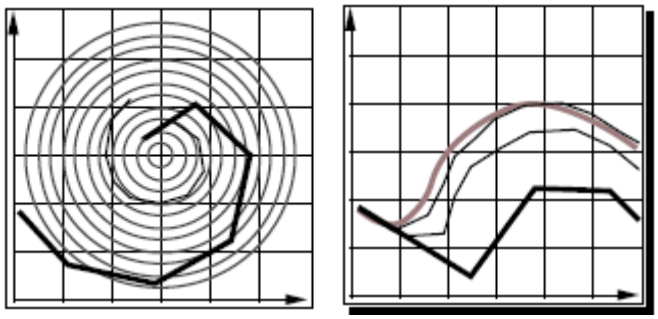


Figure C.1 Pictorial Representation of Error in Euler's Method [23]

REFERENCES

[1] Xiuzhong Wang and Venkat Devarajan, “Parameter Optimization for 3D Mass Spring Damper Models”, *Proceedings of Medicine Meets Virtual Reality 16 – parallel, combinatorial, convergent: NextMed by Design,* Vol. 132, pp 544-549, 2008

[2] Prashant Chopra and Joerg Meyer, “TetFusion: An Algorithm for Rapid Tetrahedral Mesh Simplification”, *Proceedings of IEEE Visualization 2002 – VIS’02,* pp 133-140, 2002.

[3] B. Cutler, J. Dorsey and L. Macmillan, “Simplification and Improvement of Tetrahedral Models for Simulation”, *Proceedings of the 2004 Eurographics/ACMSIGGRAPH symposium on Geometry Processing,* pp 93-102, 2004.

[4] P. Cignoni, D. Constanza, C. Montani, C. Rochhini and R. Scopigno, “Simplification of Tetrahedral Meshes with Accurate Error Evaluation”, *Proceedings of IEEE Visualization 2000 (VIS 2000) – VISUALIZATION ’00),* pp 85-92, 2000.

[5] P.Cignoni, C. Montani and R. Scopigno, “A Comparison of Mesh Simplification Algorithms”, *Computers and Graphics,* Vol. 22, Issue 1, pp 37-54, 1998.

[6] Hugues Hoppe, “Progressive Meshes”, *Proceedings of SIGGRAPH’96 (New Orleans, Louisiana, August 1996), ACM SIGGRAPH, ACM Press,* pp 99-108, August 1996.

[7] O.G. Staadt and M.H. Gross, “Progressive Tetrahedralizations”, *Proceedings of IEEE Visualization 1998 – VIS’98,* pp 397-402, October 1998.

[8] Alan D. Kalvin and Russell H. Taylor, “Superfaces: Polygonal Mesh Simplification with Bounded Error”, *IEEE Computer Graphics and Applications,* Vol 16, Issue 3, pp 64-77, May 1996.

[9] William J. Schroeder, Jonathan A. Zarge and William E. Larsen, “Decimation of Triangle Meshes”, *Computer Graphics,* Vol 26, Issue 2, pp 65-70, 1992.

[10] Greg Turk, “Re-tiling Polygonal Surfaces”, *Computer Graphics,* Vol 26, Issue 2, pp 55-64, 1992.

[11] Y.-J. Chiang and X. Lu, “Progressive Simplification of Tetrahedral Meshes Preserving All Isosurface Topologies”, *Computer Graphics Forum,* Vol. 22, Issue 3, pp 493-504, 2003

[12] Isaac J. Trotts, Bernd Hamann, and Kenneth I. Joy, “Simplification of Tetrahedral Meshes”, *Proceedings of IEEE Visualization’* 98 (October 1998), pp 287-296. 1998.

[13] Isaac J. Trotts, Bernd Hamann, and Kenneth I. Joy, “Simplification Of Tetrahedral Meshes With Error Bounds”, *IEEE Transactions on Visualization and Computer Graphics,* Vol. 5, Issue 3, pp 224- 237, 1999.

[14] Vijay Natarajan and Herbert Edelsbrunner, “Simplification of Three Dimensional Density Maps”, *IEEE Transactions on Visualizations and Computer Graphics,* Vol. 10, Issue 5, pp 587-597, September/October 2004.

[15] William J. Schroeder, “A Topology Modifying Progressive Decimation Algorithm”, *Proceedings of IEEE Visualization’ 97,* pp 205-212, 1997.

[16] M. Garland, “Multi-resolution modeling: Survey & Future Opportunities”, *EUROGRAPHICS’ 99, State of the Art Report (STAR) (Aire-la-Ville, CH, 1999),* pp 111-131, Eurographics Association,1999.

[17] Peter Fleischmann, “Mesh Generation for Technology CAD in Three Dimensions”, *PhD Dissertation, Institute for Microelctronics (IμE),* 2000.

[18] P.J. Frey, H. Borouchaki and P.L. George, “Delaunay Tetrahedralization Using an Advancing-Front Approach”, *5th International Meshing Roundtable (IMRT'96)*, pp 31-46, October 1996.

[19] James D. Foley, Andries van Dam, Steven K Feiner, John F. Hughes and Richard L. Phillips, “Introduction to Computer Graphics”, *Addison-Wesley Publishing Company*, August 1995.

[20] Joesph O’Rourke, “Computational Geometry In C”, *Cambridge University Press,* 1998.

[21] Alan Watt and Fabio Policarpo, “3D games: Real Time Rendering and Software Technology”, *Copyright ACM Press,* 2001.

[22] Jitesh Butala, “Collision Response for Virtual Laparoscopic Surgery”, *Master’s Thesis, University of Texas at Arlington*, August 2005.

[23] Andrew Witkin and David Baraff, “Physically Based Modeling: Differential Equation Basics”, *SIGGRAPH 2001 Course Notes: Physically Based Modeling,* 2001.

[24] David Baraff, “Physically Based Modeling: Implicit Methods for Differential Equations”, *SIGGRAPH 2001 Course Notes: Physically Based Modeling,* 2001.

[25] Xiuzhong Wang, “Deformable Haptic Models for Surgical Simulation”, *PhD Dissertation, University of Texas at Arlington,* August 2005.

[26] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer, “Elastically Deformable Models,” *ACM Computer Graphics, SIGGRAPH’87,* Vol. 21, Issue 4, pp. 205–214, 1987.

[27] Evgeny Gladilin, “Biomechanical Modeling of Soft Tissue and Facial Expressions for Cranofacial Surgery Planning”, *PhD Dissertation, Fachbereich Mathematik u. Informatik, Freie Universität Berlin,* June 2003.

[28] Sarah F. F. Gibson and Brian Mirtich, “A Survey of Deformable Modeling in Computer Graphics”, *Technical Report, MITSUBISHI ELECTRIC RESEARCH LABORATORIES,*1997

[29] C. Basdogan, M. Sedef, M. Harders, S. Wesarg, "VR Based Simulators for Training in Minimally Invasive Surgery", *IEEE Computer Graphics and Applications,* Vol. 27, Issue 2, pp 54-66, March-April 2007.

[30] Xiuzhong Wang and Venkat Devarajan, “1D and 2D Structured Mass Spring Models with Preload”, *The Visual Computer,* Vol 21, Issue 7, pp 429-448, August 2005.

[31] S. De and K. J. Bathe, “The Method of Finite Spheres", *Computational Mechanics*, Vol. 25, Issue 4, pp 329-345, April 2000.

[32] J. Kim, S. De and M.A. Srinivasan, “Computationally Efficient Techniques for Real Time Surgical Simulation with Force Feedback”, *IEEE Virtual Reality Conference,* 2002.

[33] M. Bro-Nielson, “Fast Finite Elements for Surgery Simulation”, *Proceedings of 5th Medicine*

Meets Virtual Reality, 1997.

[34] B. A. Szabo, “Finite Element Analysis”, *New York : Wiley*, 1991.

[35] H. Delingette, S. Cotin, and N. Ayache, “A Hybrid Elastic Model Allowing Real-time Cutting, Deformations and Force-feedback for Surgery Training and Simulation”, *The Visual Computer,* Vol. 16, Issue 8, December 2000.

[36] R. Balaniuk and K. Salisbury, “Dynamic Simulation of Deformable Objects using the Long Element Method," in *Proceedings of the 10th international symposium on Haptic Interfaces for Virtual Environment and Teleoperator systems*, 2002, pp. 58-65.

[37] G. Debunne, A. H. Barr, and M.-P. Cani, “Interactive Multi-resolution Animation of Deformable Models," *Proceedings of Eurographics Workshop on Computer Animation and Simulation*, 1999.

[38] A. V. Gelder, “Approximate Simulation of Elastic Membranes by Triangulated Spring Meshes," *J. Graph. Tools*, Vol. 3, Issue 2, pp. 21-42, 1998.

[39] K. S. Bhat, C. D. Twigg, J. K. Hodgins, P. K. Khosla, Z. Popovic, and S. M. Seitz, “Estimating Cloth Simulation Parameters from Video”, *Proceedings of ACM SIGGRAPH/ Eurographics Symposium on Computer Animation*, pp. 37-51, 2003

[40] A. Maciel, R. Boulic, and D. Thalmann, “Deformable Tissue Parameterized by Properties of Real Biological Tissue”, *Proceedings of International Symposium on Surgery Simulation and Soft Tissue Modeling*, pp. 74-87, 2003.

[41] N. Jojic and T. S. Huang, “Estimating Cloth Draping Parameters from Range Data”, in *Proceedings of International Workshop on Synthetic Natural Hybrid Coding and Three Dimensional Imaging*, pp. 73–76, 1997.

[42] X. Provot, “Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior”, *Proceedings of Graphics Interface*, pp. 147–154, 1995.

[43] T. Vassilev, B. Spanlang, and Y. Chrysanthou, “Fast Cloth Animation on Walking Avatars”, *Comput. Graphics Forum*, Vol. 20, Issue 3, pp. 260–267, 2001.

[44] R. Bridson, S. Marino, and R. Fedkiw, “Simulation of Clothing with Folds and Wrinkles”, *SCA’03: Proceedings of 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, Eurographics Association*, pp. 28–36, 2003.

[45] E. Grinspun, A. N. Hirani, M. Desbrun, and P. Schroder, “Discrete Shells”, *SCA ’03: Proceedings of 2003 ACM SIGGRAPH/ Eurographics Symposium on Computer Animation*. *Eurographics Association*, pp. 62–67, 2003.

[46] S. M. Platt and N. I. Badler, “Animating Facial Expressions”, *Proceedings of SIGGRAPH'81, ACM Press*, pp. 245-252, 1981.

[47] K. Waters, “A Muscle Model for Animating Three-dimensional Facial Expression”, *ACM Computer Graphics*, Vol. 21, Issue 4, pp. 17-24, July 1987.

[48] D. Terzopoulos and K. Waters, “Physically-based Facial Modeling, Analysis and Animation”, *Journal of Visualization and Computer Animation*, Vol. 1, Issue 2, pp 73-80, 1990.

[49] W. T. Reeves, “Particle Systems: A Technique for Modeling a Class of Fuzzy Objects”, *ACM Transactions on Graphics*, Vol. 2, Issue 2, pp. 91-108, 1983.

[50] G. S. P. Miller, “The Motion Dynamics of Snakes and Worms”, *Proceedings of SIGGRAPH'88*, *ACM Press*, pp 169-173, 1988.

[51] X. Tu and D. Terzopoulos, “Artificial Fishes: Physics, Locomotion, Perception, Behavior”, *Proceedings of SIGGRAPH'94*, *ACM Press*, pp 43-50, 1994.

[52] D. E. Breen, D. H. House, and M. J. Wozny, “Predicting the Drape of Woven Cloth using Interacting Particles”, *Proceedings of SIGGRAPH'94*, *ACM Press*, pp 365-372, 1994.

[53] B. Eberhardt, A. Weber, and W. Strasser, “A Fast, Flexible, Particle-System Model for Cloth Draping”, *IEEE Computer Graphics and Applications*, Vol. 16, Issue 5, pp. 52-59, Sept 1996.

[54] J. Brown, K. Montgomery, J.-C. Latombe, and M. Stephanides, “A Microsurgery Simulation System”, *Medical Image Computing and Computer-Assisted Interventions*, pp 137-144, 2001.

[55] Y. Shen, V. Devarajan, and R. Eberhart, “Haptic Herniorrhaphy Simulation with Robust and Fast Collision Detection Algorithm”, *Proceedings of Medicine Meets Virtual Reality*, 2005

[56] D. Hutchinson, M. Preston, and T. Hewitt, “Adaptive Refinement for Mass-Spring Simulations”, *Proceedings of the 7th Eurographics workshop on Computer Animation and Simulation '96*, Springer-Verlag New York, Inc., pp 31-45, 1996.

[57] D. Bourguignon and M.-P. Cani, “Controlling Anisotropy in Mass-Spring Systems”, *Proceedings of 11th Eurographics Workshop on Animation and Simulation*, pp 113-123, 2000

[58] D. Baraff and A. Witkin, “Large Steps in Cloth Simulation”, *Proceedings of SIGGRAPH'98*, pp 43-54, 1998.

[59] M. Hauth, O. Etzmuss, and W. Strasser, “Analysis of Numerical Methods for the Simulation of Deformable Models”, *The Visual Computer*, Vol. 19, Issue 7-8, pp 581-600, 2003.

[60] K.-J. Choi and H.-S. Ko, “Stable but Responsive Cloth”, *Proceedings of SIGGRAPH'02*, *ACM Press*, pp 604-611, 2002.

[61] O. Etzmuss, J. Gross, and W. Strasser, “Deriving a Particle System from Continuum Mechanics for the Animation of Deformable Objects”, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 9, Issue 4, pp 538–550, 2003.

[62] www.guez.org/gmsh

[63] Pierre Alliez, “A small VRML viewer using OpenGL and MFC”, *Available online at http://www.codeproject.com/KB/openGL/wrl_viewer.aspx*.

[64] P. Gopalakrishna, “Recording OpenGL and DirectX Rendered Animations”, *Available online at http://www.geocities.com/krishnapg/SimulationRecording.html*

[65] David Luebke, Martin Reddy, Jonathan D. Cohen, Amitabh Varshney, Benjamin Watson, Robert Huebner, “Level of Detail for 3D Graphics”, *Morgan Kaufman Series in Computer Graphics and Geometric Modeling, Boston, MA Elsevier,* 2003

BIOGRAPHICAL INFORMATION

Koyel Mukherje was born on June 30, 1982 to Dr. Kishalay Bikash Mukherjee and Dr. Shyamali Mukherjee in the city of Kolkata, in West Bengal, India. She completed schooling from Calcutta Girls' High School in Kolkata, India. After that, she completed her undergraduate studies from Institute of Engineering and Management, Kolkata, India, and received the Bachelor of Technology (B.Tech) degree in Electronics and Communication Engineering from West Bengal University of Technology in July 2005. She worked in Wipro Technologies, Wipro Ltd, Bangalore, India from October 2005 to July 2006. She then pursued higher studies from the University of Texas at Arlington, from August 2006 onwards from where she received Master of Science (MS) degree in Electrical Engineering in August 2008. During her MS, she was the recipient of the Dean's Masters Fellowship from UT Arlington. She is also a member of Tau Beta Pai. She worked as an intern in Texas Instruments, Inc. from May 2007-August 2007. Her research interests are mostly concentrated on Computer Science, especially Computer Graphics and she plans to pursue further research in the field of Computer Science after the completion of her MS degree. She is passionate about music and loves reading, especially literary classics by authors in English, as well as in her native language Bengali.