EFFICIENT TECHNIQUES FOR SOFT TISSUE MODELING AND SIMULATION

ALPASLAN DUYSAK

A thesis submitted in fulfillment of the requirements of Bournemouth University
for the degree of Doctor of Philosophy

September 2004

ABSTRACT

Performing realistic deformation simulations in real time is a challenging problem in computer graphics. Among numerous proposed methods including Finite Element Modeling and ChainMail, we have implemented a mass spring system because of its acceptable accuracy and speed. Mass spring systems have, however, some drawbacks such as, the determination of simulation coefficients with their iterative nature. Given the correct parameters, mass spring systems can accurately simulate tissue deformations but choosing parameters that capture nonlinear deformation behavior is extremely difficult. Since most of the applications require a large number of elements i.e. points and springs in the modeling process it is extremely difficult to reach real-time performance with an iterative method. We have developed a new parameter identification method based on neural networks. The structure of the mass spring system is modified and neural networks are integrated into this structure. The input space consists of changes in spring lengths and velocities while a "teacher" signal is chosen as the total spring force, which is expressed in terms of positional changes and applied external forces. Neural networks are trained to learn nonlinear tissue characteristics represented by spring stiffness and damping in the mass spring algorithm. The learning algorithm is further enhanced by an adaptive learning rate, developed particularly for mass spring systems.

In order to avoid the iterative approach in deformation simulations we have developed a new deformation algorithm. This algorithm defines the relationships between points and springs and specifies a set of rules on spring movements and deformations. These rules result in a deformation surface, which is called the search space. The deformation algorithm then finds the deformed points and springs in the search space with the help of the defined rules. The algorithm also sets rules on each element i.e. triangle or tetrahedron so that they do not pass through each other. The new algorithm is considerably faster than the original mass spring systems algorithm and provides an opportunity for various deformation applications.

We have used mass spring systems and the developed method in the simulation of craniofacial surgery. For this purpose, a patient-specific head model was generated from MRI medical data by applying medical image processing tools such as, filtering, the segmentation and polygonal representation of such model is obtained using a surface generation algorithm. Prism volume elements are generated between the skin and bone surfaces so that different tissue layers are included to the head model. Both methods produce plausible results verified by surgeons.

PUBLICATIONS

- Duysak A, Zhang J. J., Ilankovan V., 2003, Efficient Modeling and Simulation of Soft Tissue Deformation Using Mass-Spring Systems, *The 17th International Congress and Exhibition on Computer Assisted Radiology and Surgery (CARS 2003, London),* pp. 337-342.

- Duysak A, Zhang J. J., 2003, Identification of Simulation Parameters Using Neural Networks, *The 6th International Conference on Computer Graphics and Artificial Intelligence (3IA 2003, France)*

- Duysak A, Zhang J. J., 2004, Fast Simulation of Deformable Objects, *International symposium on Computer Animation, The 8th International Conference on Information Visualization, IEEE Computer Society, (IV 2004, London),* pp.422-427.

List of Contents

List of Tables

List of Figures

AKNOWLEDMENTS

I would like to acknowledge my supervisors Prof. Jian J. Zhang and Prof. Peter Comninos for their advice, guidance and support. I would like to acknowledge Ari Sarafopoulos for his friendship and help. Craig Senior is also thanked for his assistance with all types of software and hardware problems.

I am grateful to all my friends: Serdar, Hasan, Mehmet, Abdurrahman (Apo), Mudassar and Russell.

I would like to thank my family members, especially my wife Zuleyha and my sons Omer and Girayhan. I wish that I could spend more time with them, I am deeply sorry for that. I finally like to remember my parents and like to dedicate this work to for their memories.

CHAPTER 1

INTRODUCTION

Computer graphics and its applications in the form of animation or simulation have entered our everyday life in many forms mainly in entertainment, the recreation or prediction of real world phenomena and training. Movies are partly or fully realized in a computer environment and similarly video games have become increasingly popular thanks to their image quality and realistic implementation. Besides the entertainment industry, computer animation and simulations are indispensable tools in science. We can now accurately simulate galaxies based on the governing physics and simulate molecules and elements with the correct chemical behavior. Nowadays engineers are able to design electrical or mechanical circuits and simulate their real world responses before going through to the implementation phase. Simulation has become a *de facto* in pilot training by providing enormous benefits including cost efficiency and safety.

Surgical planning and training systems are increasingly becoming popular and gaining significant attention from the medical community. This is because they have proven to be a very powerful tool for the planning surgical operations and for training surgeons who can learn and rehearse complex surgical operations without any risk to the patient. In addition, such systems encompass some of the most challenging fields

of research in computer graphics: physically correct tissue deformation, real-time performance with haptic feedback and photo realistic visualization. A typical computer-based medical simulation system mainly consists of:

- Medical image data analysis: Data acquisition and segmentation.
- Simulation module: Polygonal model generation and simulation algorithms.
- Haptic feedback: Collision detection and interaction with the user.
- Virtual Reality: Visualization and texture mapping.

Designing such a simulator also requires inputs from a variety of disciplines including computer graphics, computer vision, robotics, material science, mechanics and finally medicine itself. A number of attempts in developing such simulators are reported in literature (KISMET, Haptica, Simulab). The essential part of any simulator of this sort is the modeling and simulation of soft tissue behavior. Only then the objective of achieving physically realistic tissue deformation in real time will be met. Ideal modeling and simulation algorithm needs to meet the following requirements:

- Speed: 30 Hz for visual update rate and 1000 Hz for haptic update rate.
- Physical realism: Nonlinear and inhomogeneous tissue characteristics and interaction with the environment e.g. surgical tools, other organs.
- Topological changes: Cutting and suturing.

There have been numerous simulation methods proposed (Gibson and Mirtich 1997) that are mainly divided into two categories: non-physically based and physically based. There are some fundamental limitations in non-physical methods such as, the deformation characteristic of the object is not taken into account and the deformation

accuracy is based on the user expertise. Physically based models, on the other hand, incorporate the physical properties of the object, thus produce more realistic deformations.

Among physically based methods, finite element modeling and mass-spring systems have been widely used in a variety of areas from cloth simulations to soft tissue simulations. An example of a deformation simulation is given in figure 1.1, where an object is subject to a deformation due to its interaction with a tool. In the following section we outline the advantages and limitations of such methods. We also propose our strategy to satisfy the requirements of the ideal modeling and simulation system mentioned above.

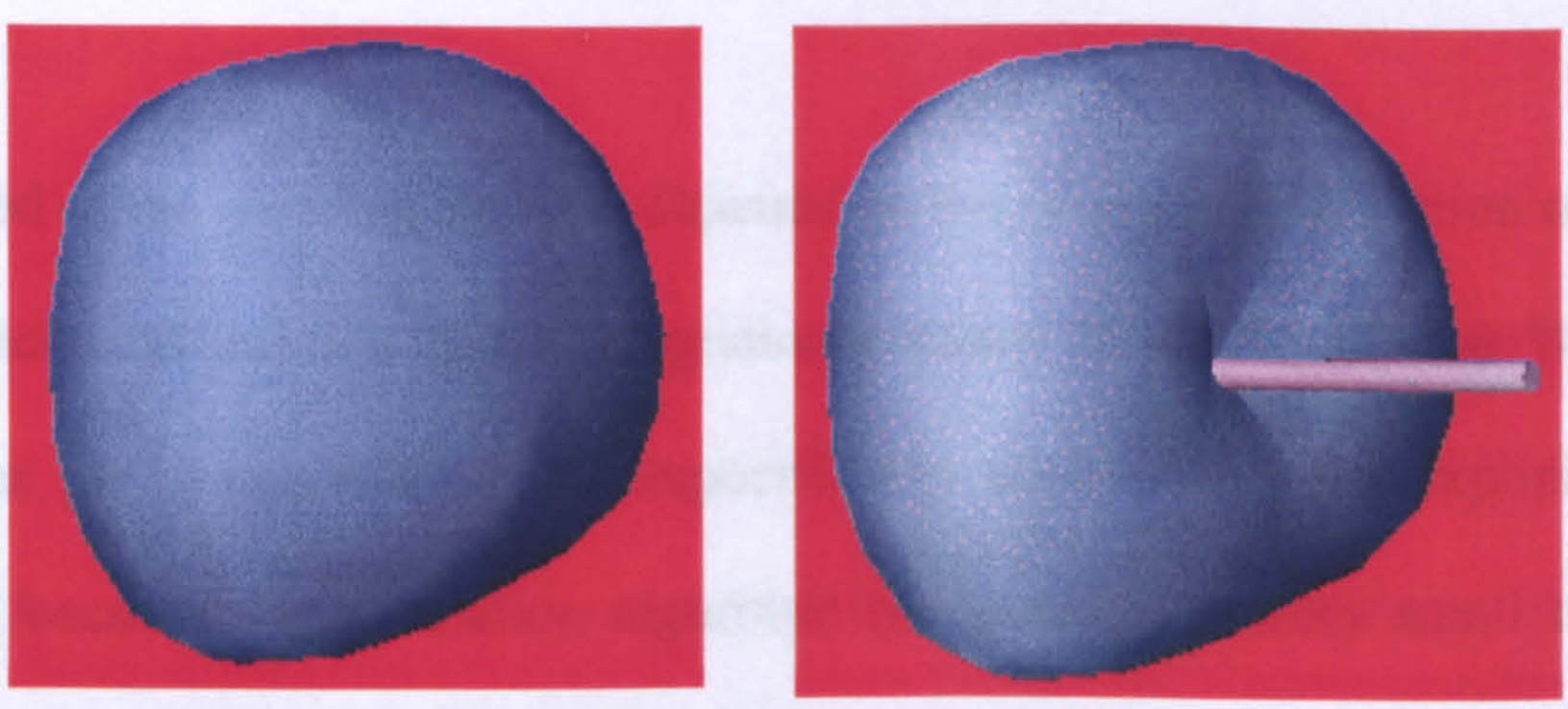


Figure 1.1 Deformation: An object interacting with a tool.

1.1 Motivation

The finite element modeling technique produces more accurate results for soft tissue deformations, as discussed by many previous publications. These publications also reveal that achieving real-time performance is a very difficult task and that reaching haptic interaction speeds is almost impossible to realize with the current technology. Mass-spring systems on the other hand, as indicated by previous publications, produce plausible results and are potentially suitable for real-time interactive

applications (Keeve et al. 1998). This thesis therefore focuses on mass-spring systems.

Accuracy of mass-spring models mainly depends on the choice of simulation parameters such as, the spring stiffness and damping coefficient (Louchet et al. 1995, Jaukhader and Laugier 1997, D'aulignac et al. 1999, Nurnberger et al. 2001, Bhat et al. 2003). In practice, such parameters are mostly determined by trial and error, based on the visual results of the simulation. This is not only time consuming but also determining physically correct parameters that capture complex deformation characteristic of soft tissue is almost impossible. As a consequence, almost all-existing applications use constant coefficients, which are far from capable of representing nonlinear tissue behavior.

The speed of mass-spring systems depends on the choice of integration methods such as, explicit and implicit Euler integration (Provot 1995, Baraff and Witkin 1998). Real-time performance is almost impossible to achieve with the explicit integration method, because the simulation algorithm needs to use a very small time step for stability reasons. The implicit integration scheme allows the use of large time steps but still the performance of overall simulation depends on the number of elements used i.e. mass-points and springs. There are other derived integration methods proposed to further speed up the mass-spring algorithm. The approximated implicit method works faster than the main integration scheme, implicit integration, at the expense of simulation accuracy. Similarly, the quassi-static method sacrifices realism in order to achieve faster simulations. Even with these speed up methods the target visual and especially haptic update rates are difficult to reach for many applications. This is because mass-spring systems use an iterative approach to reach steady state.

We propose the following: We investigate the use of neural networks in soft tissue simulation applications in terms of increasing the simulation accuracy (physical realism) and performance (speed). To this end, we will study mass-spring systems and neural networks. Our aim is to develop and integrate a neural network parameter identification method into the mass-spring system. Since real parameters representing deformation characteristics of soft tissue will be learnt, by the neural networks, the simulation algorithm will produce physically realistic deformations. The second part of the study (increasing speed) yields the development of a new deformation algorithm whose structure is not based on an iterative approach. Both old and new algorithms will be applied to model and simulate craniofacial surgery where soft tissue deformation is predicted.

1.2 Related Work

In our work we have carried out extensive research on simulation techniques, finite element methods, mass-spring systems and the ChainMail algorithm. We have also examined various other subjects. In order to be able to obtain medical models we studied medical data analysis. This study enabled us to read, segment and manipulate medical data in various formats. The geometrical representation of such data was also part of our work. To this end, we have studied triangulation and decimation algorithms. Since this thesis investigates the use of neural networks in soft tissue modeling and simulation, neural networks, system identification and parameter identification was also the focus of the thesis.

In this section we provide a detailed literature review on the related subjects studied in this thesis.

1.2.1 Finite Element Models (FEM)

A FEM is a standard simulation and analysis method extensively used in various engineering applications. It has also gained popularity in the simulation of soft tissue deformation. The advantage of the FEM method is its unsurpassed computational accuracy compared with other simulation methods. This is due to the fact that accuracy usually is of paramount concern in many engineering applications. The downside of this method, however, is its substantial computational cost. Although this is not really a problem in engineering practice, it does represent a difficulty in interactive graphics applications.

A FEM method assumes that the model has potential energy, which is related to the displacement of tissue from its initial to its deformed condition (Bro-Neilsen and Cotin 1996). A solution to the deformation is then found when the potential energy is minimized. In order to solve a deformation problem, the finite element method discretizes the region of interest into a set of finite elements (i.e. tetrahedra). A sample disretizetion is shown in the figure 1.2 where a tetrahedral element is shown as well.

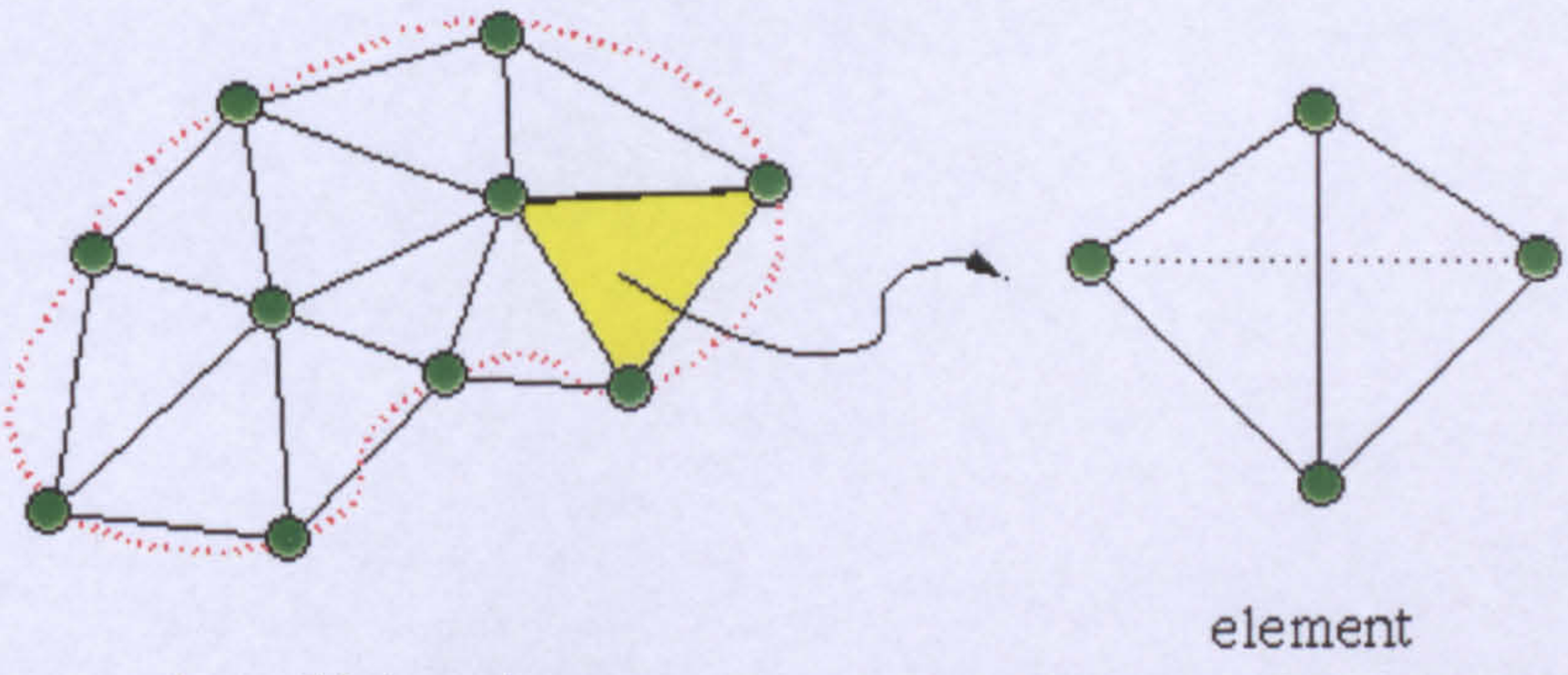


Figure 1.2 Continuous domain is subdivided into discrete elements, FEM representation.

Displacement of a point in the element is expressed as a function of the displacement of the vertices of the element. An element vertex displacement vector for the deformable body can be described as:

$$\underline{\mathbf{u}}^e = [(\mathbf{u}_1^e)^T (\mathbf{u}_2^e)^T ... (\mathbf{u}_n^e)^T]^T$$

where $\mathbf{u}_i^e$ are the vertex displacement vectors of each element e. The displacement at a point x is expressed in terms of vertex displacements as follows:

$$\mathbf{u}(x) = \sum_{i=1}^{n^e} N_i^e(x)\mathbf{u}_i^e$$

where $N_i^e(x)$ are the shape functions defined on each element (e.g. the natural coordinates of the elements). At equilibrium, the first variation of the total energy is given as:

$$\sum_e \delta E^e(\mathbf{u}) = 0$$

where $E^e(\mathbf{u})$ is the total energy of the model under load (DiMaio 2003). This equilibrium equation can be expressed as a linear matrix representation:

$$K\mathbf{u} = \mathbf{f}$$

where K is the global stiffness matrix, $\mathbf{u}$ is the vertex displacement vector for the entire model and $\mathbf{f}$ is the force vector. For a given force vector the above equation is solved for the vertex displacement vector representing the tissue deformation.

1.2.1.1 FEM Applications

Early works generally used pre-computed linear FEM models; Bro-Nielsen and Cotin (1996) studied the use of FEMs in surgical simulation. They implemented a linear elastic material model, which yields a linear matrix system along with a condensed system in order to achieve real-time performance. Condensation is achieved by

eliminating internal nodes from the system equation. For cutting and tearing operations, performed in real time, represented by FEMs they used a domain decomposition technique. Fast finite element modeling (Bro-Neilsen 1998) was developed and was applied in their later work (Bro-Neilsen et al. 1998) on abdominal surgery simulation.

Cotin et al. (1999) also used linear elasticity theory in their FEM application to simulate soft-tissue behavior. They admitted that "Since pre-computations allowing real-time interactions depend on the geometry of the mesh, it seems impossible to use only finite element models in the simulator." In their more recent work (Delingette et al. 1999, Cotin et al. 2000), they introduced a hybrid elastic model that consists of a static and a dynamic model of the organ. The static model pre-computes the deformations and forces, thus allowing real-time interactions. This model, however, does not permit any topological changes, such as cutting or tearing. Their dynamic elastic model requires more computation time, but allows topological changes to take place. Besides implementing linear elasticity, the combined model has two drawbacks. Firstly, the contact regions must be predefined so that the dynamic model is assigned to these parts. Secondly, interactions between static and dynamic models at boundaries are not well defined.

There are some publications dedicated to facial tissue simulations using FEMs. An implementation of FEMs for soft-tissue simulation is given in (Roth et al. 1998). Instead of linear elasticity theory, Roth et al. used a higher order polynomial interpolation functions using a Bernstein-Bezeir formulation, therefore aiming for more accurate results and admitting higher computational cost. The main drawback of their work is the lack of global C^1 continuity, which results in lower quality surfaces.

An anatomy based 3D finite element tissue model was developed by Keeve et al. (1996a, 1996b). Their work includes a comprehensive flexibility that allows for any craniofacial operation on the bone structure. Keeve et al. improved their early work by taking into account the individual patient's anatomy (1998, 1999). They used six node prisms to discritise the face model.

Koch et al. (1996, 2002) developed a facial surgery simulation based on volumetric finite element modeling. Their implementation aims for physical accuracy therefore includes geometric and topological detail added interactively to the model, which represents a facial volume by prismatic shape functions. This model provides globally C^1 and internally C^0 continuity. In their work, they registered 3D laser scan data with CT data to achieve photo realistic appearances. Results from their simulations are compared to real surgery images for several different patients. Koch et al. (1998) also used a FEM model to implement a facial expression editor. Their generic facial model uses medical data and correct facial anatomy in defining muscle groups. Accuracy was the main concern in early applications, while speed became more important in recent works (where researchers modified the FEM algorithm in order to increase its performance).

1.2.1.1 Recent Trends

In recent years researchers have focused on improving the algorithm in order to achieve physical realism and real-time performance. Debunne et al. (2000, 2001b) have implemented a space and time adaptive sampling method to achieve real-time performance. The idea here is that the algorithm adaptively adjusts the resolution in the regions that deform the most. They also address the problems associated with adaptive sampling such as different vibration modes between different resolutions, as

well as contact regions between them. In order to avoid vibration, which yields instability, they used arbitrarily and independently defined meshes representing levels of details. In order to handle large displacements and global rotations they used the (green) strain tensor formulation. Some researchers developed better FEM models (Wu et al. 2001), which use adaptive nonlinear finite elements. The aim here is to improve the realism by using nonlinear terms in their modeling. In addition to this, they also implemented an adaptive mesh refinement scheme to allow real-time applicability. Similarly, a multi-scale finite element algorithm was developed (Lim et al. 2004) in order to increase the performance of the original algorithm, which concentrates detail where it is needed while still providing global deformations. In (Hauth et al. 2004) a polar decomposition technique was implemented with hierarchical finite elements. Their work enhanced the stability and increased the speed at the expense of higher error.

1.2.2 Mass-Spring Systems (MSS)

The use of MSSs leads to a simplified simulation model. In this model any deformable objects can be represented as a set of mass points connected with springs (Baraff and Witkin 1999, 2001). A simple representation is given in figure 1.3. Although this configuration does not accurately represent the physical structures of the objects to be simulated, it is usually able to achieve perceptually realistic effects as long as the parameters of the MSS, such as spring stiffness and viscosity, are properly specified. Because of their simplicity and generality, mass-spring systems have become indispensable tools in many simulation applications, such as cloth animation, facial animation and surgical simulations. A detailed study of this technique is presented in chapter 2. Some of the applications using mass-spring systems are shown below.

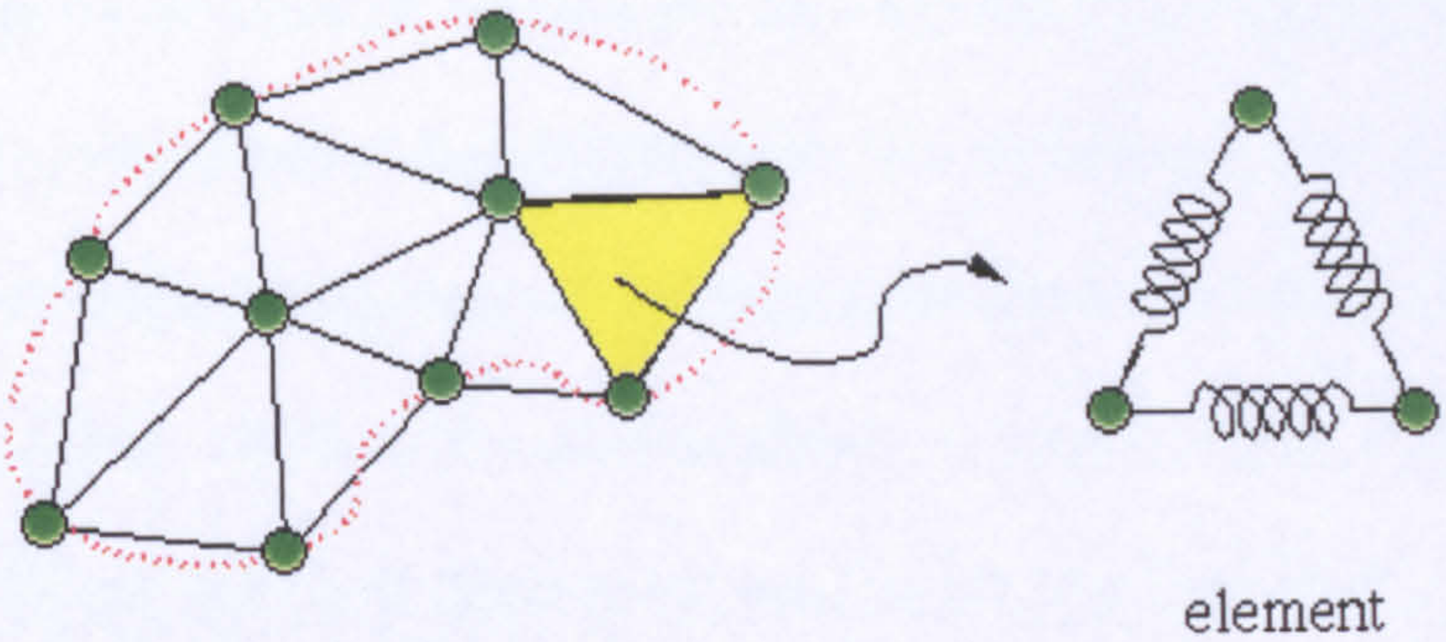


Figure 1.3 Mass-spring representation of a deformable body.

1.2.2.1 MSS Applications

Mass-spring systems are successfully used in cloth-like object animations. Provot (1995) simulated cloth behavior in his early work, where he implemented a detailed mass-spring system structure for cloth modeling and studied the super-elastic effect. An adaptive refinement scheme is described in (Hutchinson et al. 1996) where the algorithm simplifies the geometric mesh adaptively. Unwanted coarse discretization is avoided and the speed of the algorithm is therefore improved. In (Howlett and Hewitt 1998), adaptive components called non-active points were introduced to a mass-spring system in order for the new system to adapt the shape of the cloth at different types of edges. Therefore, visually convincing cloth animation is achieved (especially the simulation of draping cloth over irregular objects). In (Eischen and Bigliani 2000) a comparison of the algorithms in cloth simulation was studied. In their paper, they provided a detailed study of the use of FEMs, MSSs and particles in cloth modeling.

Baraff and Witkin (1998) studied one of the biggest problem of mass-spring system; the small time step that is necessary to stabilize the system at the cost of a slower simulation time. Their simulation system couples a new technique for enforcing constraints on individual cloth particles with an implicit integration method. They

used a combination of implicit integration and the direct constrained satisfaction method. Instead of using reduced coordinates for the acceleration or the velocity of a particle, which introduce complications by altering the size of the derivative matrices, or penalty functions, which add extra stiffness to entire system, or additional Lagrange multipliers, which introduce extra variables into the system they employed a different approach. They modified the mass of particles. In their differential equation, a constant inverse mass is used for each particle. If a 3-dimensional matrix representing the mass of each particle is used, they show that constrains can be imposed on dimensions. The modifications allow them to use larger time steps providing an interactive frame rate for many applications. Their implicit integration scheme allows the mass-spring system to be used in interactive animation.

Desburn et al. (1999) derived an integration scheme from the implicit method in order to achieve real-time interaction for virtual reality applications using mass-spring systems. They further improved their early work in (2000) and in (Debunne et al. 2001a) for interactive animations of cloth-like objects in virtual reality. The implicit integration scheme was modified to avoid solving a linear system through an approximation of the Hessian matrix. A predictor-corrector approach was introduced to speed up the system, i.e. positions of vertices at the next time step were predicted and corrected. Approximation was done by splitting the force equation into two parts, namely, the linear and nonlinear components. Correction was done through an inverse dynamic relaxation phase, which also included the simulation of nonlinear behavior.

Mass-spring systems are also used in a variety of other simulation applications. In (Duchille et al. 1999), the authors introduced a novel haptic approach for the direct manipulation of physics-based B-spline surfaces. Their mass-spring system permits

the user to interactively sculpt virtual material with a standard haptic device with force feedback. This work demonstrates the interactive potential of mass-spring systems. By employing either a mass-spring system or a FEM in (Chen et al. 1998), a voxel-based animation technique is presented. In (Giacomo and Thalmann 2003), a mass-spring system is modified in order to increase speed in the simulation of flexible thin or thick liner bodies, such as strings, ropes or pipes.

An MSS is also used in soft-tissue simulations. A real-time muscle deformation using an MSS was studied in (Nedel and Thalmann 1998). Using a new kind of spring type, called an angular spring, the authors simulated a surface-based muscle model. Terzopoulos and Waters (1991) and Lee et al. (1995) successfully used an MSS for realistic facial modeling and simulation. A detailed facial model represented by a four-layered mass-spring model. In their motion equation, they include volume preservation forces and other constraint forces, such as skull penetration forces. U. Kuhnapfel et al. (1999, 2000 and Cakmak and Kuhnapfel 2000) implemented an MSS in their system for simulating soft tissues. They integrated a mass-spring system simulation module into their surgical training system, which is capable of performing several surgical tasks (such as grasping and cutting). In (Webster et al. 2001) a mass-spring system is used in a prototype haptic sturing simulator. They designed a simulator system working in real-time to teach basic suturing for simple wound closure.

In recent years, researchers have focussed on improving physical realism as well as performance of the mass-spring algorithm.

1.2.2.2 Recent Trends

Some researcher focused on improving the performance of the MSS (Kang et al. 2000a, 2000b, 2001). Here the authors introduced a method to overcome the computational burden of the implicit method. The velocity of vertices at the next time step is approximated using a special formula therefore avoiding having to solve the Hessian matrix as in (Baraff and Witkin 1998) or use the pre-computed filter as in (Desburn et al. 1999). They also addressed the stability issue of such a system by employing a stable damping function. Although their proposal for damping does not represent the real damping force it provides overall stability. Their work purely aims to produce faster simulations at the expense of the accuracy of the results. Bourguigron and Cani (2000) studied the excessive spring elongation causing an unrealistic appearance in MSS applications. In their work, the user can choose a region of interest and the mechanical properties of the material along a given number of axes. Since these mechanical properties and force directions are manually changed for the specified regions where spring elongation may occur, this work provides a means of controlling anisotropy. A solution to excessive spring elongation was proposed by Provot (1995), which however does not have any mathematical justification and its sole purpose is to achieve good visual effects.

Teschner et al. (1999a, 199b and 2000) used an optimization approach to improve the physical realism as well as the performance of mass-spring systems. A multi-layer soft tissue model of the head was developed including the skin turgor and sliding effect between soft tissue and bone. They employed a variety of different optimization methods and compared them with regard to the computational cost and the robustness of their results. They showed that the conjugate gradient method is a fastest method, which provides reliable results. They achieved a speed of less then 4 seconds for a

facial model consisting of 2092 points and 16547 springs. Although simulation time is increased, it is not clear how accuracy is improved. Brown et al. (2001) used MSS for soft tissue simulation with some modifications, such as the introduction of a quasi-static algorithm with real-time performance. In their quasi-static algorithm they neglected dynamic internal and damping forces, creating a static but simplified equation. They also constrained the iteration process so that the iteration is concluded within a given time interval, guaranteeing real-time performance. In order to enhance the performance of such a method they developed a node-ordering algorithm, which only takes into account regions of the model saving major computational costs. Their method reaches 24 iterations at 30 Hz for a model with 216 nodes and 1440 edges. Again accuracy is sacrificed to gain speed.

A new simulation technique using MSS was proposed in (Muller et al. 2002) where stability and performance is addressed. A linear stiffness matrix is precomputed and a tensor field describing local rotations is calculated during the simulation. Their method allows the use of larger time steps for linear and non-linear cases without effecting stability. Multiresolution techniques frequently used in FEM applications were also integrated into the MSS algorithm (Choi et al. 2002) for performance reasons. Using a modified butterfly interpolation subdivision, which produces smooth surfaces while preventing shrinking, their method locally refines the surface mass-spring models at different levels of detail. They also introduced a shape preserving spring to prevent the surface model from collapsing during simulation. In their paper, they present a comparison between the conventional implementation and their method. For a model that consists of 450 nodes and 1344 spring, while conventional method takes approximately 10 seconds to compute, their method reduces the computation time to approximately 3 seconds. The mesh refinement introduces some

problems that need to be addressed, such as the oscillation between different regions and the problem of interaction between them.

1.2.3 The ChainMail Algorithm

The ChainMail algorithm performs deformation locally comparing only two neighboring nodes. It therefore does not involve any matrix inversion or iteration in the deformation process. When the volume is manipulated, the object stretches and contracts to satisfy the minimum and maximum allowable distance between the neighboring elements (Gibson 1997). These simple deformation rules enable the ChainMail algorithm to perform deformations at an interactive rate. The basic concept of the algorithm is illustrated in figure 1.4. Each element in the volume is linked to its six nearest neighbors. Displacements are transferred to neighbors by these links.

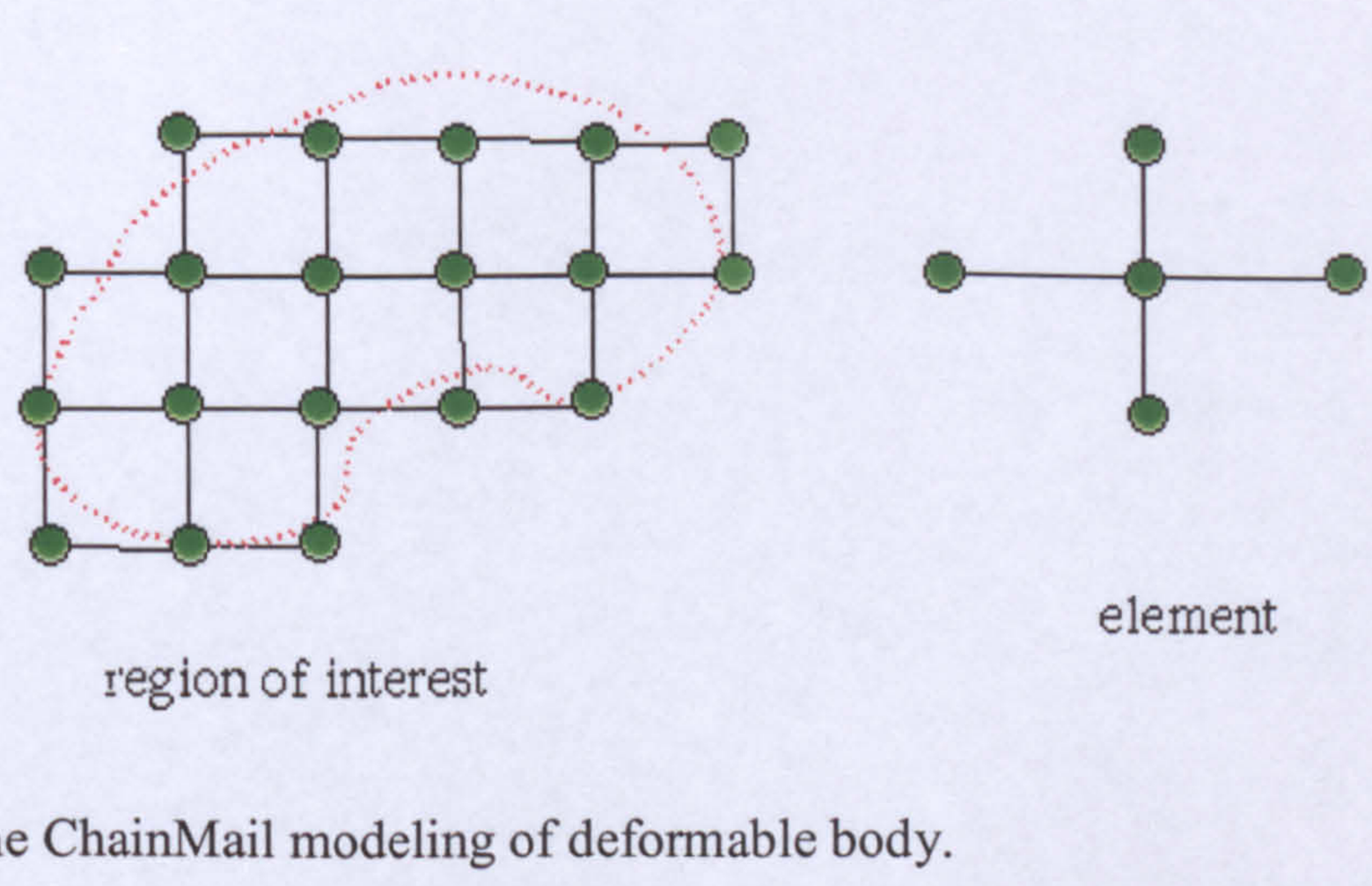


Figure 1.4 The ChainMail modeling of deformable body.

The allowable distances (limits) between the neighboring elements are defined as follows. Each element must lie within a horizontal range of min Dx and max Dx from its left and right neighbors. It must lie within a vertical range of min Dy and max Dy from its top and bottom neighbors. These limits control the stretching and contraction of the material. In addition to these limits, each element must lie within

+/− max *HorizDy* from its horizontal (left and right) neighbors and within +/− max *VertDy* from its vertical (top and bottom) neighbors. These limits control the maximum amount of shear that is possible in the material.

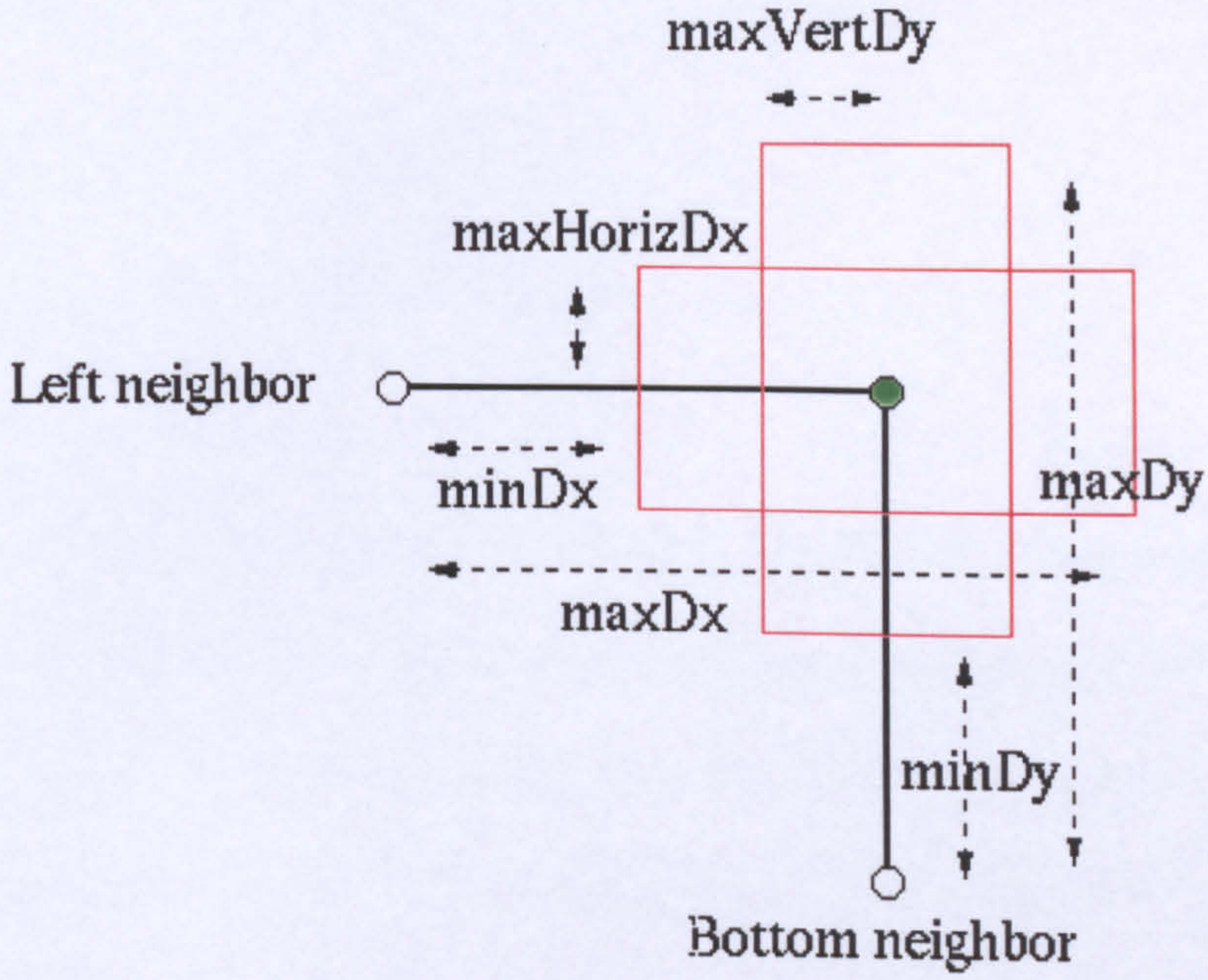


Figure 1.5 The set limits for the deformation and the deformation region.

A possible deformation region defined by these limits is shown in figure 1.5. This figure only shows a 2D representation of this algorithm. The 3D extension is given in (Schill et al. 1998). In the 3D version of the algorithm a cubic cell is formed around each element and the same rules for deformation apply. When an element (point) is moved, its old and new positions are added to a list of moved elements. Its six nearest neighboring elements are also added to the top, left, bottom and right lists of the candidates for movement. The deformation algorithm then processes these candidate lists in turn from right, left, top and bottom order. The set limits are checked and if they are violated, necessary adjustments are made according to the defined limits. An elastic relaxation step further ensures that neighboring elements satisfy the constraints.

The deformation conditions for the ChainMail of figure 1.5 is given by following *if-then* construct:

$$if\,(x - x_{left}) < \min Dx \;\text{ then }\; x = x_{xleft} + \min Dx$$

$$if\,(x - x_{left}) > \max Dx \;\text{ then }\; x = x_{xleft} + \max Dx$$

$$if\,(y - y_{left}) < -\max HorizDy \;\text{ then }\; y = y_{left} - \max HorizDy$$

$$if\,(y - y_{left}) > -\max HorixDy \text{ then }\; y = y_{left} + \max HorixDy$$

Although the ChainMail algorithm allows the deformation of models containing thousands of elements in real time, it has not gained sufficient popularity among the scientific community. This may be due to its simple deformation rules that are only designed for efficiency. A number of applications of the ChainMail algorithm are given below.

1.2.3.1 The ChainMail Applications

The ChainMail algorithm was designed to produce very fast deformation rates because only very simple deformation rules are defined between the individual elements of the chain. This algorithm therefore is aimed at interactive frame rates for large networks and sacrifices accuracy. This algorithm was further developed in (Schill et al. 1998) where the modeling capabilities of the original chainmail algorithm were further expanded to handle inhomogeneous material. A detailed analysis of the ChainMail algorithm and its applications are given in (Gibson 1999), where collision detection was also examined. An elastic relaxation phase was also integrated into this algorithm. The ChainMail algorithm was modified in (Park et al. 2002) to take into account the residual energy left in the model after its interactions with other objects. Their work emphasized the shape retaining property, which was

not addressed in the original algorithm. They also implemented a force-voltage analogy concept into the ChainMail algorithm in order to compute reflected forces.

1.2.4 Medical Data Analysis

In order to perform surgical simulations, patient-specific medical data from various imaging modalities must be obtained. The image then undergoes various medical image-processing operations, such as registration, filtering and segmentation. Other necessary operations, including measurements, cutting and separation are also performed. This phase can be called the pre-simulation phase, which prepares medical data for the simulation as well as helps the user to decide details of the operations and treatments to be applied to the data.

There are many commercially available medical data analysis packages that deal with all the image processing necessary for the pre-simulation phase. Some of the well-known commercially available programs are Analyze (Analyze), Amira (Amira) and 3D Doctor (3d Doctor). Some of the programs are freely available, such as 3D Slicer (3D Slicer) and Opedx (Opendx). In our work we use a program called 3DWIEVNIX (3Dwievnix) developed by the University of Pennsylvania. This is a simple program that enables us to segment and manipulate medical data.

There are research groups that specialize on medical data analysis and simulations. A virtual reality training system for minimally invasive surgery was developed in Germany (KISMET). MIRLab (MIRLab) has long been specializing on medical data analysis and simulations. Some of the other research centers are based at Stanford University (Stanford), INRIA (INRIA) and the computer graphics lab (CGL).

1.2.5 Polygonalization

Once the medical data is registered and segmented, we need to represent its voxel data by a set of geometric primitives suitable for the simulation algorithm. The FEM and MSS methods work on triangular (tetrahedral) representations. There are many different methods for converting voxel data into a surface representation. One method uses an unorganized point set in order to do the triangulation based on either a surface or a volume approach (Amenta et al. 1998, Boissonnat 1984, Veltkamp 1995). Another method was reported as a triangulation using contours, (Christiansen and Sederberg 1978, Edelsbrunner and Mucke 1994, Schumaker 1990). Since the medical images obtained from MRI or CT imaging modalities came as slices, a set of contours in each slice are then connected to each other with triangles. Lorensen and Cline proposed a well known and frequently used algorithm called the marching cubes algorithm (Lorensen and Cline 1987). This algorithm generates high quality meshes from volume data but produces a large number of triangles, which need to be reduced. Triangles, specifically in the flat regions, can be merged or removed in order to reduce the number of triangles representing the object without significantly compromising the accuracy of the approximation. This will allow for faster rendering, less storage space and simpler manipulations. There have been many algorithms reported in the field for mesh simplification (Schroeder et al. 1992, Ciampalini et al. 1997, Hoppe 1993, 1996, Ropovic 1997).

1.2.6 Neural Network System Identification

Neural networks were developed in an attempt to duplicate the information processing capabilities of the human brain. A neuron in the brain is a simple processing unit, which is connected to many other neurons. Each neuron receives a signal form other connected neurons through its input paths. A neuron fires (i.e., produce output) if the

received signal is strong and modifies its synaptic junctions (weights). This process constitutes learning. A representative diagram of a biological neuron is given in figure 1.6 and its basic representation (artificial neuron) is given in figure 1.7. Although modern computers, compared to the human brain, are much faster in numerical computation, the human brain is superior in solving complex problems. This is because simple neurons in the brain have so many interconnections making them massively parallel computing systems. This attractive property makes them widely useful in many areas of research including (Uhrig 1995, Jain and Mao1996):

- Modeling complex systems.
- Pattern classification.
- Function approximation.
- Prediction.
- Control.
- Character recognition.
- Text-to-speech conversion.

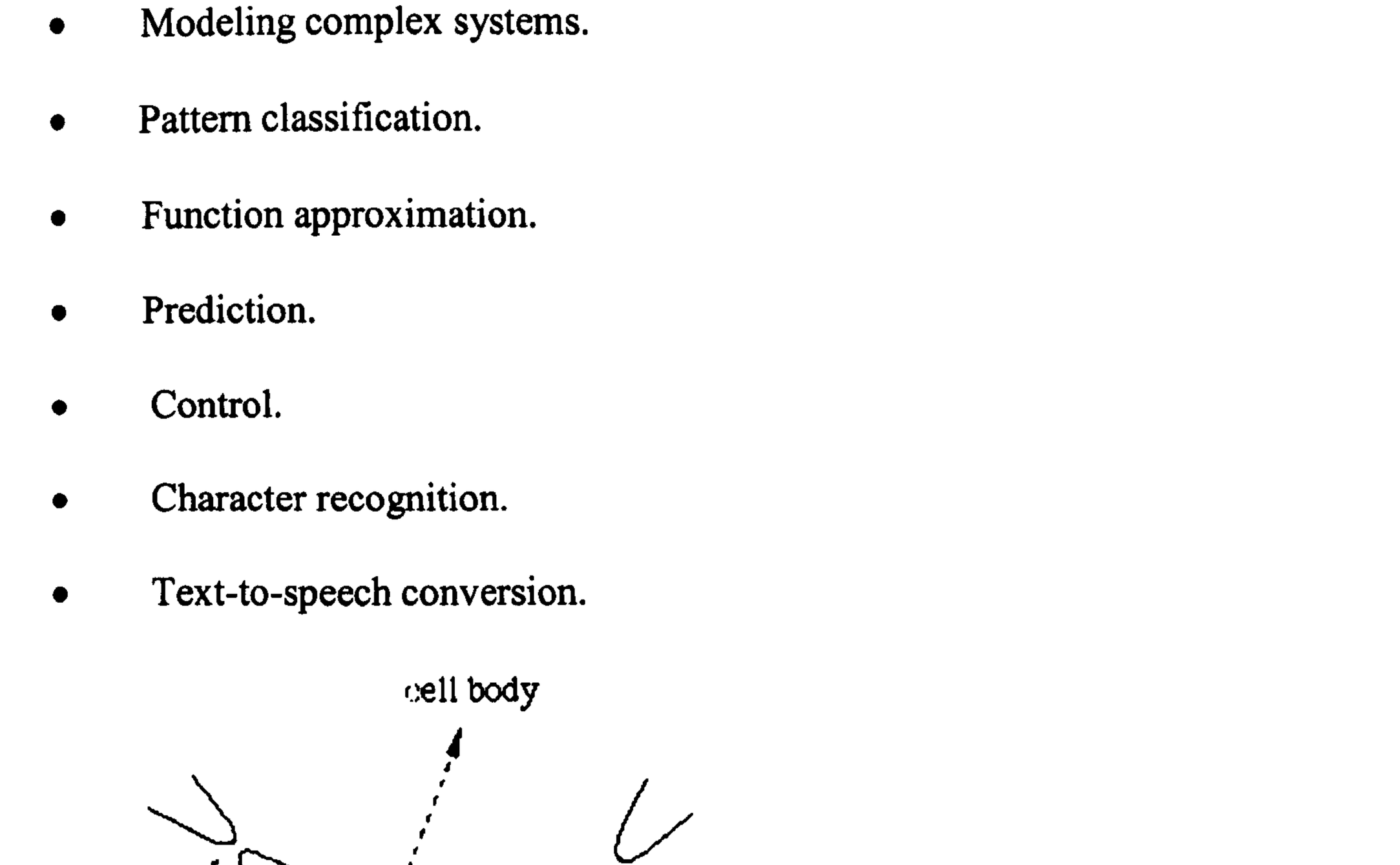


Figure 1.6 A sketch of a biological neuron.

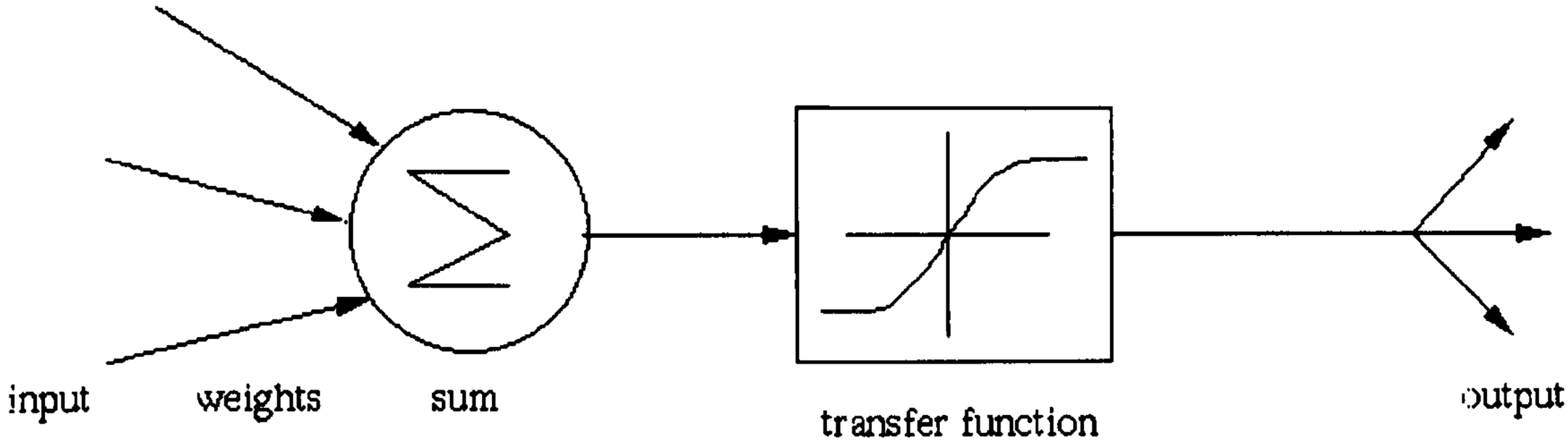


Figure 1.7 A basic representation of artificial neuron.

There are a number of neural network configurations proposed in the literature, including (Jain and Mao 1996): Multilayer feed-forward networks that are the most commonly used networks especially in data analysis, classification and function approximation. Kohonen's self-organizing maps can be used for projection of multivariate data, density approximation and clustering. Typical application areas are speech recognition, image processing and process control. Hopfield networks are recurrent neural networks and work on the energy minimization principle. They can be used as an associate memory and in finding optimal solutions.

In this thesis we use the Cerebellar model articulation control (CMAC) neural network, which is a two-layer feed-forward neural network developed by Albus (1972, 1975a, 1975b).

1.2.6.1 CMAC Neural Networks

CMAC neural networks did not receive much attention until the late eighties, because of their large memory requirements and the lack of analytical results regarding their training algorithm convergence. Recently, however, it has been shown that CMACs are a powerful alternative to the well-known back propagation trained multilayer

neural networks (Miller et al. 1990). In comparison to other neural network architectures, the CMAC network has two desirable features. First, the training algorithms converge at speed orders of magnitude faster than back propagation learning for real problems. Second, there are no local minima on the error surface used in training.

CMACs gained more attention after Miller (1987) used the CMAC network for real-time control of a full-scale multidegree-of-freedom industrial robot with considerable success. CMAC controllers are widely used in robotic applications. For example, Miller (1989) applied a CMAC-based controller to a robot system, which used image feedback. Another application of CMACs was reported by Lin and Song (1992). CMACs are utilized to perform feed-forward kinematics control of a four-legged robot in straight-line walking and step climbing. Nelson and Kraft (1994) used a CMAC neural network in a pole-balancing system. They designed a linear regulator controller and used a CMAC network to adjust the linear controller in order to compensate for nonlinearities and noise. Shiraishi et al. (1995) used a CMAC controller for a fuel-injection system and experimentally evaluated the CMAC's performance on a research vehicle in a configuration fully compatible with production hardware.

The performance of CMAC neural network control systems has been compared against that of other control methods. Kraft and Compagna (1990) compared a CMAC controller against a self-tuning regulator (STR) and a lyapunov-based model reference adaptive control (MRAC) with respect to closed-loop system stability, speed of adaptation, noise rejection, the number of required calculations, and system training performance. They found that the neural network approach functions well in noise,

works for linear and nonlinear systems, and can be implemented very efficiently for large-scale systems. Another comparison is given in (Ananthraman and Gargk 1993), where dynamic control of a robotic manipulator is achieved using both a back propagation-based neural network controller and a CMAC controller. This work indicated that the CMAC controller performed better than the back propagation-based controller, both in terms of trajectory tracking and the number of iterations required for a reasonable solution.

Although many papers describing the application of a CMAC controller have been published, far less literature is available on the stability of the closed-loop system and the convergence properties of the training algorithms. In (Kraft and Ho 1991), where the CMAC controller is used as part of the feedback loop, the stability of the CMAC network itself is analyzed in terms of a simplified linear model. They found that the eigenvalues of the open loop and closed loop systems depend on the structure of the CMAC neural network. Other researchers also investigated the stability of closed-loop systems using neural networks (Chen and Chang 1994). They showed that the CMAC-based control system is not guaranteed to be stable, and suggested some methods for improving its stability.

Parkz and Militzer (1992) studied five training algorithms for associative memory systems (AMS), particularly for CMAC networks. After testing the training algorithms, they recommended the maximum error (ME) algorithm. Two training methods are studied in (Thompson and Kwon 1995), the neighborhood sequential training and the random training method. In the first method, the strategy is to select points in the input space, which would train the CMAC system in the most rapid manner possible. The random method is found to converge on the training function

with the greatest precision, although it requires longer training periods than the neighborhood sequential training method.

Albus did not manage to prove that the CMAC training algorithm converges. It is now known (Wong and Sideris 1992) that the CMAC learning always converges with an arbitrary accuracy for a set of training data. Wong also proposed an alternative way to implement CMAC neural network. Improvements in the original CMAC neural network architecture have been also reported. A method for adaptively determining the quantization resolution was given in (Kavamato et al. 1995) and (Kim and Lin 1992). The quantization resolution was increased over the range where the input values were likely to occur so that more information would be captured. In (Wen et al. 1996), a mapping function, which mapped quantization input states to physical memory locations by using a memory banking technique was developed. This approach exploited the sparse distribution of weight addresses, and as a result reduced the memory requirement.

1.2.6.2 Parameter Identification

Most of the real-world dynamic processes are nonlinear and therefore developing mathematical models of such systems may not be possible due to the lack of information and difficulties in determining system parameters (Narendra et al. 1990). Using the observed input-output data, system identification is increasingly used by providing models such as the so-called "black-box" model where the system is completely unknown or the "gray-box" model where the system structure is known with undetermined parameters. If there is not enough physical information about a dynamic system or the system is too complex, then the "black-box" representation is used based on measured input-output data (Narendra et al. 1990, 1992). Figure 1.8

represents the general system identification model. Here the main idea is to apply the same input to both the neural network and the unknown system and to compare their outputs. The error between these two outputs is then used to update the weights of the neural network model until the error is minimized. At this stage the neural network is said to converge and the neural network model produces the same output as the unknown system for a given input.

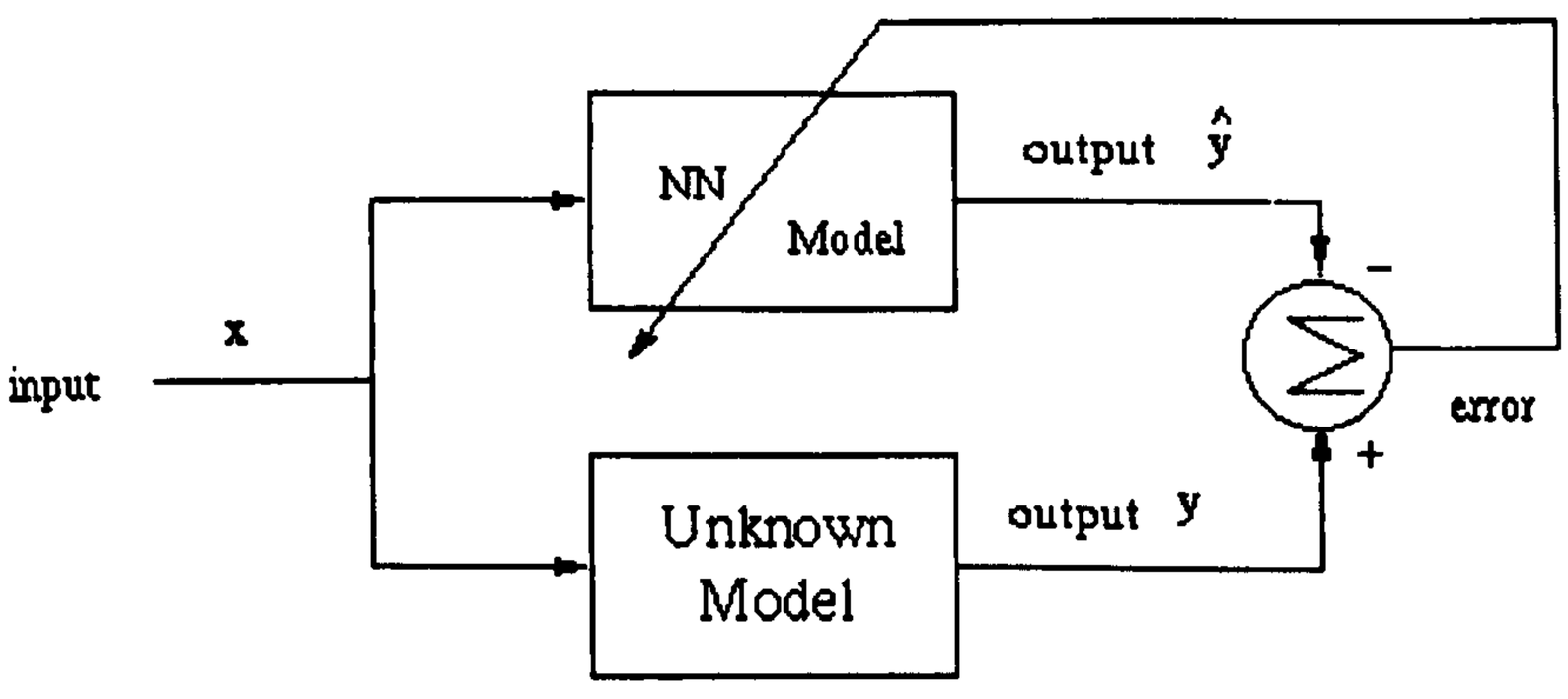


Figure 1.8 Identification of an unknown process using neural networks.

The determination of the spring parameters is crucial to capturing the non-linear material properties, as well as, to prevent unrealistic behavior of such systems while maintaining system stability. Much research effort has been expended on this research topic in recent years. In (Louchet et al. 1995), an optimization method was suggested to identify spring stiffness. The authors set up an objective function that is minimized using the positional difference between the simulation model and the real object for cloth simulation. In this work only the spring stiffness was identified. This is a relatively straightforward approach especially for constant coefficients. In (Koch et al. 1996), a method called intensity-based segmentation of volume data was used to compute the spring stiffness. This method did not consider the dynamic behavior of soft tissues because it assumed a relationship between color density and stiffness. There is no proof that this assumption reflects the behavior of its real life counterpart.

Using genetic algorithms, (Joukhadar et al. 1997) proposed a method for the identification of the parameters of an MSS in the simulation of a tie and its interaction with a robot. This was again done based on an assumed constitutive model. In a recent work (Bhat et al. 2003) an algorithm for estimating cloth parameters from video images was presented. This is again an optimization based model fitting approach. In their paper, they identified folds in the original cloth simulation and used a metric comparing both the video sequence and the parameter model. An optimization method was used to minimize the measured error. This approach may produce a parameter model mimicking the cloth behavior for similar situations where the error used in optimization process is taken. Their model lacks the generalization ability that is provided by neural networks.

Neural networks (NN) are a widely used approach in engineering applications for system identification (Narendra et al. 1990, Sjoberg et al. 1995, Pearson and Pottmann 2000). Recently, NNs have also found applications in MSS simulations and animation control (Bouzas and Arnold 1998). In (Grzeszczuk et al. 1998) NNs were used for various animation tasks. NNs were used to learn specific behaviors rather than system parameters. NNs were then used with a larger step size to speed up the animation process. Work relating to the identification of MSS parameters has been presented in (Ishikawa et al. 1998), where the authors used markers on a human face to capture the data representing facial expressions. The data were then used to train the NN to identify system parameters. However, no detail was given in the paper as to how the identification process was carried out.

A parameter identification method in a satellite orbit determination application was proposed in (Sinha 2000) where they compared the neural network approximation

with the Kalman Filter. A two-layer recurrent neural network was employed in (Lu et al. 2003) in order to identify the damper parameter of the switched reluctance motors. An online identification was carried out and results were validated from the operating data. NN parameter identification was implemented in (Almeida and Voit 2003) in order to identify parameters in S-system models of biological networks. Narendra et al. (1996) proposed identification models for unknown systems and for systems with unknown parameters. Based on Narendra's models a system identification and control has proposed in (Duysak 1997) using CMAC neural networks. In (Horvath et al. 1996) a CMAC neural network is used for system identification. They studied different ways of system identification by modifying the original network.

So far the most detailed research on this topic was carried out by Nurnberger et al. (1998, 1999, 2001). They developed a problem-specific neural network model for parameter identification using a mass-spring system and applied this technique in some medical visualization applications. Their identification network model represents the entire mass-spring structure using neural network. Each of the dynamic parts of the mass-spring system, spring stiffness, damping, spring force, acceleration, velocity and position represented by the neural networks. Using the only available data, the positions of mass spring, six neural networks were trained. They also addressed the basic concepts in neuro-fuzzy techniques for the identification and simulation of time-dependent physical systems. As examined later in the related chapter of this thesis, this identification model has its drawbacks and limitations. Their model however provides a good base for developing a system identification model specific to mass-spring system modeling and simulation. A detailed study of such a method is given in chapter 4.

1.3 Thesis Organization

Chapters in the thesis are organized as follows:

- Chapter 1: This chapter provides a detailed literature review on relevant subjects, deformation modeling algorithms (FEM, MSS, ChainMail), medical data analysis, polyganization methods, neural networks and parameter identification.

- Chapter 2: We study and implement a mass-spring system to simulate a variety of deformable objects such as cloth and soft tissue. A detailed theory of the algorithm is presented and a dynamic model is obtained. This chapter also reviews collision detection and collision forces. The spring elongation problem and its solutions are also addressed here. The advantages and disadvantages of such a method are outlined.

- Chapter 3: In this chapter, we examine medical data image analysis. Segmentation, filtering measurements and manipulations (i.e. cutting and separation) fall in the scope of this chapter. The main purpose of this chapter is to produce a patient-specific head model suitable for our applications. This head model further undergoes triangulation and has a decimation algorithm applied to it in order to generate a polygonal model suitable for mass-spring simulations. At this stage, polygonal representations of skin and bone surfaces are further modified to obtain a volume tissue model including different tissue layers. We then simulated facial tissue deformation caused by the underlying bone realignment using a mass-spring algorithm. The simulation application also includes facial animation by the developed head model.

- Chapter 4: We integrate the application of neural networks to mass-spring systems in order to increase the physical realism of the simulations. This results in a unique identification method specific to a mass-spring system. Our neural network identification method successfully learns unknown nonlinear system parameters and therefore increases the physical accuracy of the simulation outcome. Neural network algorithms in general and particularly the CMAC neural network are studied in detail. The neural network system and its parameter identification are included in this study.

- Chapter 5: We examine simulation methods along with FEM, MSS and ChainMail and come up with a deformation technique suitable for a neural network implementation. A detailed theory of the developed model is presented in this chapter. Its application to various deformation simulations, including the simulation of craniofacial surgery is given. A comparison with a mass-spring system for accuracy and speed is presented. A comparison with other applications presented in the literature in terms of performance is given.

- Chapter 6: We finally discuss the work done in this thesis as well as provide a list of our contributions. Future work is also proposed.

CHAPTER 2

MASS SPRING SYSTEMS (MSS)

Mass-spring systems have been widely used in computer graphics applications because of their simplicity, speed and their acceptable accuracy. This chapter presents a detailed analysis of mass-spring theory, its implementation and its application to cloth-simulation as well as soft tissue deformation. Since the theory of mass-spring systems is a well-studied subject we take a different approach in this work. By dividing its dynamic structure into two parts we analyze the individual dynamics of such systems and drive their governing mathematical equations. Our analysis is constructed in a way that yields a dynamic model development to be used in neural network system identification in later chapters. The study includes different integration schemes, collision detection and forces. We also address the downsides of the mass-spring algorithm such as the determination of system parameters and stability of the simulations.

2.1. Mass-Spring Structure

In mass-spring simulations the geometry of a deformable object is represented by a 3D mesh consisting of n nodes, which are interconnected by m links. Each node in the mesh represents a virtual mass and is called a mass-point. Displacements of these mass-points describe the deformation of the object. The total mass of the object under

consideration is distributed among these mass-points. The links between each pair of mass-points are virtual springs with damping elements. It is assumed that the springs are weightless. These springs define the distance relationships between mass-points because the deformation characteristic is embedded into the 3D mesh by spring parameters; namely stiffness and damping. A simple representation of such a system is shown in figure 2.1. The object under consideration is firstly represented by a set of geometric elements (triangles). The edges of the triangles are then considered as being springs and their vertices are assigned as mass-points. Each spring element consists of two parts. One part simulates resistance to external forces while the other is responsible for energy dissipation.

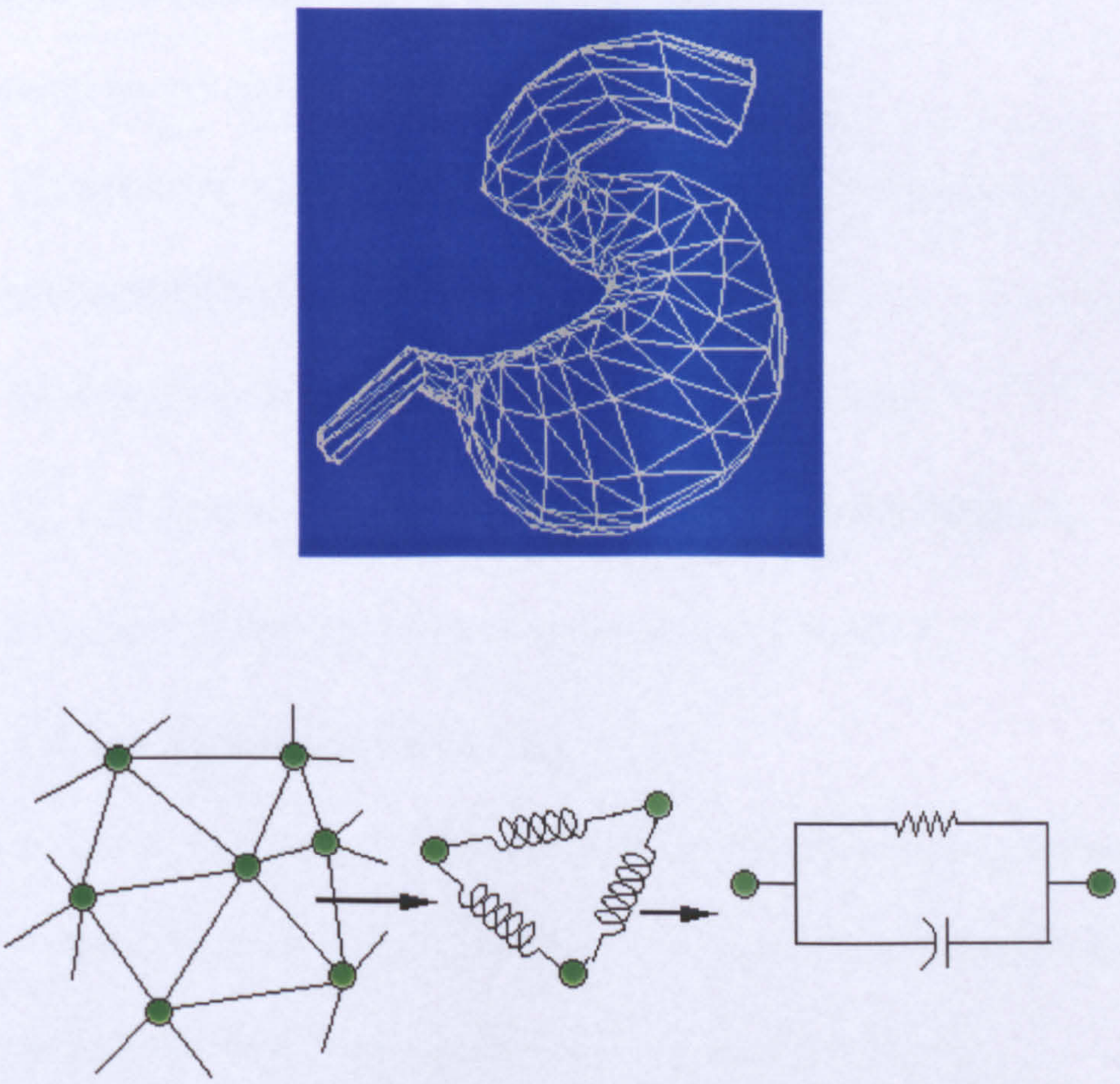


Figure 2.1 Illustration of a 3D Mesh and its mass-spring structure.

The mass-spring system given in figure 2.1 can be considered as a mechanical system whose behavior is described by a coupled system of second-order ordinary differential equations. Before analyzing the system dynamics, in detail we present a brief description of the notation used throughout this thesis.

- $\mathbf{X}_i (i = 1,2,...,n)$ is the position of the mass-points ($\mathbf{P}_i$) in the 3D mesh given in Cartesian coordinates.
- m_i is the mass of each point in the mesh and may be the same for all the mass-points.
- K_{ij} and D_{ij}, $(j = 1,2,...,n)$, are the spring stiffness and damping factors associated with each link or spring between mass-points i and j, $i \neq j$. For clarity we will omit the use of indexes.
- $\mathbf{f}_K^{ij}$ represents the spring force caused by spring stiffness. In the following sections this force is abbreviated to $\mathbf{f}_K$ to aid readability.
- $\mathbf{f}_D^{ij}$ is the damping force and it is similarly abbreviated to $\mathbf{f}_D$.
- $\mathbf{f}_{\text{int}}^{ij}$ is the internal force (total spring force), which is abbreviated to $\mathbf{f}_s$.
- $\mathbf{f}_{ext}^{i}$ represents the external force applied to the mass-point i.
- $\mathbf{f}_i$ is the total mass-point force ($=\mathbf{f}_{\text{int}}^{ij} + \mathbf{f}_{ext}^{i}$).
- $\mathbf{A}_i$ and $\mathbf{V}_i$ are the acceleration and velocity of the mass-point i, respectively.
- r and r_0 represent the spring length at the current state and the spring length at the rest state (between mass-points i and j, $i \neq j$), respectively.
- dt is the time step of the integration algorithm.
- Bold letters are used to represents vectors.

Figure 2.1 depicts the modeling structure of the mass-spring algorithm. As figure 2.1 suggests mass-spring systems consist of two forms of dynamics: spring dynamics and point dynamics (modeling is achieved by springs and points). The two forms of dynamics are not independent from each other but can be analyzed separately.

2.1.1 Spring Dynamics

A spring between two points and its detailed dynamics is given in figure 2.2. A typical spring consists of two components represented by spring stiffness and damping. Any changes at the two ends of the spring start a dynamic process. Therefore the internal forces in the MSS are generated by the springs because of the changes in their lengths and velocities. These changes are caused by external forces or by the movement of the mass-points. Together with the external forces, the internal forces will alter the rest positions of the mass-points. A mass-spring system eventually will reach a steady state, where these changes represent energy stored in the springs. This state represents the deformed shape of the original object. The changes and their effects on the system equations are examined in the following sections.

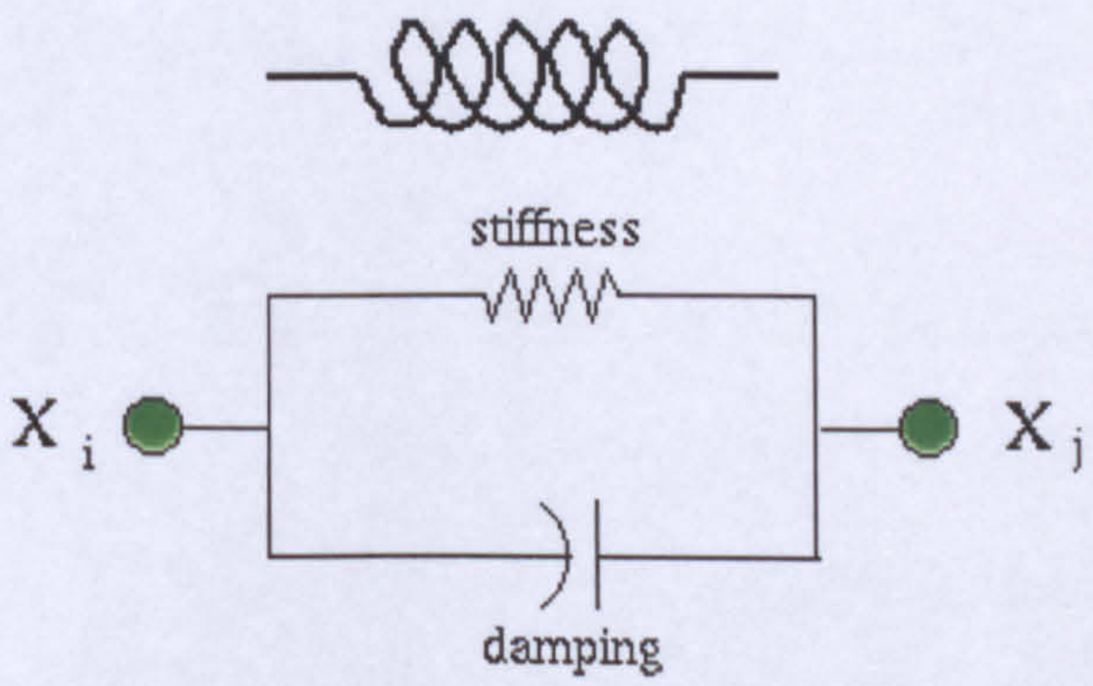
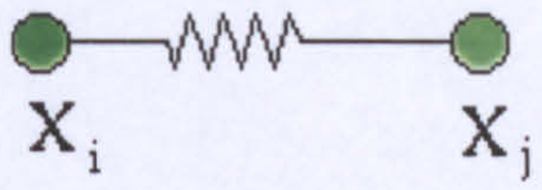


Figure 2.2 A typical representation of a mass-spring model between two points.

2.1.1.1 Length Change

It has been shown (Witkin et al. 1997, Baraff and Witkin 1999, 2001) that the spring force due to the spring length change is related to the changes in its length by the following formula:



$$\mathbf{F}_i = \mathrm{K}(|\mathbf{X}_j - \mathbf{X}_i| - r_0)\frac{(\mathbf{X}_j - \mathbf{X}_i)}{|\mathbf{X}_j - \mathbf{X}_i|} \tag{2.1}$$

where $|\mathbf{X}_j - \mathbf{X}_i|$ is the current length of the spring. The last part of the equation is a unit vector in the spring direction and can be defined as:

$$\mathbf{u}_{ij} = \mathbf{u}_K = \frac{(\mathbf{X}_j - \mathbf{X}_i)}{|\mathbf{X}_j - \mathbf{X}_i|}. \tag{2.2}$$

Equation (2.1) can now be represented as:

$$\mathbf{F}_i = K(r - r_0)\mathbf{u}_K \tag{2.3}$$

where r ($\left|\mathbf{X}_i - \mathbf{X}_j\right|$) is the current spring length.

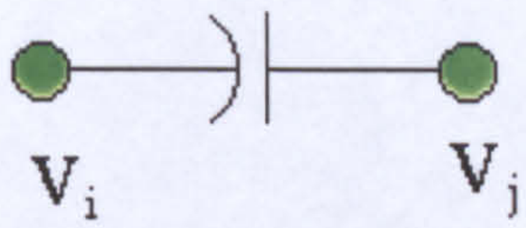
If we drop the use of the direction vector (unit vector) in the force equation 2.3, the spring force due to the length change can be expressed in a generic form:

$$f_K = K * dr \tag{2.4}$$

where dr is the change in length of the spring between its current length and its rest length. Due to this change in its length the spring offers a resistance related to its stiffness, therefore equation (2.4) represents the spring force due to spring stiffness. This formula is valid for linear cases in which stiffness is constant. It is also known experimentally that this coefficient is not constant but is a nonlinear function of spring length change, i.e. $K(dr)$.

2.1.1.2 Velocity Change

The spring force due to its velocity change across the spring is given in the literature (Nurnberger et al. 1998, Howlett and Hewitt 1998) as:



$$\mathbf{F}_i = D(\mathbf{V}_j - \mathbf{V}_i) \tag{2.5}$$

where $(\mathbf{V}_j - \mathbf{V}_i)$ is the velocity change of the spring. This equation can be represented in the form of a force function due to spring stiffness as in (equation 2.4). A unit vector representation for the spring direction of the spring velocity can be given as:

$$\mathbf{u}_{ij} = \mathbf{u}_D = \frac{(\mathbf{V}_j - \mathbf{V}_i)}{|\mathbf{V}_j - \mathbf{V}_i|}. \tag{2.6}$$

If we substitute equation 2.6 into equation 2.5 and define a new variable dv as the velocity change in magnitude ($|\mathbf{V}_j - \mathbf{V}_i|$) between the two ends of the spring, this new form of the spring force due to spring damping is given as:

$$f_D = D * dv. \tag{2.7}$$

Although more research is needed to determine the relationship between damping and spring velocity change, we assume that they are nonlinearly related, i.e. $D(dv)$. In almost all applications however constant damping coefficients are used.

2.1.1.3 Total Spring Force; Internal Force

There are two common formulas used in the literature to calculate the total spring force. Basically they sum the forces caused by the spring stiffness and damping

elements (Baraff and Witkin 1999, 2001). Using the simplified equations given in (2.4) and (2.7), the total spring force can be given as:

$$f_s = K * dr + D * dv . \tag{2.8}$$

A second formula, which is also commonly used, is given as:

$$f_s = K * dr + D * \frac{\mathbf{dr} \bullet \mathbf{dv}}{dr} \tag{2.9}$$

where the symbol ' • ' represents dot product of the two vectors.

The spring force is measured in the direction of the spring itself. Therefore, the formulas given by the equation (2.8) and (2.9) are multiplied by the relevant unit vector in the spring direction of each spring.

2.1.2 Point Dynamics

Each point in the MSS receives internal forces generated by the connected springs and the applied external forces. Point dynamics is then responsible for finding the acceleration, velocity and finally the new position of the mass-point, see figure 2.3.

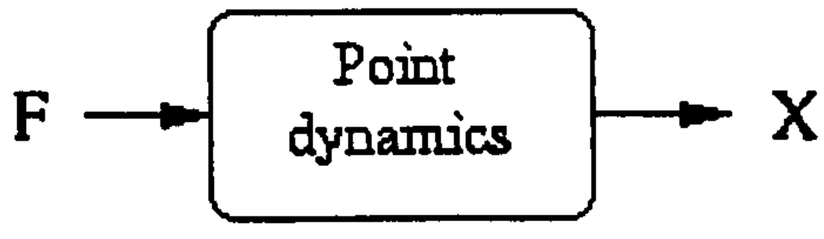


Figure 2.3 Point dynamics receives external and internal forces and finds the new positions of the mass-points.

Given the internal and external forces, there are well-established physical formulas for calculating point dynamics. Newton's law gives the relationship between the applied forces and the acceleration of each mass-points as:

$$f = ma . \tag{2.10}$$

Once the acceleration is known, there are well-known integration methods, proposed in the literature that can be used to determine the velocity and the new mass-point positions.

2.1.2.1 Explicit Euler Integration

One of the best-known and straightforward methods for calculating point dynamics is the explicit Euler method, whose integration scheme is given below (Baraff and Witkin 1999, 2001):

$$\mathbf{V}(k+1) = \mathbf{V}(k) + \frac{dt}{m}\mathbf{F}(k)$$

$$\mathbf{X}(k+1) = \mathbf{X}(k) + dt\mathbf{V}(k+1). \qquad (2.11)$$

This algorithm takes internal and external forces at the current time step (k) and calculates the velocity and position for the next time step $(k+1)$. Stability comes in the picture here because the velocity and position for the next time step is blindly calculated using the current values available. To achieve stability in this integration method the time interval denoted by dt must be chosen to be very small. This alone sometimes is not enough to ensure stability, which also depends on the choice of other system parameters. This technique is very easy to implement but because of its small time step requirement, it is often difficult to achieve real time performance by using it. In order to improve stability and allow large time steps the explicit Euler method is often modified.

2.1.2.2 Implicit Euler Integration

With the explicit method, given the total mass-point force and the initial conditions, we can determine the next mass-point position. If the time step is too large, a very big change in the position occurs because the internal forces are constant over that period.

With the implicit Euler integration method, however, the new mass-point position is consistent with the applied forces, because the next position is evaluated using the force at that instance. The algorithm of the implicit integration method is summarized below (Baraff and Witkin 1999, 2001):

$$\mathbf{V}(k+1) = \mathbf{V}(k) + \frac{dt}{m}\mathbf{F}(k+1)$$

$$\mathbf{X}(k+1) = \mathbf{X}(k) + dt\mathbf{V}(k+1). \qquad (2.12)$$

The only change between the two methods (equation 2.11 and 2.12) is the choice of the force function, which determines the velocity. The implicit method uses the force in the next time step to calculate the velocity in that step. This simple change ensures an unconditional stability thus allowing the choice of arbitrary time steps.

There are, of course, disadvantages to using this method. The force function $\mathbf{F}(k+1)$ can not be calculated at the current instance k. An approximation of this function can be used instead by taking the first-order derivative for its linear coefficients. The resulting solution involves a Hessian matrix of the system (Baraff and Witkin 1998). This means solving a $3n \times 3n$ matrix at each time steps. Use of this method then introduces an additional computational burden to the simulation system. As indicated in (Kang et al. 2000a, 2000b, 2001), implicit integration allows larger time steps to be used but the performance of such systems will still be far from interactive.

2.1.2.3 Approximated Implicit Method

There are two methods proposed to further speed up the integration method using implicit integration. Desburn at al. (1999) suggested a precomputed filtering method to approximate The Hessian matrix method. Their method, as indicated in (Kang et al. 2000a), is faster than the original implicit method but it fixes the stiffness matrix

whose values can not be changed dynamically. A more direct approximation of the implicit method is proposed by Kang et al. (2000 a, 2000 b, 2001).

The implicit integration algorithm given by equation (2.13) calculates velocity for the next time step. Their work produced a method that approximates the velocity of the next time step using the force function at the current time step:

$$\mathbf{V}(k+1) = \frac{\mathbf{F}(k) + dt^2 K \sum^{\mathbf{F}(k)dt} / (m+dt^2 Kn)}{m + dt^2 Kn} \tag{2.13}$$

where n represents the number of mass-points that are connected to each of the mass-points in the algorithm. Since the velocity for the next time step is known the positions of the mass-points can be calculated.

Their method does not require the solution of large linear systems, thus it is much faster compared to the original implicit method or the precomputed filtering method. The downside of this method is that it assumes a uniform spring constant. This may cause problems for applications when nonlinear coefficients are used. In addition, this algorithm introduces further approximations, therefore sacrificing more accuracy.

2.1.3 A Dynamic Model

The mass-spring structure studied above is shown to consists of two different types of dynamics; spring dynamics (including damping) and point (mass-point) dynamics. Spring dynamics are responsible for generating nonlinear spring forces (internal forces), driven by the spring stiffness and spring damping (see sections 2.1.1.1 and 2.1.1.2). Point dynamics use the internal forces from the connected springs and any applied external forces to determine the acceleration, velocity and finally the new

position of the point (see section 2.1.2). Using the explicit Euler integration method the dynamics are represented by a block box as shown in figure 2.4.

In this figure, the nonlinear function $\mathbf{f}_K$ represents the spring force (see equation 2.4) caused by the spring stiffness, similarly the nonlinear function $\mathbf{f}_D$ represents the damping or viscosity force (see equation 2.7). The summation of these two forces represented by $\mathbf{F}$ (see equations 2.8 and 2.9) is the total spring force exerted on the mass point from individual springs. This part of the block diagram is called spring-dynamics. The point-dynamics part of the diagram receives two inputs: an internal force, $\mathbf{F}$, from connected springs and an external force, $\mathbf{F}_{ext}$, from the environment. The combination of these two forces is called the total point force, which drives the acceleration, velocity, and position of that individual point (see equations 2.11 and 2.12).

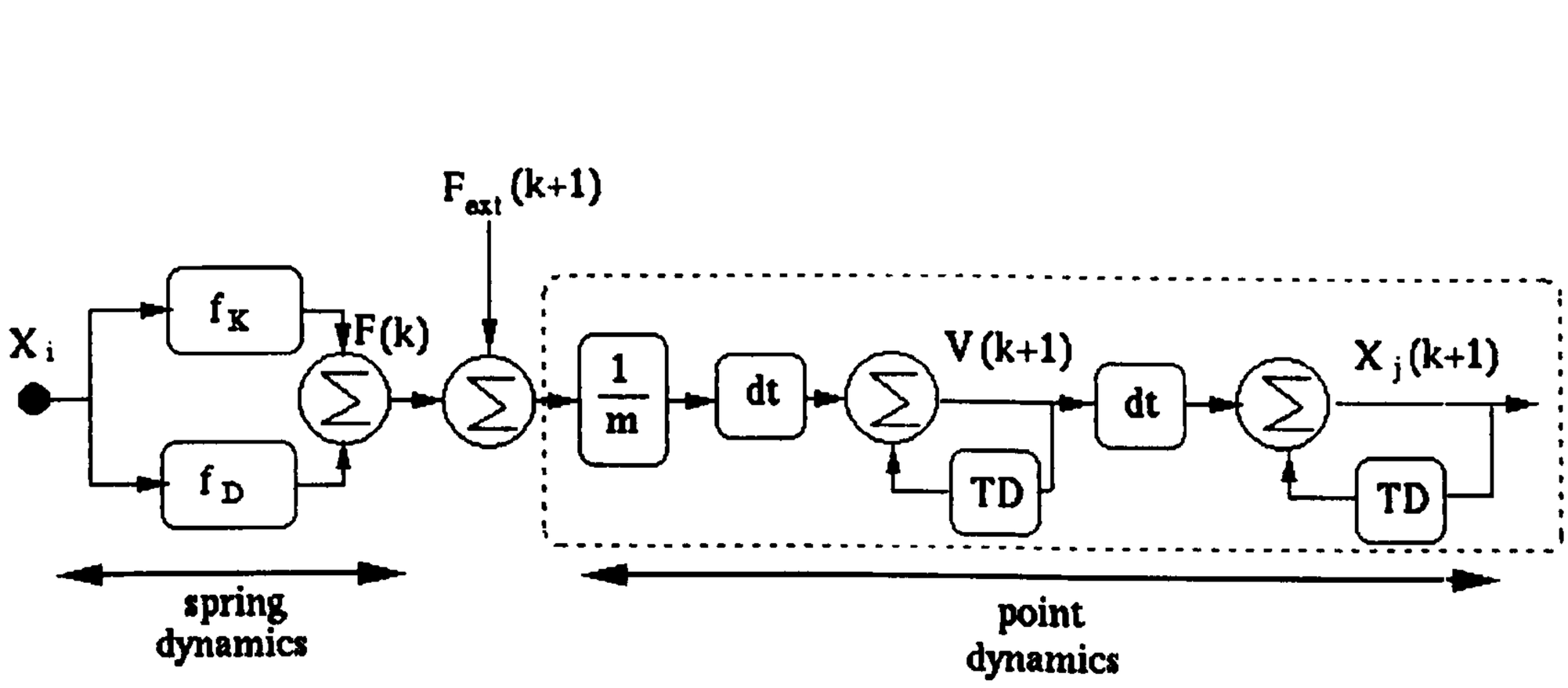


Figure 2.4 Mass-spring dynamics (TD: Time Delay).

The dynamic model given in figure 2.4 summarizes the mass-spring system. This model is used in chapter 5 to develop a neural network identification model. Here we provide some of the considerations that one has to take into account in order to implement mass-spring systems.

2.2 Implementation

There are several issues that have to be taken into consideration while implementing an MSS algorithm. Stability is the first consideration in any simulation project. There are other concerns specific to MSSs, such as excessive spring elongation and the choice of system parameters. These considerations are examined in detail in the following subsections.

2.2.1 Stability

Stability is the main concern in every simulation system. In mass-spring systems the stability is strictly dependent on the choice of system parameters and on the integration techniques used. These dependencies are given by the formulas (Kuhnapfel et al.1999):

$$D > \frac{dt * K}{2} - \frac{M}{dt}$$

$$dt < \frac{D + \sqrt{D^2 + 2MK}}{2} \tag{2.14}$$

One must choose system parameters according to the above formulas. In general the choice of parameters is based on the time step of the simulation system that is inversely proportional to the square root of its spring stiffness. An unconditional stability is quarantined by the implicit integration method.

2.2.2 System Coefficients

One of the biggest problems in implementing an MSS is the choice of the system parameters. In order to overcome the instability issue one may choose a different integration scheme or take equation (2.14) into account while determining the parameters. There is no easy solution in establishing the system coefficients. In most of the applications they are determined by trial and error, based on the visual

appearance of the simulation outcome. As an example, figure 2.5 shows a cloth model simulated using different spring stiffness values. These simulations are carried out over the same number of iterations. As clearly seen from the figure, the choice of system parameters dramatically effects the simulation outcome.

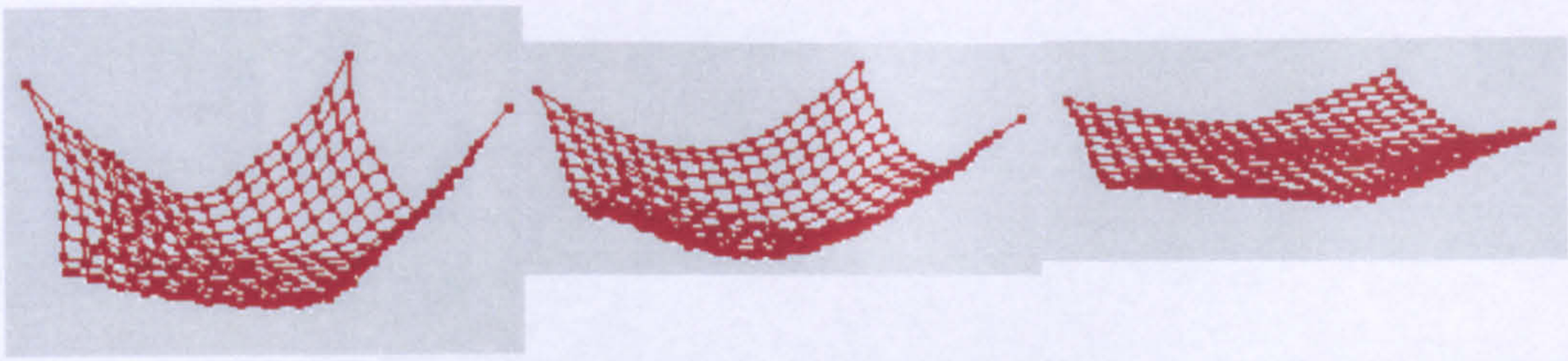


Figure 2.5 Cloth simulation using different spring stiffness values but the same values for other parameters under the same the initial conditions (i.e. the same external forces).

Figure 2.6 shows a typical stress-strain relationship for biological tissues (Kuhnopfel et al. 1999). As the figure suggest this relationship is nonlinear and can not be represented by simple constant values. Choosing the linear values themselves is difficult and determining complex nonlinear parameters may be impossible using conventional methods.

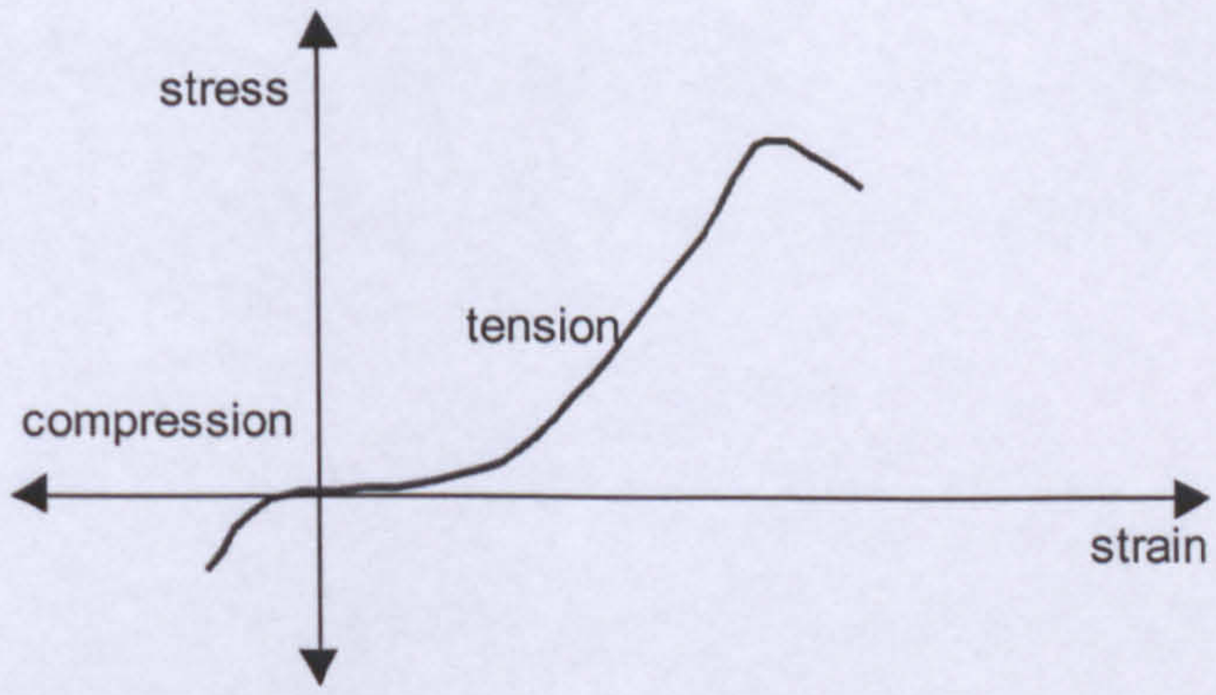


Figure 2.6 A stress-strain diagram for biological tissues.

2.2.3 The Choice of Integration Method

Implicit Euler integration gives a guaranteed stability with large time steps, therefore it is very desirable to use this method despite the fact that it does not still yield a real-time performance for most applications. Approximated implicit methods further increase the speed but also further decrease the accuracy of the simulation. The choice of the integration method is then depended on the nature of the application. In the simulation of cloth models implicit or approximated implicit Euler methods can be chosen because generally linear coefficients are used and accuracy is not the main concern. In other simulations, such as medical simulations, explicit Euler integration is often employed (Kuhnapfel et al. 1999, 2000, Cakmak and Kuhnapfel 2000). Other methods such as the Runge-Kutta method (Montgomery et al. 2002) may be used as an alternative to the explicit Euler method in terms of accuracy and performance but they are not appropriate for collision handling.

2.2.3 Excessive Spring Elongation

In mass-spring simulations some of the springs are being over-stretched and an undesirable appearance occurs. Super-elongated springs are found in places where the mass-points are constrained. Figure 2.7 illustrates this problem. A cloth model lying flat constrained from two of its corners is simulated using only gravitational force. Both figures have exactly the same parameters (mass, stiffness, damping and number of iteration). The left image of this figure shows that elongation occurs at the hanging points.

Provot (1995) proposed a simple and effective inverse dynamics process to resolve this problem. In his solution, springs that are excessively stretched are aligned in order to meet predefined conditions. Each spring is allowed to deform to a certain

percentage of its rest length. If this condition is violated his algorithm adjusts the spring length. As accepted by the author himself, this solution has no mathematical justification although it may produce plausible results in some applications. The right image of figure 2.7 implements Provot's algorithm for elongation correction. As seen from this image the elongation is corrected but the two images are considerably different from each other.

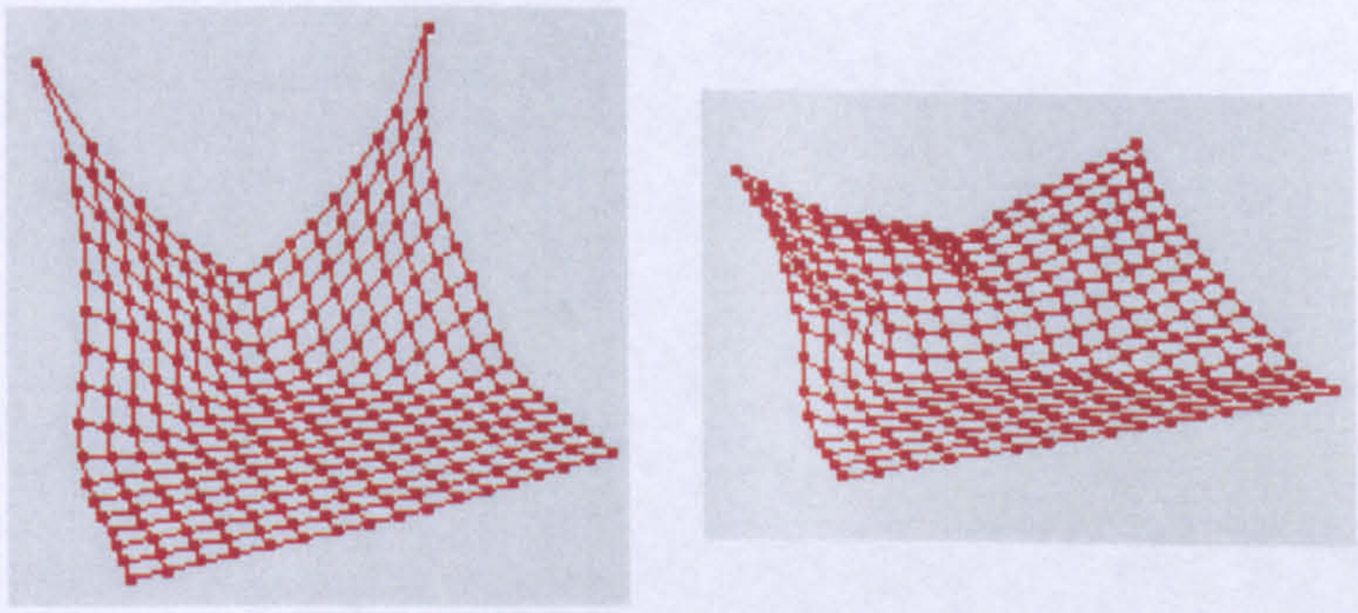


Figure 2.7. Spring elongation occurs at hanging points (left) and correction (right).

Provot's algorithm was further developed in (Kang et al. 2000 b). In the original proposal, the order in which the springs were to be adjusted was not given. This matter was investigated with an ordering algorithm, which produces a list where the springs are ordered according to their elongated lengths. Then these springs are adjusted based on this order. This problem was also addressed in (Bourguigron and Cani 2000), where the user can choose a region of interest and the mechanical properties of the material along a given number of axes. Since the mechanical properties and force directions are manually changed for the specified regions where spring elongation may occur, this work provides a means of controlling anisotropy.

None of the previous work provides a real solution to this problem. It is well known that a force/displacement curve is nonlinear as the stress-strain diagram shows (see

figure 2.6). This curve cannot be approximated by linear parameters correctly. This implies that spring parameters should be nonlinear. Even in linear cases, however, determining these parameters is not easy. Therefore in most applications such parameters are determined by trial and error based on the visual results of the simulation

2.3 A Solution: Nonlinear Parameters

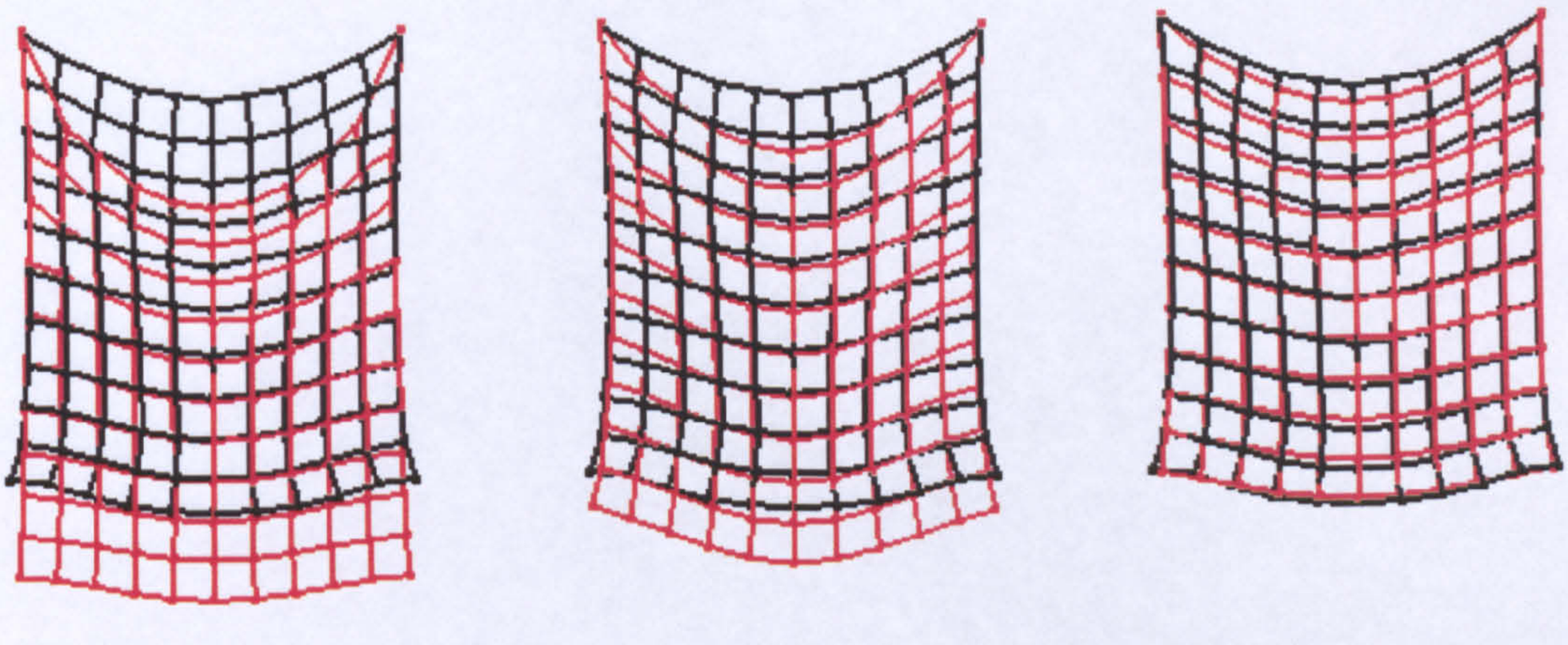
Stability of the simulation system and super-elongation of the springs all depend on the choice of system parameters as discussed above. If system parameters are properly chosen springs will not elongate excessively while system stability is maintained. More research, however, is needed in this area to determine the complex nonlinear relationships between spring stiffness and spring length change as well as between spring damping and spring velocity change. In chapter 4 we will propose a way of establishing these parameters by means of system identification.

To demonstrate this concept we give an example here. Suppose that the spring stiffness is nonlinearly related to spring length change by the following formula:

$$K(dr) = k_0(1 + k_1 * dr^2). \tag{2.15}$$

where the parameters k_0 and k_1 are constants. For simplicity we now use a constant coefficient for damping and formula (2.15) in the following example. We also implemented Provot's algorithm for elongation correction. The results are shown in figure 2.8 for the same number of iterations. The image in black is obtained using the spring elongation correction algorithm while the image in red is obtained using a nonlinear coefficient. Figure 2.8 suggests that the spring coefficients are nonlinear and that if they are established correctly the difficulties discussed in sections 2.2.1-

2.2.3 are avoided. In the following section we will discuss another implementation issue, collision detection which is an essential part of any simulation algorithm.



(a) (b) (c)

Figure 2.8 Cloth simulation with different stiffness functions. (a) Uses constant coefficient, (b) and (c) use the function defined by equation 2.15 with different values for k_0 and k_1.

2.4 Collision Analysis

Collision detection and handling is the computationally expensive phase of deformable object simulation algorithms. This is because the algorithm searches for any possible collisions and responds to this at each time step of the integration. However, collision detection is an essential part of dynamic simulations where objects interact with each other. Therefore this subject is the focus of many researchers who have already made significant progress and provided efficient collision detection algorithms (Yang and Thalmann 1993, Provot 1997, Wagner et al. 2002, Algorithms 2004). There are two possible ways in which collision may take place: a collision with another objects in the simulation environment (collision with the environment) and a collision of the object with itself (self-collision).

2.4.1 Collision with the Environment

Most simulation applications require more than one object in the scene and interactions between them. In surgical simulations, for example, organ models are manipulated with surgical tools as well as interacting with the surrounding areas. There are two types of collision possible between two objects.

2.4.1.1 Mass-Point Polygon Collision

This type of collision occurs when one of the mass-point in the 3D mesh goes through a polygon (triangle). If the path of the mass-point is considered as a ray, then the collision detection becomes a ray-polygon intersection. Ray-polygon intersection algorithms are well studied (Badouel 1990, Mooller and Trumbore 1997) and we provide a detailed analysis of a ray-polygon intersection algorithm here. This algorithm will be used in chapter 3 in order to generate prismatic volume elements representing different tissue layers.

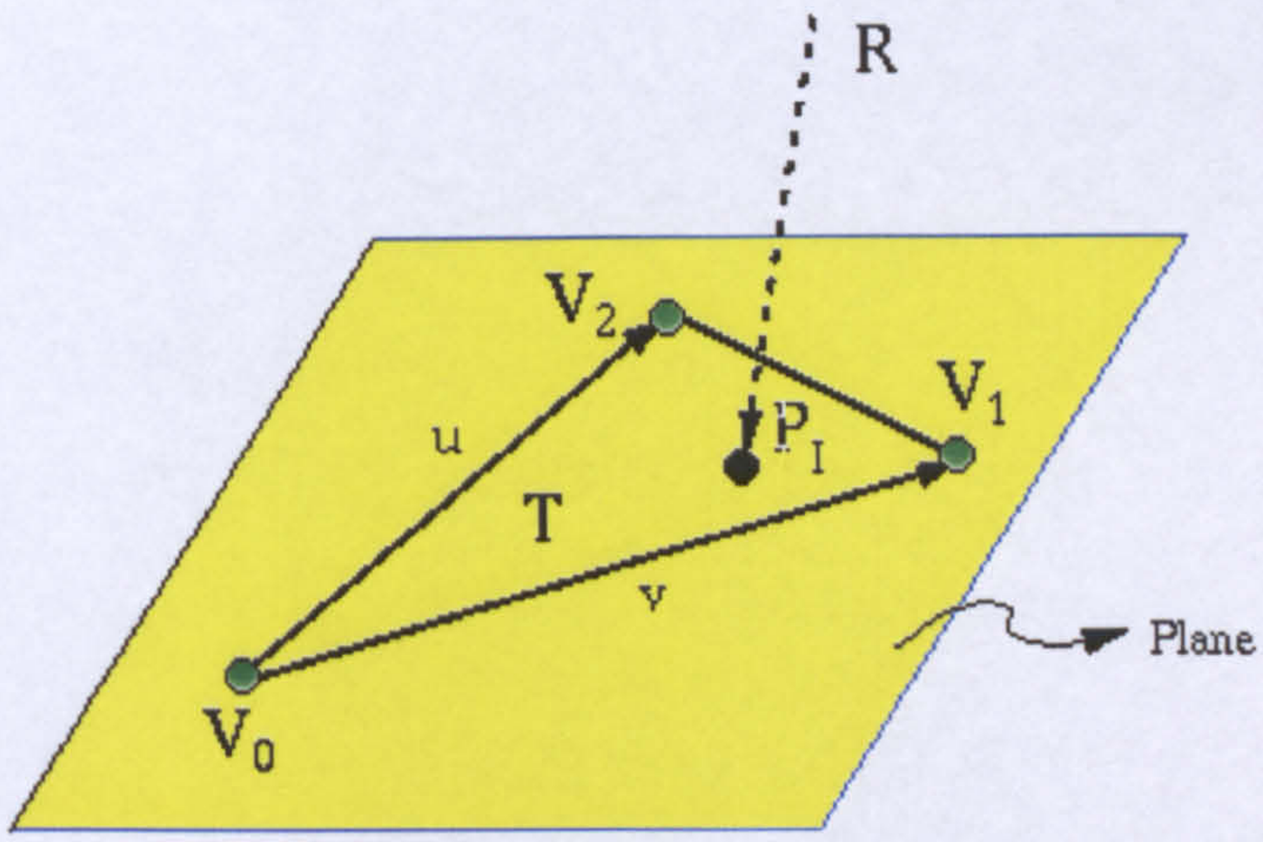


Figure 2.9 Ray-triangle intersection.

The algorithm we employed here uses direct 3D computations to determine inclusion, thus avoiding projections into a 2D coordinate plane (O'Rourke 1998). Figure 2.9 depicts a ray $\mathbf{R}$ intersecting a triangle T, defined by vertices $\mathbf{V}_0$, $\mathbf{V}_1$ and $\mathbf{V}_2$. The

ray-triangle intersection algorithm starts by determining whether the ray and the plane (of the triangle) intersect. If they do not intersect, this means that the ray also does not intersect with the triangle itself. If they do intersect, we need to find their point of intersection.

The plane equation in terms of the vertices of the triangle is given as:

$$\begin{aligned}\mathbf{V}(s,t) &= \mathbf{V}_0 + s(\mathbf{V}_1 - \mathbf{V}_0) + t(\mathbf{V}_2 - \mathbf{V}_0) \\ &= \mathbf{V}_0 + s\mathbf{u} + t\mathbf{v} \qquad (2.16)\end{aligned}$$

where s and t are parameter values and $\mathbf{u} = (\mathbf{V}_1 - \mathbf{V}_0)$ and $\mathbf{v} = (\mathbf{V}_2 - \mathbf{V}_0)$ are the edge vectors of the triangle. The intersection point is in the triangle if $s >= 0, t >= 0$ and $s + t <= 1$. The intersection point is on an edge of the triangle if one of the following conditions is true: $s = 0$, $t = 0$ and $s + t = 1$. In such a case, each condition corresponds to one edge.

The problem is then reduced to finding the coordinates of the intersection points and checking the above inequalities to determine if the intersection is inside the triangle. A detailed description of the above algorithm can be found in (Sunday 2001), where the parameter values are computed as:

$$s = \frac{(\mathbf{u} \bullet \mathbf{v})(\mathbf{w} \bullet \mathbf{v}) - (\mathbf{v} \bullet \mathbf{v})(\mathbf{w} \bullet \mathbf{u})}{(\mathbf{u} \bullet \mathbf{v})^2 - (\mathbf{u} \bullet \mathbf{u})(\mathbf{v} \bullet \mathbf{v})}$$

$$t = \frac{(\mathbf{u} \bullet \mathbf{v})(\mathbf{w} \bullet \mathbf{u}) - (\mathbf{u} \bullet \mathbf{u})(\mathbf{w} \bullet \mathbf{v})}{(\mathbf{u} \bullet \mathbf{v})^2 - (\mathbf{u} \bullet \mathbf{u})(\mathbf{v} \bullet \mathbf{v})} \qquad (2.17)$$

where $\mathbf{w} = \mathbf{P}_I - \mathbf{V}_0$ ($\mathbf{P}_I$ is the point of intersection of the ray and triangle) and '$\bullet$' represents dot product of two vector. The intersection point is then found as

$$\mathbf{P}_I = \mathbf{V}(s,t). \qquad (2.18)$$

2.4.1.2 Edge-Edge Collision

Edges of two polygons can collide with each other without having any mass-point collision involved. Again this is a well-studied problem. Here we will provide the underlying concept of the edge-edge collision algorithm and the refer reader to various sources (Moore and Wilhelm 1988, Volino and Thalmann 1995).

Such an algorithm consists of three stages. First, collisions are considered between two polyhedra, one of which may have vertices inside the other. Determining if such a collision is accomplished by finding the dot product between the two normals of the faces of the first polyhedron and vertices of the second polyhedron. If this dot product is negative the vertex of the first polyhedron is inside the volume of the second polyhedron. In this case the algorithm proceeds to the second stage which determines edge collisions. The singed perpendicular distances of the two end points of an edge of the first polyhedron from the plane of the face of the second polyhedron are determined. If the two distances differ in sign, then the edge is said to intersect the face of the polyhedron. The last stage of the algorithm deals with a situation where the faces of the two polyhedra are perfectly aligned. The centroid of the face of one of the polyhedra is computed and the vertex-polygon intersection is determined (the first stage of the algorithm). If this centroid is found to be inside the volume of the second polyhedron then a collision occurs.

2.4.2 Self Collision

The underlying idea here is that each mass-point has to be checked against its neighborhood and other mass-points. Since adjacent mass-points can not pass through or collide with the surface of the object to which they belong, self-collision detection algorithms prevent these situations from happening.

The implementation of such an algorithm is very simple; a virtual sphere is placed around each mass-point and points are prevented from getting any closer to each other than the radius of this sphere. If there is a collision, the colliding mass-point is pushed away towards to the surface of this sphere and its position and velocity are adjusted accordingly. The reader is referred to the work of Volino and Thalmann (1994, 1995) and Lafluer et al. (1991). In figure 2.10, we show the original image of a piece of cloth hanging from its two corners, (left image). In the middle image we have moved one of its corners closer to the other and simulated the cloth behavior. This image shows the output of the simulation algorithm ignoring self-collisions. As is clearly seen from the figure, some of the mass-points have passed through the surface of the cloth, which looks unrealistic. The right image employs a self-collision test and therefore looks more realistic.

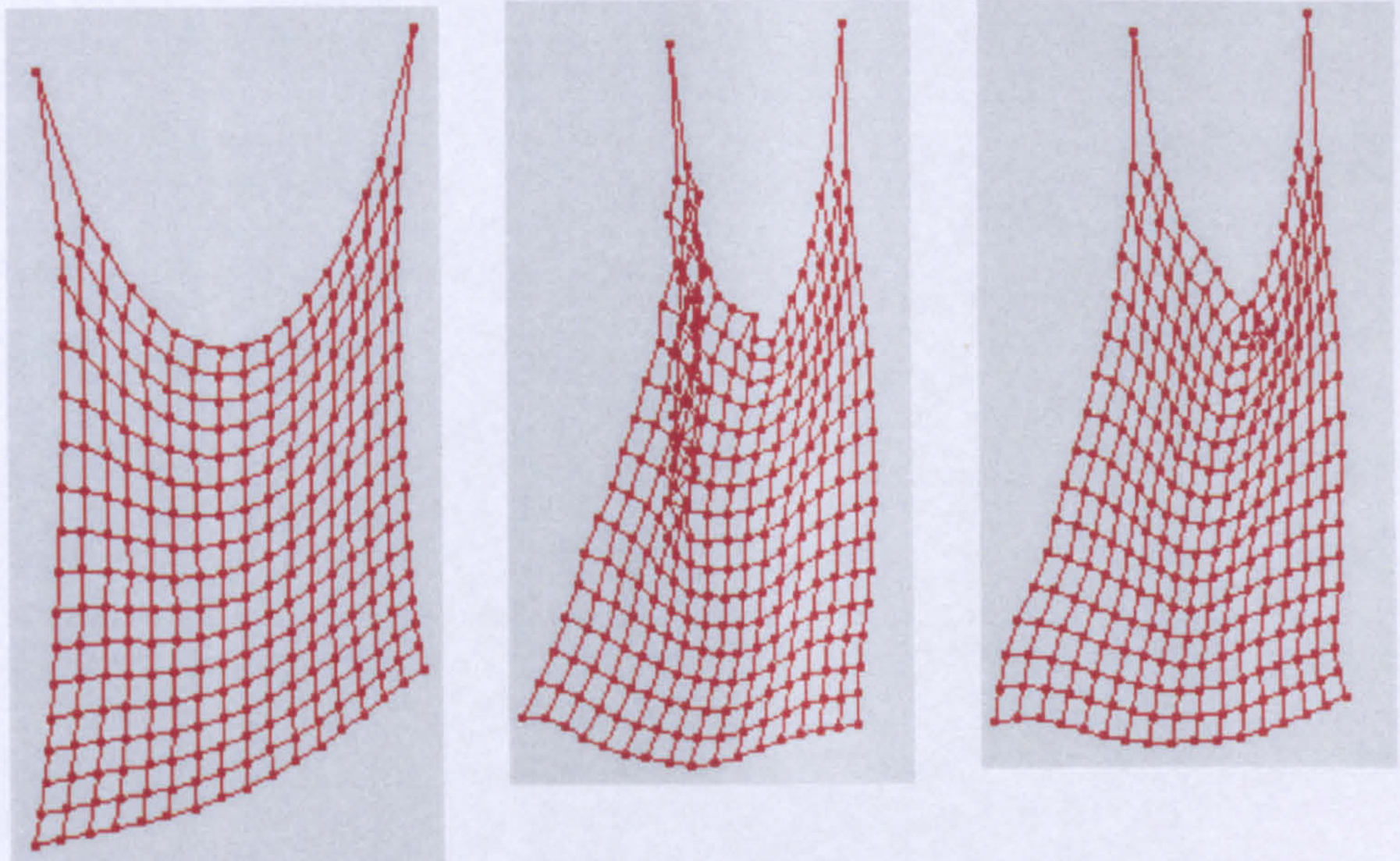


Figure 2.10 Hanging cloth simulation with and without the self collision test.

2.4.3 Collision Time

If there is a collision we need to determine its exact time thus allowing the algorithm to respond to it. This subject is explored in (Kang et al. 2000 b) where the authors

provide a collision time calculation for mass-point triangle collisions and edge-edge collisions. The collision time is computed using the following equations:

$$\begin{aligned} & t \in [t_0, t_0 + dt] \\ & u, v \in [0,1], u + v \le 1 \\ & \mathbf{AP}(t) = u\mathbf{AB}(t) + v\mathbf{AC}(t) \\ & \mathbf{AP}(t).(\mathbf{AB}(t) \times \mathbf{AC}(t)) = 0 \end{aligned} \tag{2.19}$$

where $\mathbf{P}$ is a point and $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are vertices of a triangle. The time calculated using above equation gives the actual collision time in the corresponding time interval. The algorithm needs to respond to the collision at that time. Alternatively, the algorithm sets a threshold distance d_t and detects the collisions. If the distance between the deformable object and the other objects are less then this defined distance the collision time is adjusted as:

$$t' = t - \frac{d_t}{\mathbf{V}} \tag{2.20}$$

where t' is the adjusted collision time.

2.4.4 Collision Response

When two objects collide with each other there will be friction and energy dissipation. If the collision is elastic, there will not be any energy dissipation. In the opposite case, where the total energy is dissipated during an inelastic collision, the collision algorithm needs to adjust the velocity of the mass-points just before the collision time. The velocity is adjusted by the following formula (Kang et al. 2000 b):

$$\begin{aligned} & if \mid \mathbf{V}_T \mid \ge K_f \mid \mathbf{V}_N \mid \\ & \mathbf{V}' = \mathbf{V}_T - K_f \mid \mathbf{V}_N \mid \frac{\mathbf{V}_T}{\mid \mathbf{V}_T \mid} - K_d \mathbf{V}_N \\ & if \mid \mathbf{V}_T \mid < K_f \mid \mathbf{V}_N \mid \\ & \mathbf{V}' = -K_d \mathbf{V}_N \end{aligned} \tag{2.21}$$

where $\mathbf{V}_N = (\mathbf{V}.\mathbf{N})\mathbf{N}$ is the normal component of the velocity before the collision and $\mathbf{V}_T = \mathbf{V} - \mathbf{V}_N$ is the tangential component. The coefficients K_f and K_d represent friction and dissipation constants respectively.

2.5 External Forces

There are various types of external forces acting on each mass-point deforming deformable objects. The most common of these are caused by gravity, air resistance (fluid) and interaction with various tools. The gravitational force is given as:

$$f_g = mg \tag{2.22}$$

where g represents the acceleration constant given as 9.81 m/sec^2.

When air is in contact with the simulated surface it exerts a force on each of its mass-points. In general an object can be considered as if it is inside a fluid that has a certain velocity. Ignoring the lift force of the fluid, the magnitude of the drag force is given as (Kang et al. 2000 b):

$$|\mathbf{f}_{drag}| = \frac{1}{2} C_{drag} \rho |\mathbf{V}|^2 S \sin\theta \tag{2.23}$$

where C_{drag} is the drag force coefficient, ρ is the density of the fluid, $\mathbf{V}$ is the relative velocity of the object, S is the area of the object surface and θ is the angle between object's velocity and surface vectors.

The force applied to the deformable object from external tools, such as surgical instruments, is calculated using the depth information. The actual force is not known exactly, but experimental approximations can be used. The determination of such a force is completely application dependant.

2.6 Application

A simulation algorithm is developed based on the concept examined in this chapter. The algorithm is then used in cloth simulations and the simulation of soft tissue deformations. The simple structure shown in figure 2.11 is used to develop cloth models. Three types of springs are used in this model. First, all mass-points are connected to each other by structural springs (shown as black). They are also connected by shear springs, which are diagonal springs (shown in blue). In order to simulate bending and wrinkles a spring type called bending spring is used (shown in red).

A cloth model is obtained using 289 (17×17) mass-points. The model consists of 544 structural springs, 512 shear and 510 bending springs. In the simulation of the cloth model a collision algorithm is used but no elongation correction algorithm is used. The simulation was run for 300 iterations and the result is shown in figure 2.12. A second example simulates soft tissue deformation. A face model was generated (examined in detail in chapter 3) and the model interacted with a probe. The model consists of 2437 mass-points and 6722 springs. The simulation output is shown in figure 2.13.

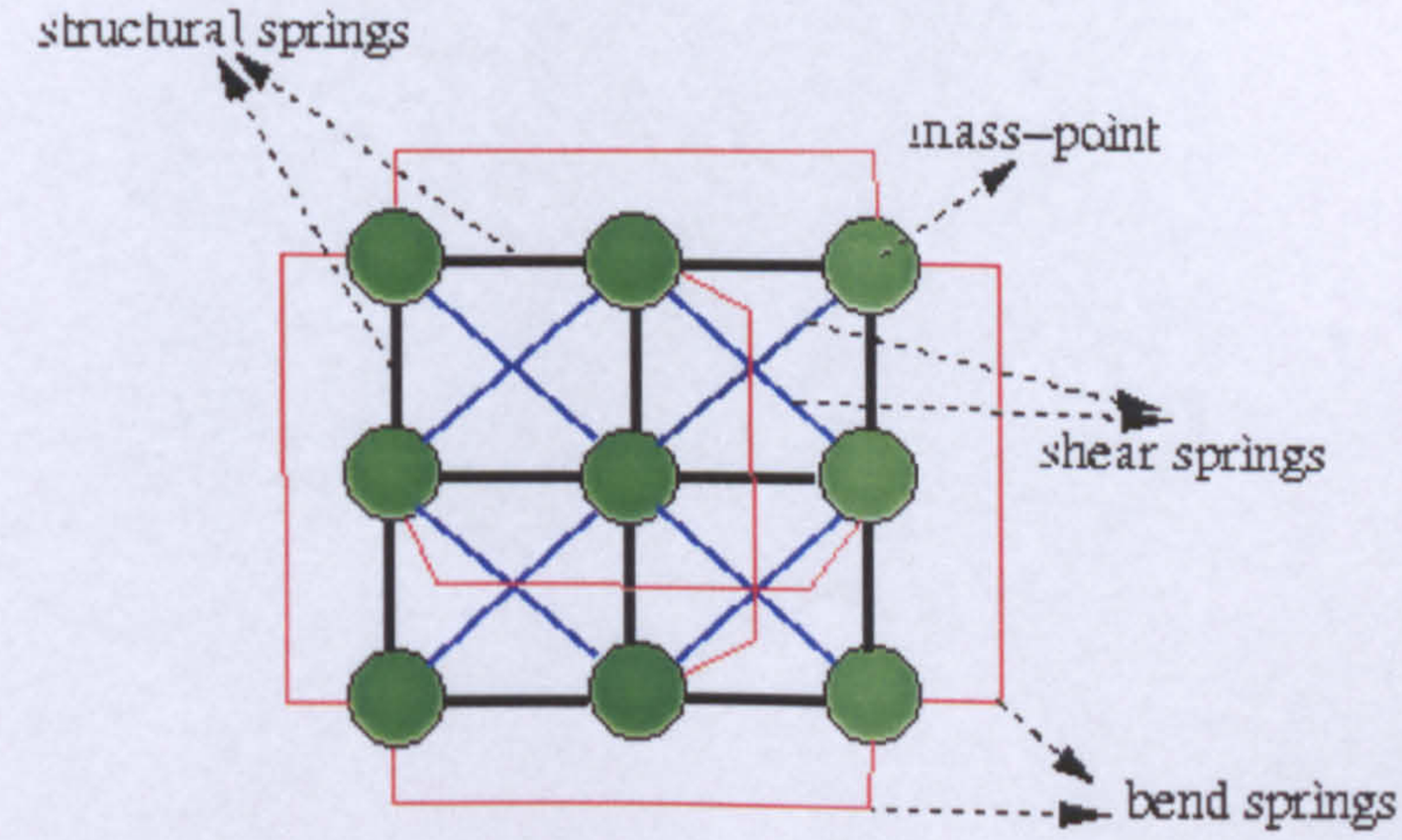


Figure 2.11 Spring structure is used to create cloth models.

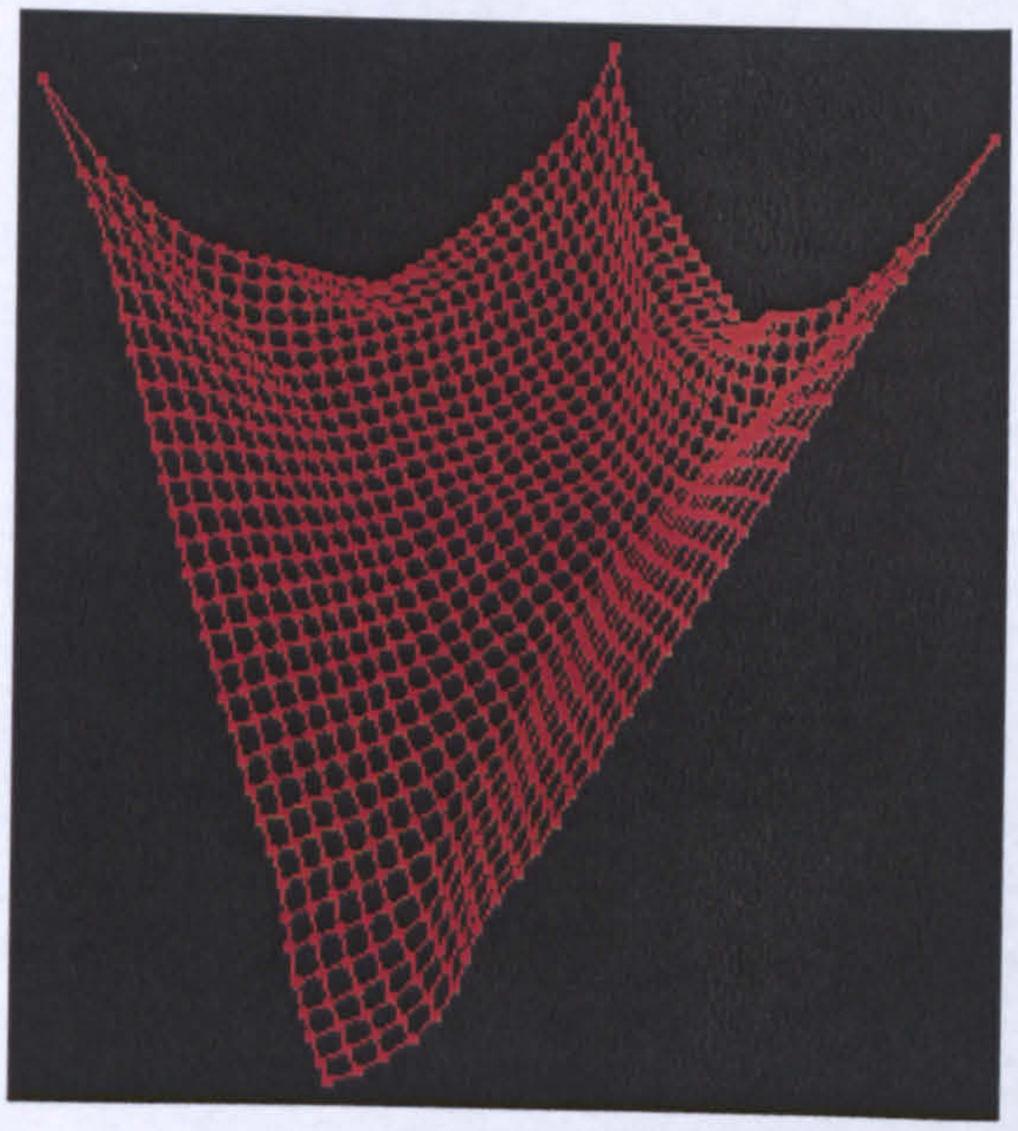


Figure 2.12 Cloth simulation.

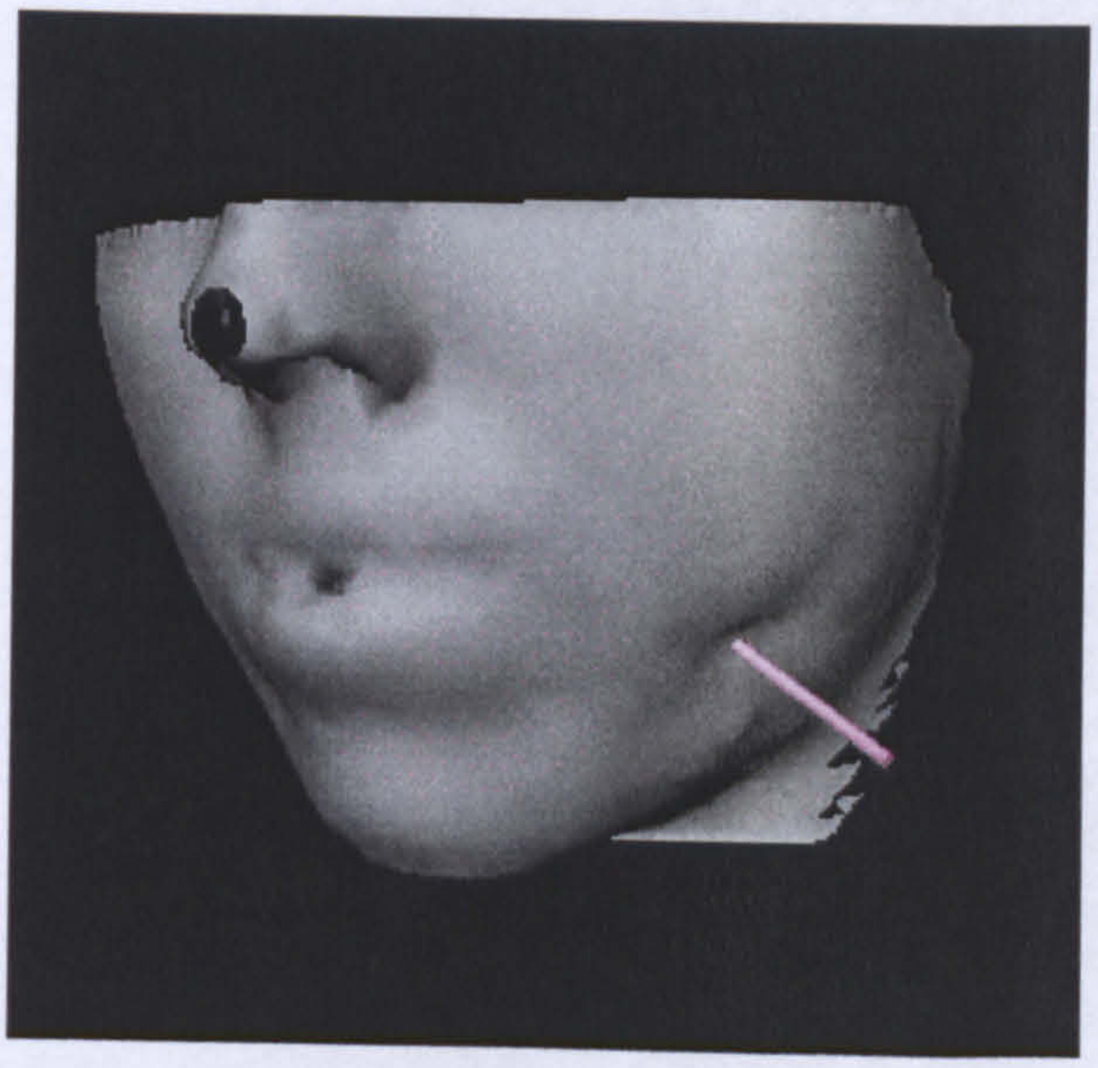


Figure 2.13. Soft tissue interacting with a surgical tool.

2.7 Summary

In this chapter we have examined a mass-spring algorithm in great detail, including integration techniques, collision analysis and spring elongation. In addition we have outlined the importance of simulation parameters. The coefficients used in the mass-spring algorithm have an enormous effect on the simulation outcome as well as the stability of the simulation. The concept was demonstrated by examples. A major application of such an algorithm will be given in chapter 3 where we examine a surgery simulation. Specifically, a mass-spring algorithm is applied to a craniofacial surgery simulation.

CHAPTER 3

SIMULATION OF SOFT-TISSUE DEFORMATIONS

Surgery simulator systems will become increasingly popular, as flight simulators are now for pilots. This is because they have proven invaluable in assisting surgical training and planning during pre-operative and intra-operative procedures. One of the core parts of the surgery simulator systems is the modeling and simulation of soft tissue deformations. There are numerous methods available, as discussed in chapter 1, for such a task. In chapter 2 we have studied mass-spring systems to be used in our simulations to predict soft tissue deformations. We now examine the process of obtaining a suitable model from medical image data for the simulation algorithm. This model will then be used in the simulation of a craniofacial surgery, where the lower jaw is cut and aligned with the upper jaw.

Simulating facial tissue deformations due to the underlying bone realignment requires several stages of preparation. These stages include reading and visualization of the medical image data. At the second stage, measurement and manipulation of the data takes place. At this stage, experts decide the required surgical operations and procedures. Once surgery has been planned, the medical data is processed by a

number of algorithms in order to generate a polygonal model. A mass-spring algorithm then assigns springs and mass-points to the polygonal model and finally runs the simulation.

Simulating craniofacial surgery is one of the most challenging applications in deformable object simulation for the following reasons. It is very important that the outcome of the simulation is as accurate as possible. This is a challenge because facial tissue consists of several different tissue layers with different deformation characteristics. It is also very important that the simulation concludes as quickly as possible. Since very high numbers of springs are used in the polygonal model to approximate the facial appearance, simulation takes a long time to perform. Polygonal models must include different tissue layers as well as their connections with the skull. Finding connections may introduce problems because of the skull's unique structure. Finally, the simulation algorithm must carefully choose simulation parameters that capture the tissue characteristics. Each of these concerns is addressed while the model is created during the simulation.

This chapter deals with every aspect of model generation, including the reading of the medical image data and medical image processing, such as filtering, segmentation and measurements (see section 3.2). Our main contributions here, are the generation of a realistic facial tissue model for the jaw area and the simulation of facial tissue deformation. In section 3.3, we will generate polygonal models of skin and bone surfaces. By connecting vertices on the skin surface to bone surface, we will obtain prism volume elements representing different tissue layers (see section 3.4). This model takes into account the curves on the facial surface and the holes on the bone structure as well as different tissue thickness around the jaw. We will then use this

model to implement the mass-spring systems algorithm examined in chapter 2, in order to simulate the facial tissue deformation. Results are presented in section 3.5 where we also use our model to generate facial animations.

3.1 Surgery Simulation Systems

The standard procedure for surgery simulation starts with the acquisition of 3D medical images of patients. Images obtained using different imaging modalities need to be transformed into a common spatial alignment. This process is called image registration and allows the comparison and interpretation of different image data. The volume of interest is then extracted from the original 3D image using various segmentation methods. Segmented volumes are then represented by geometric models described by a set of triangles. This is achieved using isosurface generation algorithms. The output of the triangulation algorithm is often fed into a decimation process in order to allow real-time manipulation and fast rendering. The resulting geometric model can then be used in simulations where surgical operations are possible. Simulation systems may include hardware components allowing user interaction. Figure 3.1 shows the outline of a basic surgery simulation system.

Figure 3.1 suggests that surgical simulation systems involve several disciplines including data acquisition, segmentation, surface generation, simulation and visualization. Interaction of the user through a virtual reality environment with haptic feedback (KISMET, PHANToM, Bar-Cohen and Breazeal 2003) is not addressed in this work. If there are more than one datum from different image modalities or from the same image modality but at different time instances, these images need to be registered. There are numerous methods for image registration. One is based on a set of landmark points, which can be externally placed or anatomical landmarks can be

used. Another image registration method uses geometric image features such as surfaces and finally third a method uses the intensity of images for different tissue types. This subject is outside the scope of this thesis and we refer the reader to the following references (Arun et al. 1987, Maes et al. 1997, Maintz and Viergever 1998, Maurer et al. 1996, Van den Elsen et al. 1995, Woods et al. 1993). In our work, we use a visualization package called 3DVIEWNIX developed by the University of Pennsylvania (3Dviewnix). This package is not as sophisticated as other commercially available programs but it enables us to read, manipulate and take measurements from the medical data. Other necessary programs were developed to work with 3DVIEWNIX. The relevant parts of the diagram are briefly examined in the following sections.

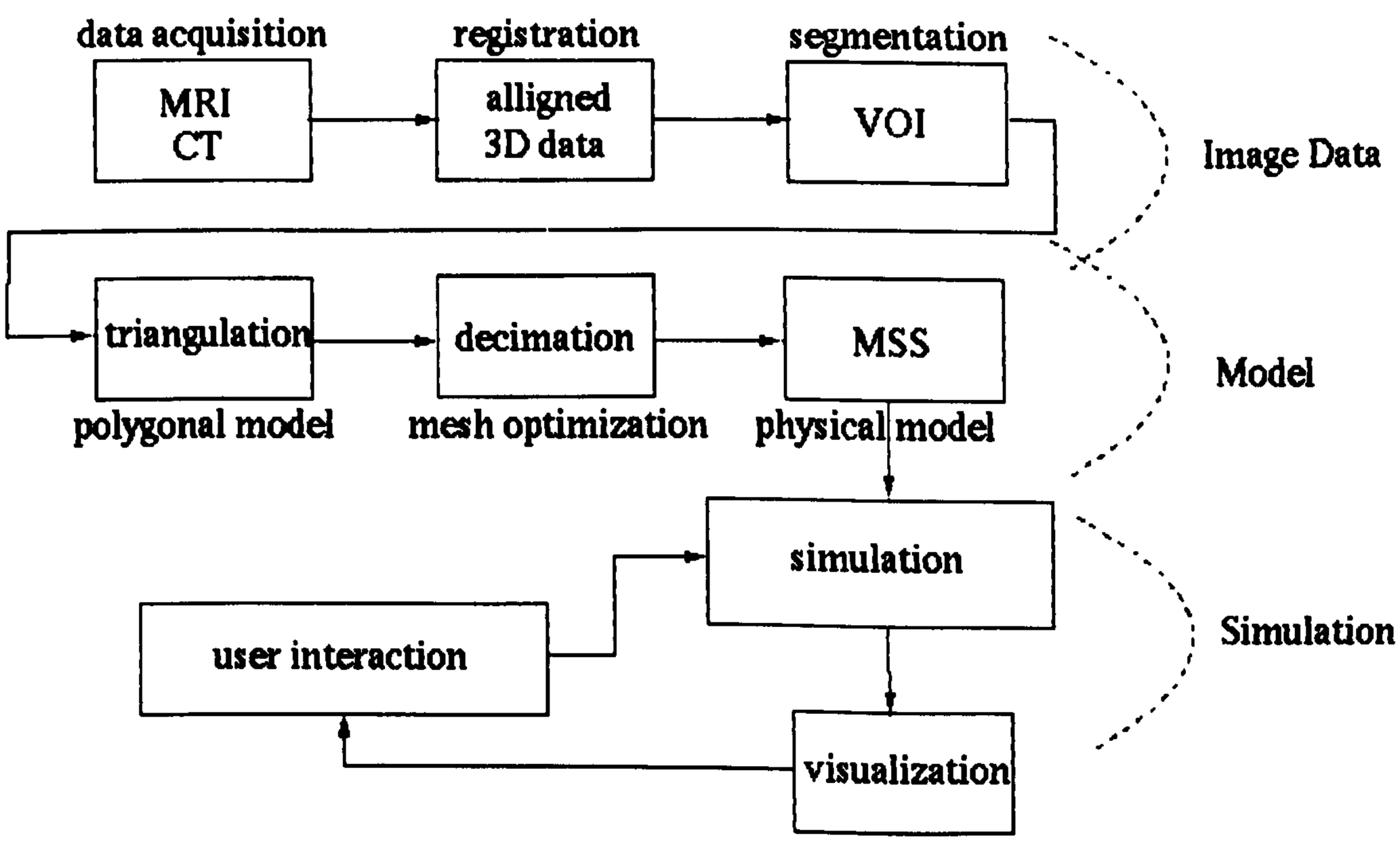


Figure 3.1 A Basic surgery simulation system.

3.2 Medical Image Data

Medical images are often presented as pictures, but they differ significantly from conventional photographs, which only captures how the object reflects and transmits light. In the field of medical imaging on the other hand, an MRI image (for example) is computed numerically. The object (body) is placed in a strong magnetic field and responses of the different tissue types to radio-frequency pulses are measured. Different responses representing the various body parts such as muscle or bone then form the medical image. In the following sections we briefly explain how medical imaging is obtained in different forms.

3.2.1 Data Acquisition

The two most common techniques used in acquiring 3D medical images are Computerized Topography (CT) and Magnetic Resonance Imaging (MRI). There are other imaging modalities (IMAGE), including Possition Emission Topography (PET), Single Photon Emission Computed Topography (SPECT), and 3D Ultrasonic Imaging. The medical images obtained using these technologies provide detailed information about the anatomical structures of patients. This information will be complementary because the modalities are based on different physical phenomena. For example MR imaging, which is based on nuclear magnetic resonance, is very suitable for soft tissue analysis. On the other hand CT, which is based on X-rays, successfully detects bone structures.

In this study we use data provided by Poole Hospital (PooleHospital). The head image is 512*512 pixels in size and consists of 22 slices in the DICOM file format (DICOM). Slice spacing is 4.91 millimeters and maximum density is 4095.00 while minimum density is 0.00. The pixel size is 0.39 by 0.39 millimeters. The image

captures the whole head of a patient. Data is read using the 3DVIEWNIX software and then it is visualized.

CT and MRI imaging have become very popular because they create cross-sectional sliced images, which can be stacked to form volume data showing the internal structures as well as outer surface of a body. In the following sections we briefly explain how 3D volume data is formed from slices generated by the MRI or CT imaging techniques and how pre-operative procedures such as measurement and cutting are performed.

3.2.1.1 Slice Data

Imaging modalities produce a series of cross-sectional slice images representing specific body parts or the whole body. Currently, MRI technology is able to offer sub-millimeter accuracy. The slice image consists of a number of pixels with various densities representing different tissue types. Each scan is called a slice image. Examples of such slice images, representing a human head, are shown in figure 3.2. Only few selected slices of the head are shown in this figure.

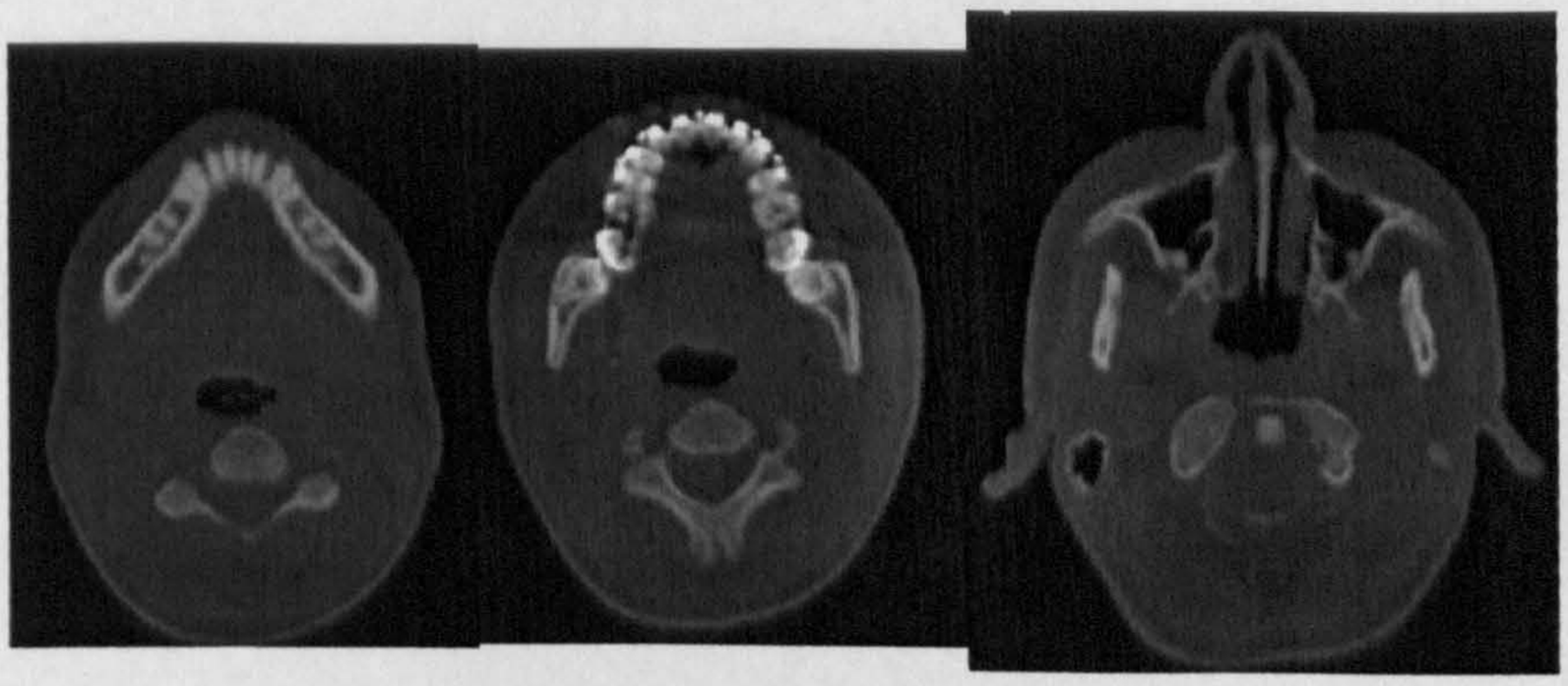


Figure 3.2 MRI slice images of a human head.

Image slices provide valuable information for diagnosis and planning. As can be seen from figure 3.2, it is possible to identify specific parts of the head and brain from the sliced images. In addition, some measurements can be taken from the slices. Slices are also be used in the process of manual image segmentation. More importantly, continuous 2D slices can be stacked one on top of the other to form 3D volume data.

3.2.1.2 Volume Data

When two slices of the same image are put one on top of the other, the pixels in the 2D slices can be interpolated to create cubic voxels by using the four corners from each pixel. Therefore a total of eight corners from the two pixels form a voxel. This process is illustrated in the figure 3.3 where two consecutive slices numbered with k and k+1 are shown. A voxel $V(x, y, z)$ is three dimensional, where the first two dimensions refer to the pixel location and the third dimension refers to the slice number.

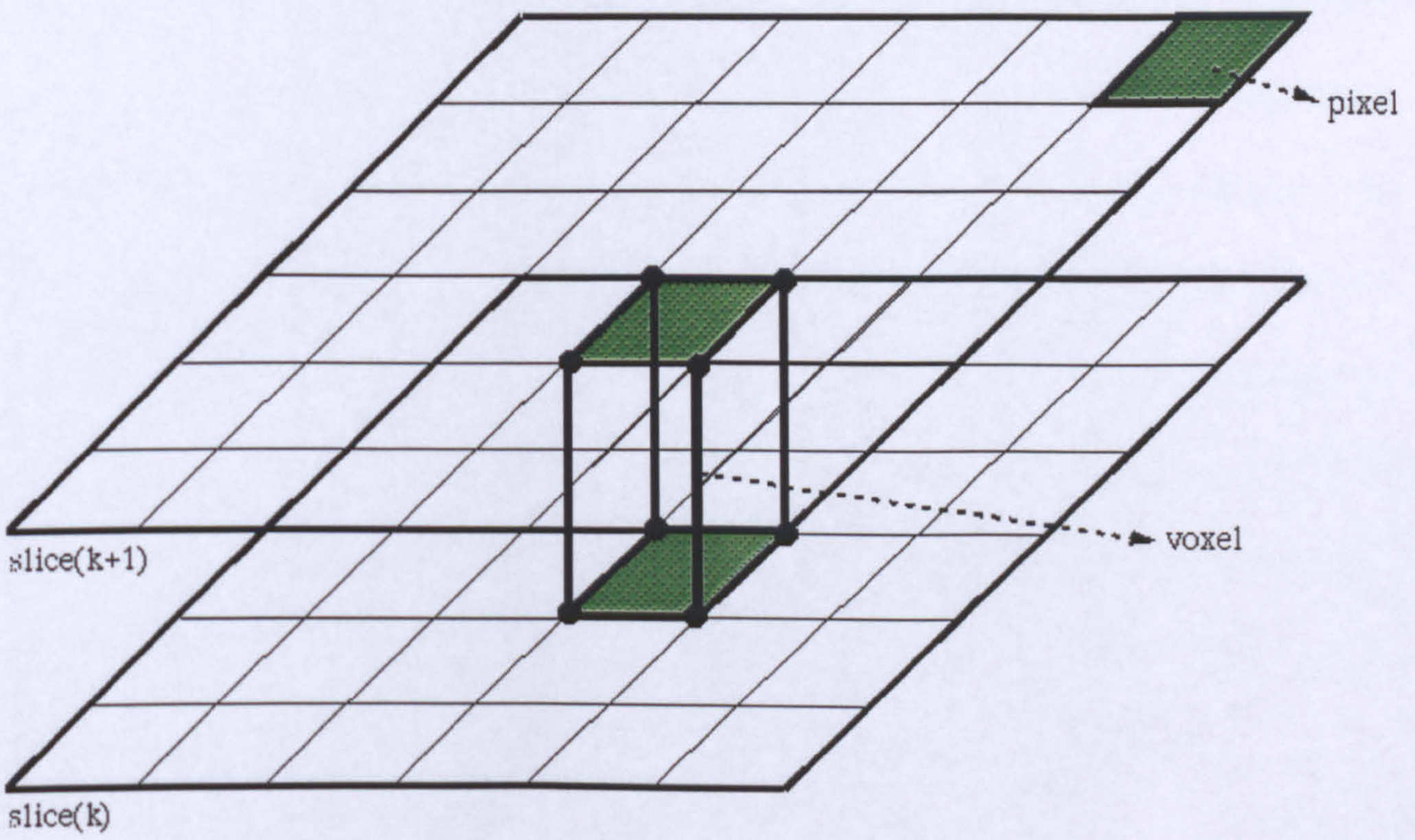


Figure 3.3 A voxel is formed between two consecutive slices.

Volume data provides more opportunities in the analysis and diagnosis of medical treatment as well as allowing surgical simulations. It is easier to differentiate various structures, make assessments and take measurements from volume data. A volume image is constructed as shown in figure 3.5 from a series of slices given in figure 3.2.

3.2.2 Volume of Interest (VOI)

A volume medical image is constructed from slices of data. Some image processing tools, such as filtering, may be used at this stage to remove slice artifacts from the image. Such artifacts are visible in figure 3.5. It is essential to identify or specify the regions of the object before any further processing takes place. This process is defined as selecting a volume of interest. Determining a VOI will allow better diagnostics as well as produce specific models for the simulations. Segmentation is the first step towards this direction.

3.2.2.1 Segmentation

Image segmentation subdivides an image into surgically identifiable structures making it more useful to the operator. Segmentation is achieved by categorizing voxels as belonging to certain tissue types such as bone or fat (Atkins and Mackiewich 1998, Cohen et al. 1992, Collins et al. 1996, Szekely et al 1996). This is achieved using two different methods. The first method searches the differences in pixel grey level and detects structure boundaries. The second method looks for similarities in pixel grey level for the detection of object regions. Since analyzing complex 3D structures from a series of slices is a very difficult task, image segmentation provides following advantages:

- It generates 3D models for visualization of complex structures.

- It allows the operator to see how these structures lie in relation to one another.
- It provides volume measurements and better operative planning.

Segmentation can be done mostly with semi-automatic procedures using thresholding, region growing or manually on each slice. We have employed a simple thresholding method to extract skin and bone surfaces. Figure 3.4 shows thresholding in operation while obtaining tissue elements from the slice data. There may be several attempts to obtain the best results. The result of the segmentation procedure is shown in figure 3.5 where a head image is depicted.

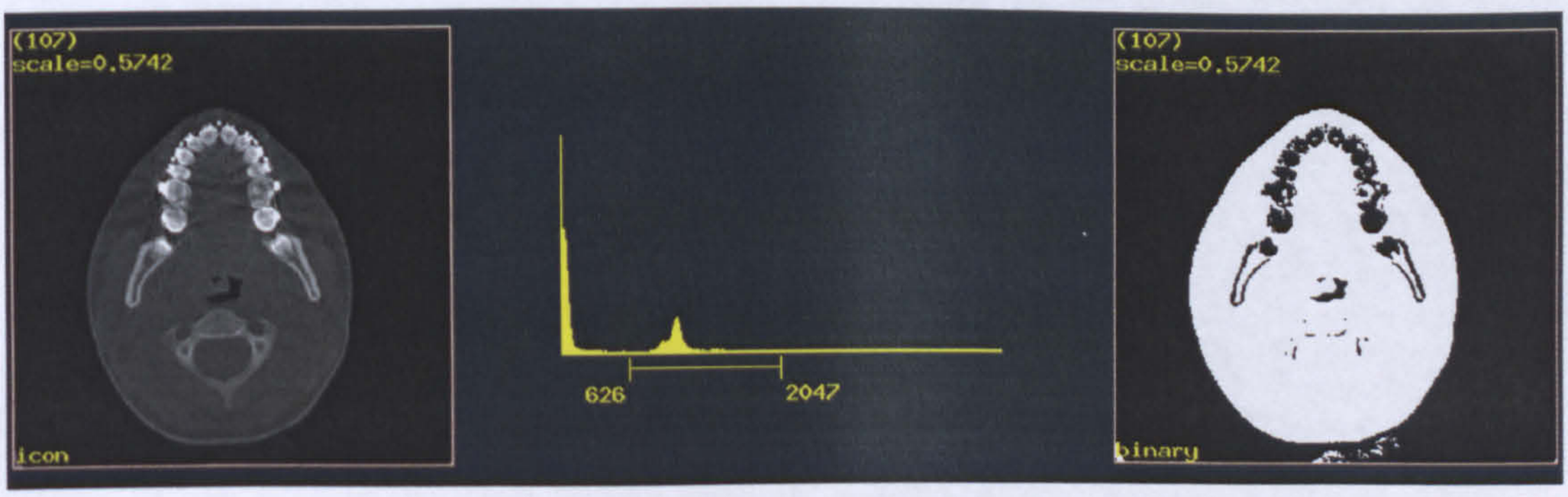


Figure 3.4 Segmentation of soft tissues from a head image.

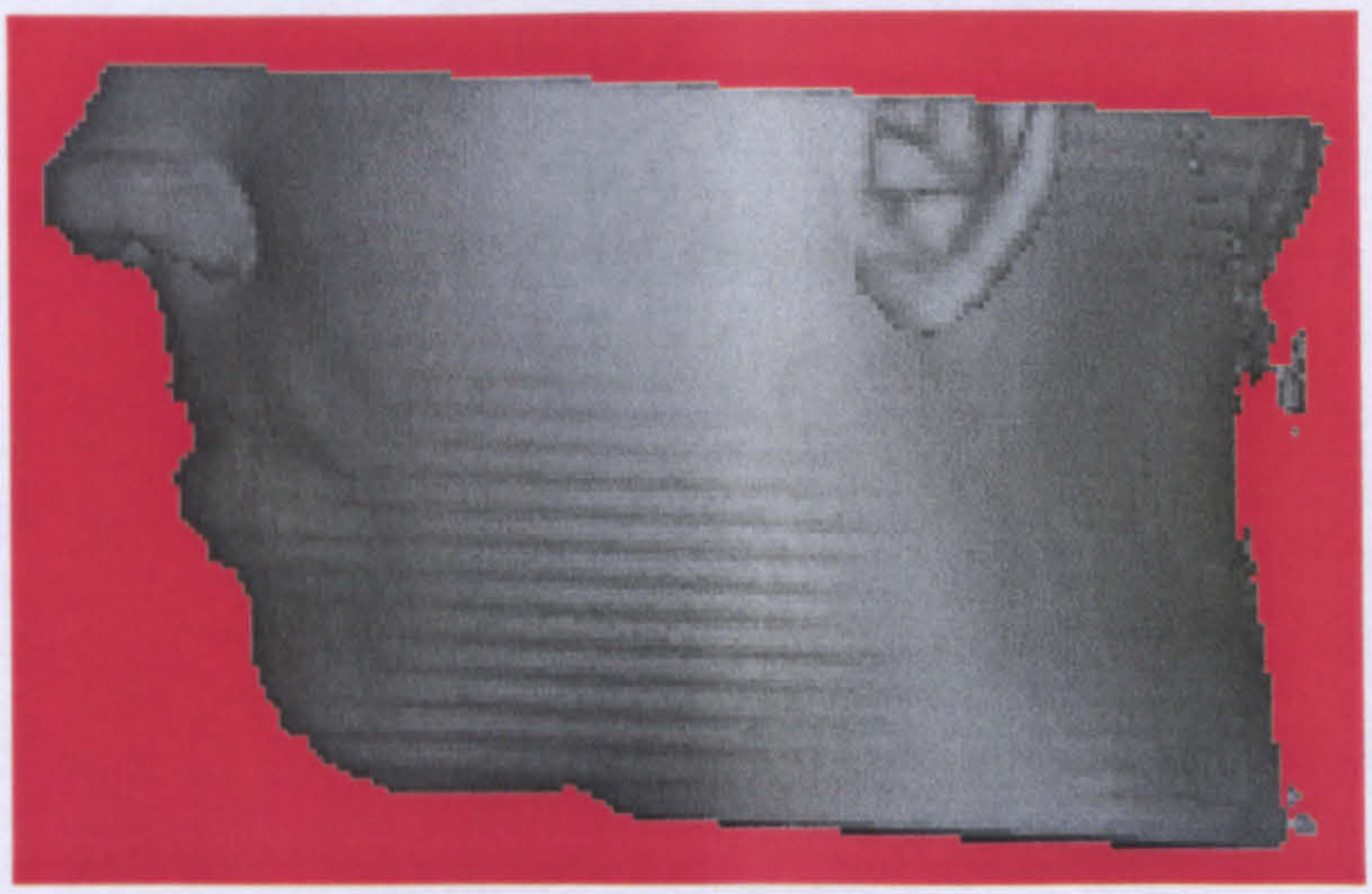


Figure 3.5 A segmented head image.

Threshold levels are different for bones and soft tissues. Segmenting the bone structure is represented as seen in figure 3.6 and the corresponding skull image is shown in figure 3.7. Once an object of interest is segmented, the next step involves further localizing the volume image. Necessary measurements are taken on the volume image allowing us to make assessments. Cutting and separating of the volume image may then be required as examined in the following sections.

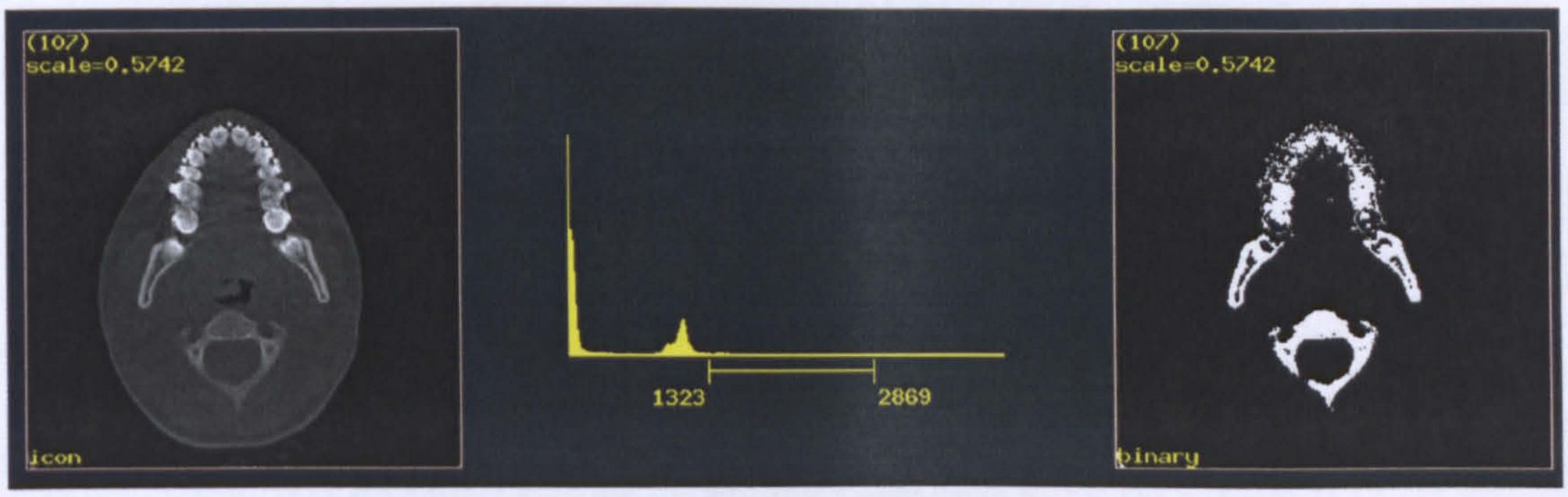


Figure 3.6 Segmentation of bones.

Figure 3.7 Bones segmented from head image.

3.2.3.2 Manipulations and Measurements

Based on the requirements of the operation and planning, segmented objects may need to be cut and separated. Individual pieces may be translated and rotated allowing the operator to make assessments for different variations preparing for surgical planning. Surgeons can make necessary measurements, which can be 2D or 3D. This is essential for surgical planning because most of the decisions are made at this stage.

The head model given by figure 3.7 is cut to focus on the jaw area (VOI), where the operation takes place. The resulting image is shown in figure 3.8, where some marks are placed. The operator can put marks on the image for measurements or cutting purposes. Whereas image 3.9 (a) and (b) shows some measurements and markings for cutting, figure 3.10 (a) and 3.10 (b) shows the top and side view of a jaw cut on the marked places. The image shown in figure 3.10 (c) include all the parts used in modeling, the lower and upper jaw. Individual parts can now be moved or rotated. The lower jaw is pushed backward by about 8mm to close the gap caused by the cutting operation. The final image is shown in figure 3.11.

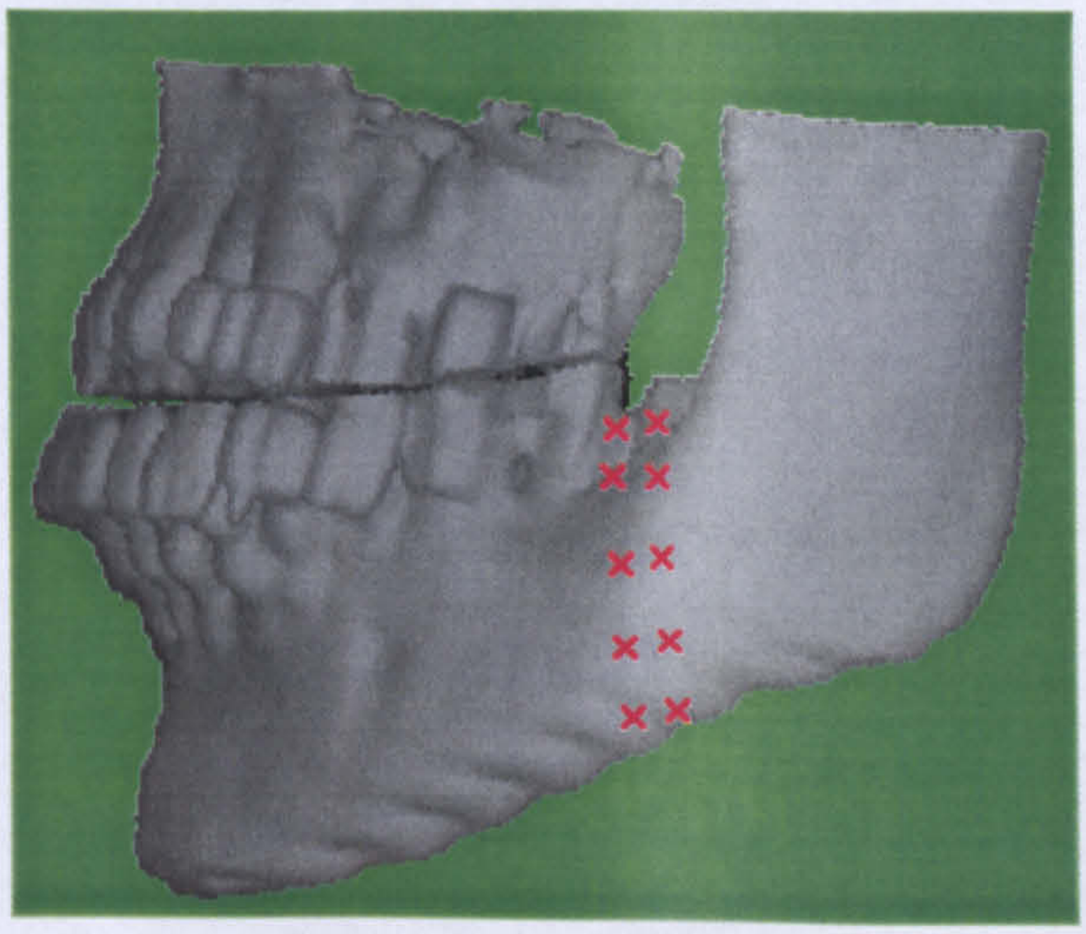


Figure 3.8 The upper and lower jaw with some marks for cutting.

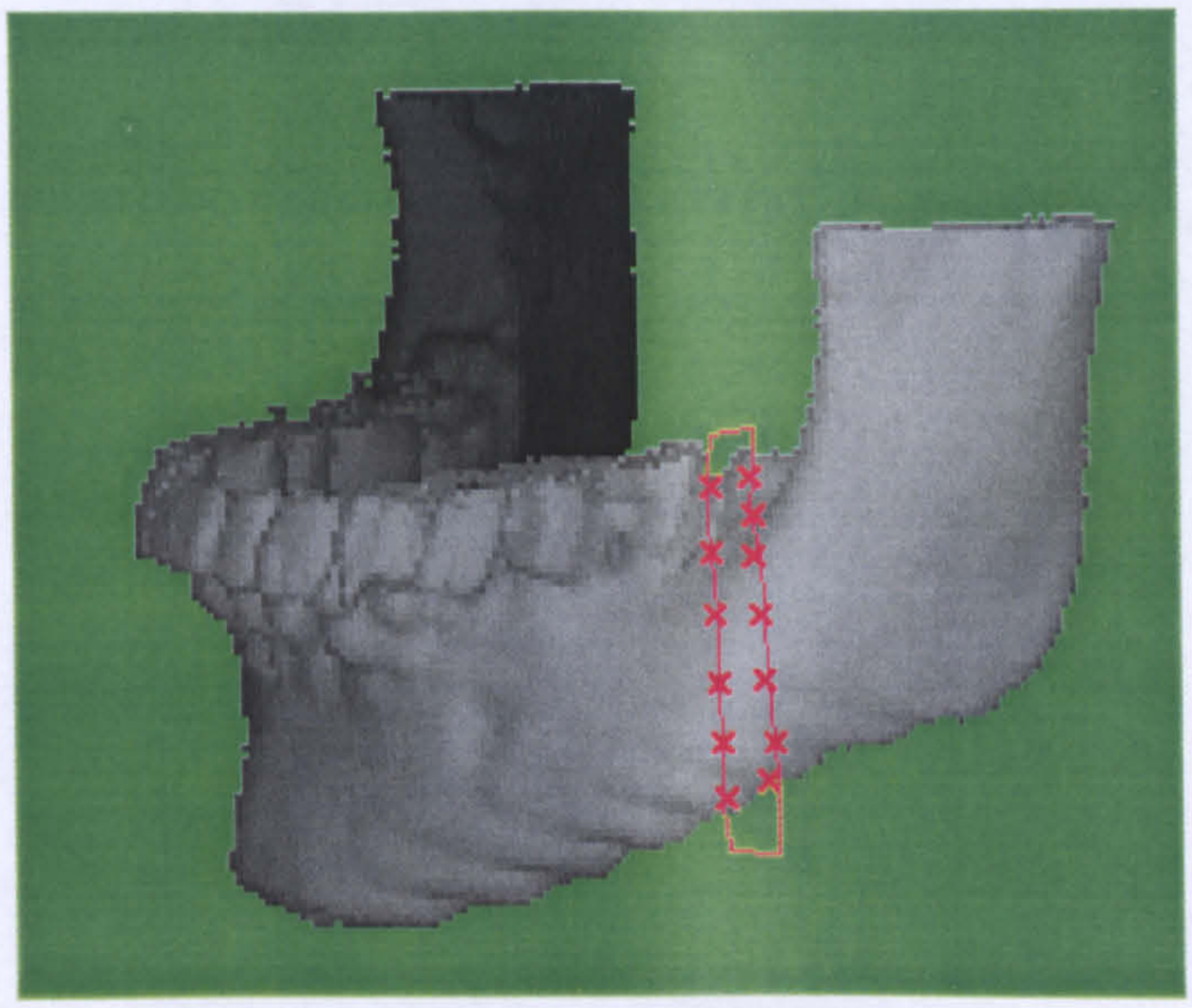


Figure 3.9 (a) The lower jaw is measured and marked to be cut.

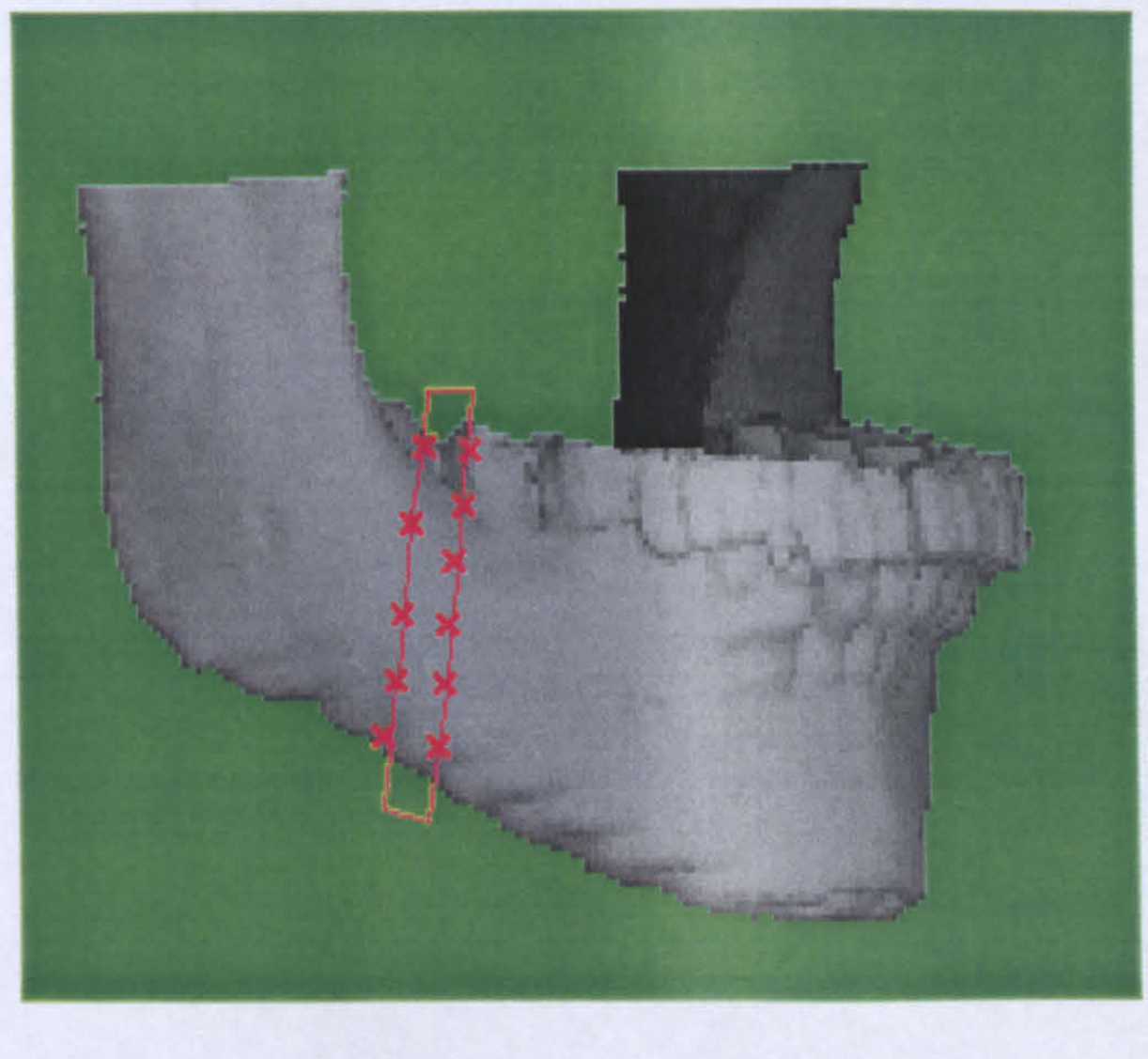


Figure 3.9 (b) The other side of the lower jaw ready for cutting.

Figure 3.10 (a) The lower jaw view from top showing the cutting operation.

Figure 3.10 (b) The jaw cut, a side view.

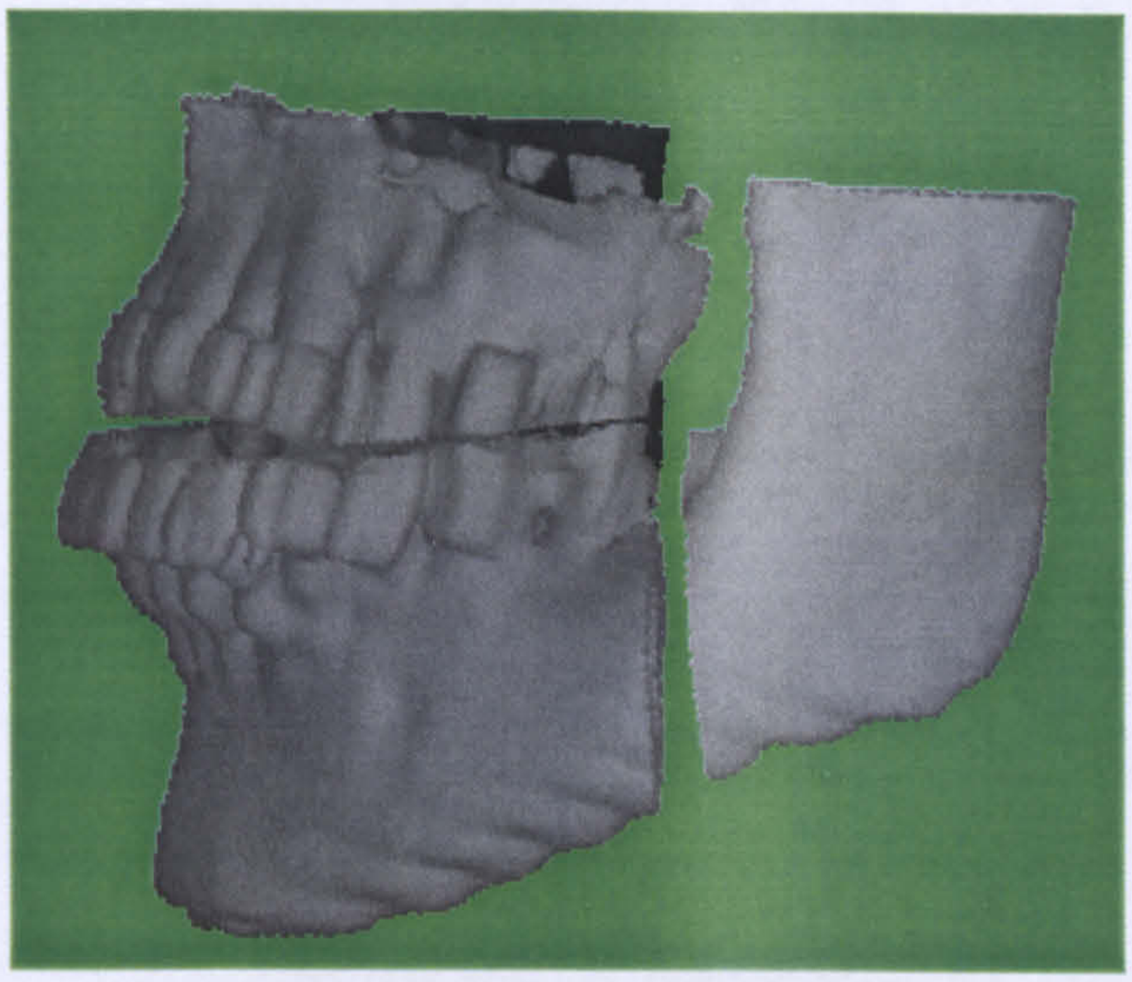


Figure 3.10 (c) Bone structures for all cutting operations are complete ready for movement.

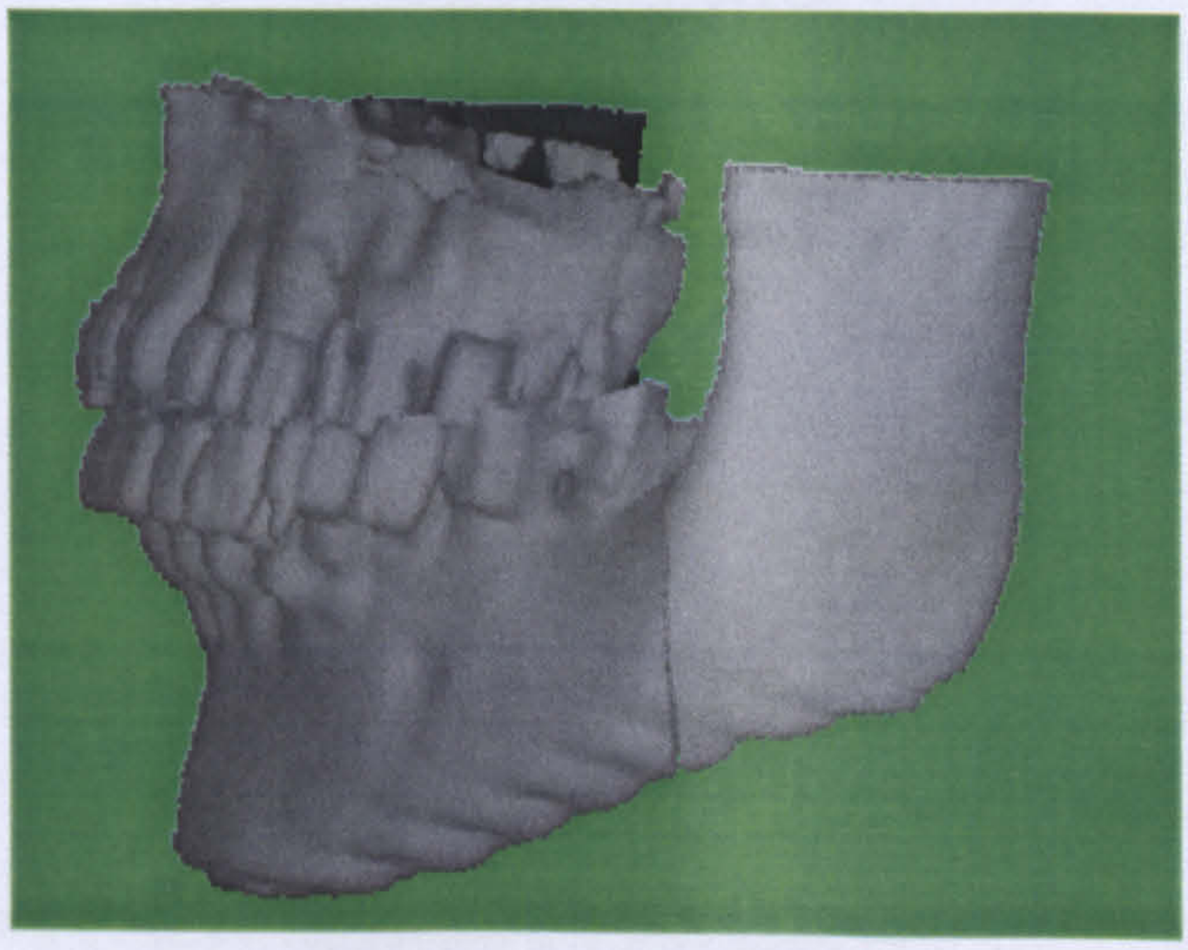


Figure 3.11 Bone structure after movement of lower jaw. Upper and lower jaws are aligned.

This phase includes several attempts to determine on how much bone cutting is necessary movement direction of the parts. The operator reaches a decision after seeing the outcome of these attempts. A final decision of course is reached after the simulation of the soft tissue changes based on the cutting and movement of these bones. It may well be necessary to come back to this stage and repeat these manipulations.

Once the necessary operations are determined it is required to obtain a polygonal model of the VOI. This model is used in the mass-spring simulation, which operates on mesh representations. Obtaining such a model is examined next.

3.3 Polygonal Model Generation

The first step in the reconstruction of 3D geometric models is segmentation, extracting regions or features of interests. The second step is to generate a surface representation of segmented volumes. The geometric models representing the surface of a 3D segmented volume are often described by a set of triangles because of their simple structure. This structure allows fast mathematical manipulations and it is very suitable for surgical simulations using mass-spring systems and finite element modeling. There are many methods proposed for the isosurface generation. The marching cubes algorithm (Lorensen and Cline 1987) is one of the most popular methods used in generating surface triangulation because of its sub-voxel processing that produce high quality meshes. Here, we will briefly describe the algorithm by explaining how it forms triangulated surfaces using voxel information. Since the marching cubes algorithm produces a very large number of triangles, we will also examine decimation algorithms.

3.3.1 Triangulation

The marching cubes algorithm uses the pixel information at the eight corners of a voxel and compares the density information of these pixels with a given threshold value, thus determining how the surface intersects the voxel. Therefore, this algorithm can be seen as being a thresholding algorithm. The voxels (cubes) are considered to fall within the object, if their eight corners (vertices) are above the given threshold. Alternatively, if all eight corners are below this threshold value, then the cube is considered to be completely outside the object. For the remaining cubes, each corner of a cube is classified as being either inside or outside the isosurface of the volume. The isosurface is formed at cube edges where one vertex of the edge lies inside the volume and the other outside. Triangle vertices are then computed by finding the points of intersection of the cube edges that penetrate the isosurface. The intersection point is determined using the density information of the pixels.

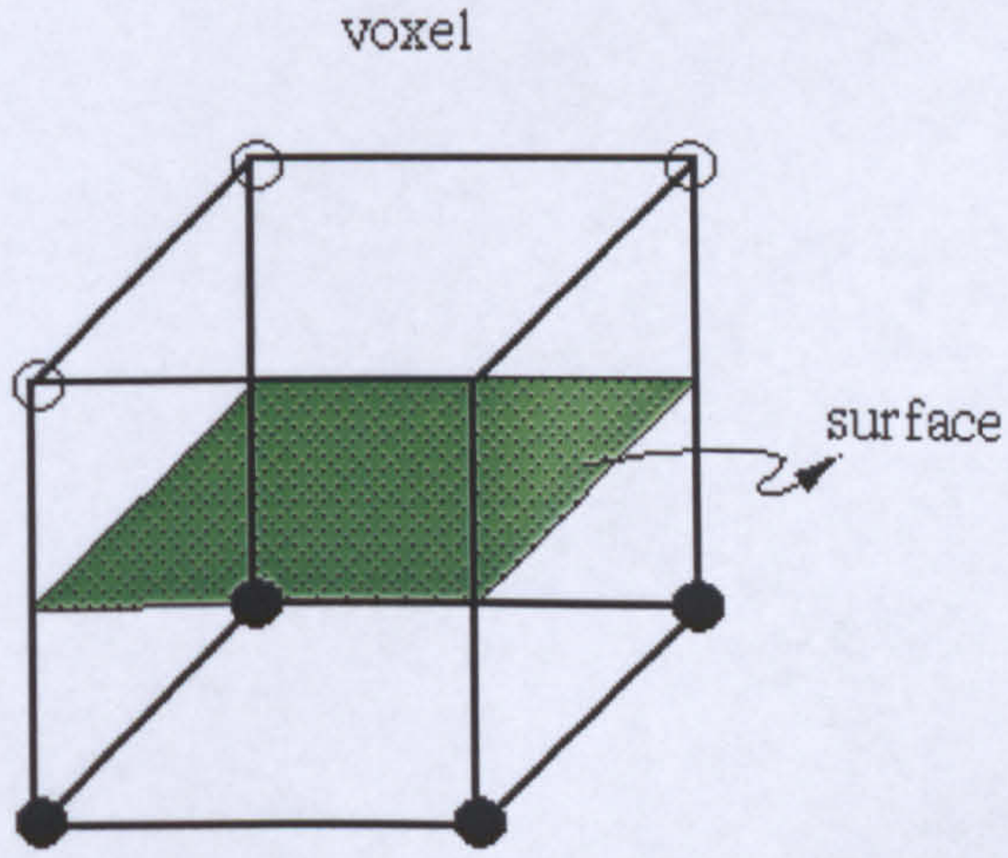


Figure 3.12 Forming a surface with the marching cubes algorithm.

Figure 3.12 depicts a voxel that is partially inside the volume. As seen from this figure, four of the cube's vertices are above the threshold value, while the remaining are below it. The isosurface is then formed somewhere between the vertices that are below and those that are above the given threshold value. It is important to note that

the figure shown here only represents one of the 23 possible cases of the marching cubes algorithm (MarchingCubes).

Triangulation algorithms are a well-researched subject and the original marching cubes algorithm has been significantly improved. In our work we use the version of the algorithm developed by Nielson (Nielsen). We have used this algorithm on the brain image data, mentioned above, to generate the surface representation. The result is shown in figure 3.13. As expected the marching cubes algorithm produces a large number of triangles, which needs to be reduced in order to allow fast rendering. This was achieved by using the decimation algorithms examined below.

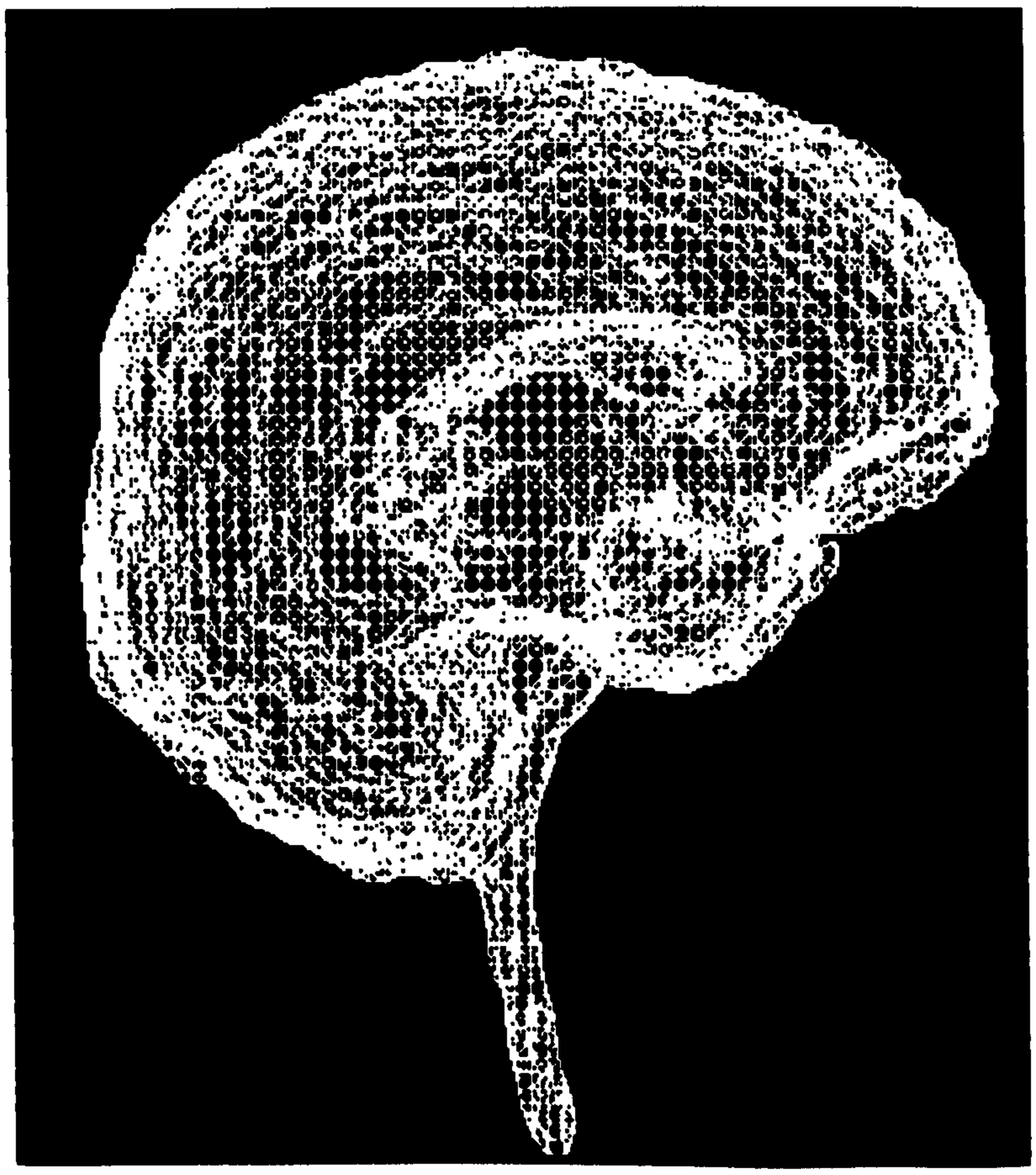


Figure 3.13 Brain surface model generated using the marching cubes algorithm that consists of 12264 vertices and 24473 triangles.

3.3.2 Decimation

The marching cubes algorithm produces a very good surface approximation. Unfortunately, it produces a large number of triangles that usually are difficult to manipulate and render in real time. Therefore the large number of triangles produced needs to be reduced to a reasonable number without significant loss of surface detail. There have been many mesh simplification (decimation) algorithms developed, which are based on the elimination of degenerate triangles and triangles located in the flat regions. Neilsen's algorithm, which is based on an edge collapse technique, was used to decimate the triangle mesh. Details of his algorithm can be found in (Nielsen). The generated brain volume image shown in figure 3.13 was then decimated and the output of the decimation algorithm is shown in figure 3.14.

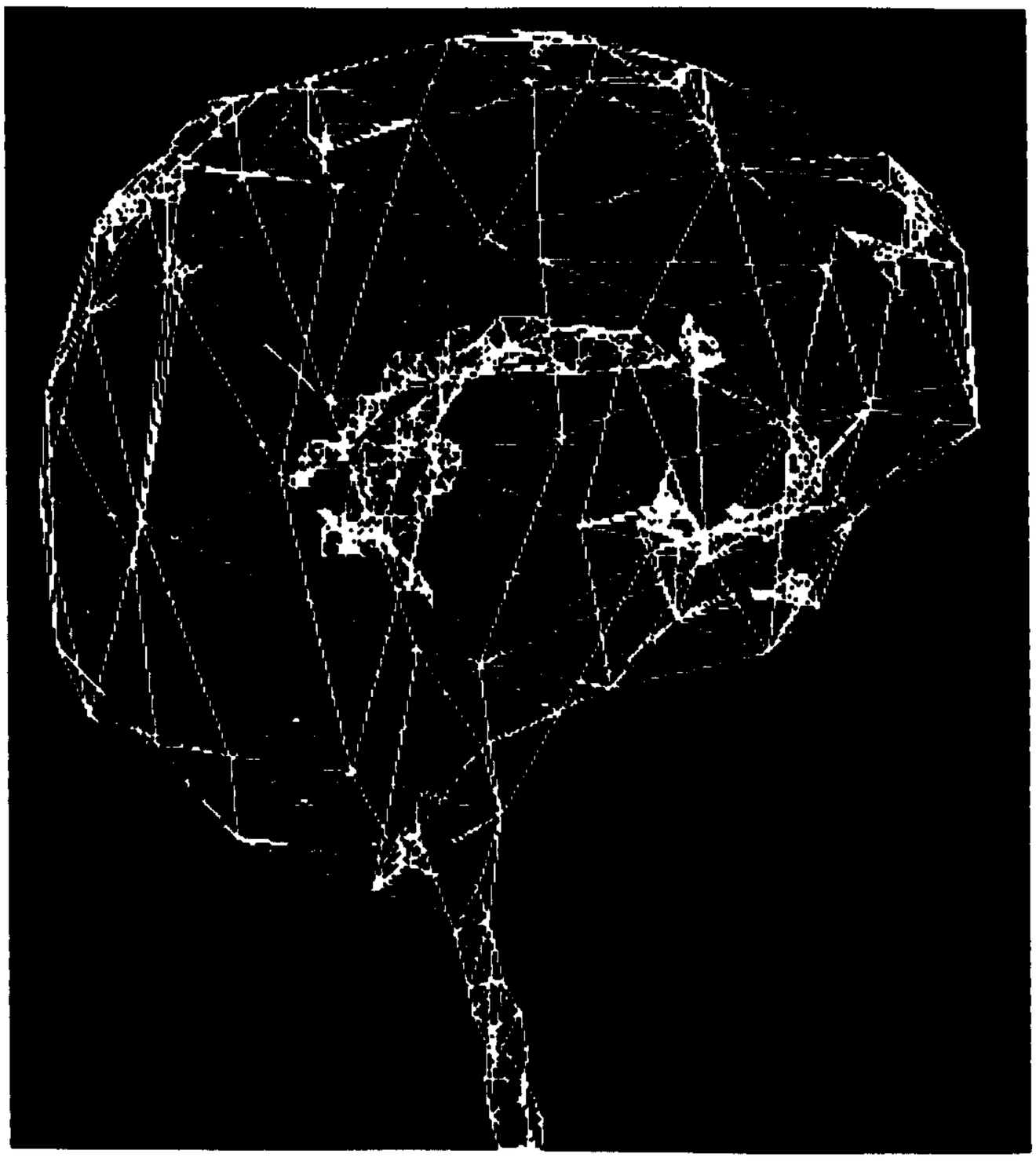


Figure 3.14. The triangular mesh decimation of a brain image. The output model was reduced to 486 vertices and 997 triangles.

3.4 Craniofacial Surgery Simulation

Simulation of facial tissue deformations is a very challenging task because it consists of a number of difficult subtasks including:

- The generation of appropriate models for the different tissue layers.
- The determination of the unknown model parameters.
- The verification of the simulation output.

In the following sections we obtain a facial model consisting of bone and skin surfaces. The different tissue layers between the two surfaces are represented using volume prism elements. A new method is used for finding the connections of the prism elements to the bone structure. Once the model is obtained the mass-spring systems algorithm can be employed to run the simulation.

3.4.1 Mass-Spring System Model Generation

The bone and face surfaces are generated using the marching cubes algorithm and the results are then decimated ensuring the number of triangles generated is suitable for the simulation algorithm. Both surfaces are shown in figure 3.15, where the face surface consists of 6292 vertices and 10519 triangles and the bone surface has 10056 vertices and 19770 triangles.

In some applications only this data, i.e. the triangular representation of the surface, is used in simulations. This type of modeling does not capture the volume characteristics of the soft-tissue behavior. Therefore volume modeling between the face surface and the bone surface is necessary to simulate the interior structure of the object. To model the volume, many different building blocks have been proposed, the most popular of

which include the use of thetrahedra and prismatic elements. We have implemented our model using prismatic elements.

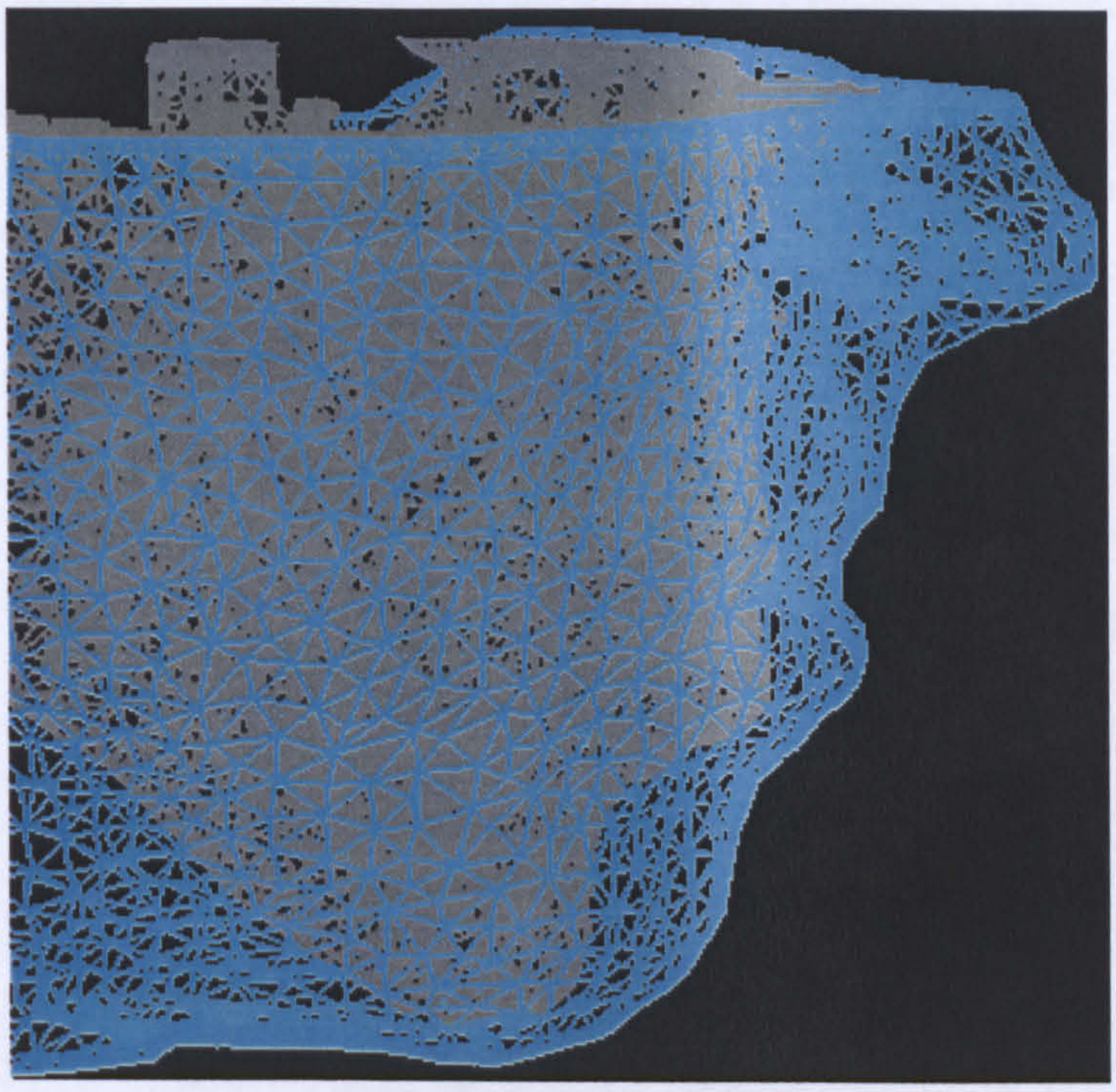


Figure 3.15. The surface representation of the skin and bones of a human head.

3.4.1.1 Generating Prism Elements

The tissue between the face surface and the bone surface is represented by a number of small prismatic volumes. Given that the skin and bone surfaces are represented by triangles, prismatic elements can be generated using various methods. Each vertex of a triangle of the skin surface is projected on to the bone surface. This is done by finding the intersection points between the normal vectors of each skin vertex and the bone triangles. For better and smooth results the normal vertex is determined by averaging the normals of the triangles meeting at this vertex. Since the face surface contains curved regions, it may be impossible to find an intersection point for every normal vector of a skin vertex. Besides, there are hollow areas on the bone structure preventing an intersection. In addition, some of the intersection points found may be

at completely the wrong places. Therefore using the skin vertex normal will not result in a good representation of tissue layers.

In (Keeve et al. 1998) a method tracing a ray from each skin vertex to a predetermined point on the bone structure (or a point inside the skull) is used in the generation of the prismatic elements. An average point, called the center point, is determined for all bone triangles. Each skin vertex is then traced towards this point and intersections with the bone surface are recorded. This process is illustrated in figure 3.16. If there is no intersection, then by interpolating the neighboring points of intersection we generate a false point of intersection. This method guaranties an intersection point for each skin vertex but may not produce very good prismatic element shapes. In addition some prisms may overlap. This happens because a single center point can not realistically represent the midpoint of the face, which is not a sphere. Interpolating neighboring elements in order to assign a connection point may result in an unrealistic approximation as well.

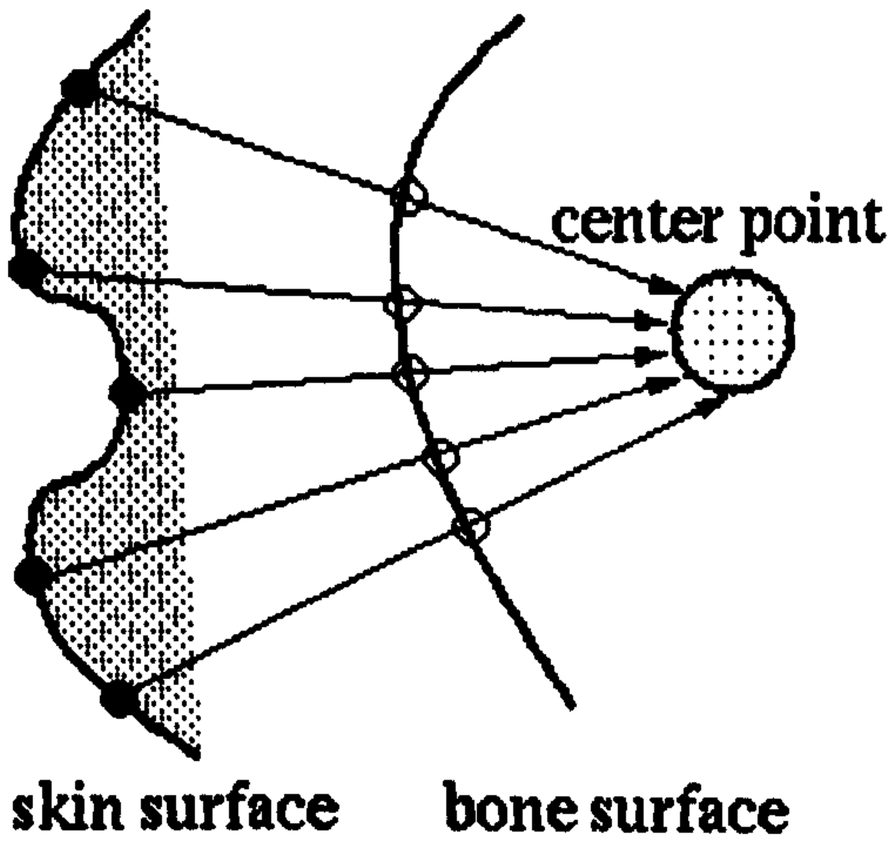


Figure 3.16 Projecting skin vertices on the bone surface.

We modified the method mentioned above as follows. Figure 3.15 reveals that the facial model is higher than it is wide, resembling an ellipsoid. Therefore assigning two center points may better represent the facial model in terms of finding the origin. As shown in figure 3.17 we placed two center points into the bone structure. The lip level at the skin surface determines which skin vertices use a specific center point. Vertices above the lips level are traced back to the center point at the upper part of the jaw. Vertices below the lip level use the center point at the lower jaw. Each skin vertex is then traced back to one of the two center points and any intersection with the bone triangles is recorded. On this first run we also determine an average thickness based on the intersected rays. On the second run we assign an intersection point to those vertices that did not get a hit in the ray direction on the first run. The average thickness is used to determine the depth of these intersections. This method also guaranties an intersection for each skin vertex and produces better-shaped prism elements, representing different tissue layers, while minimizing any possible overlaps.

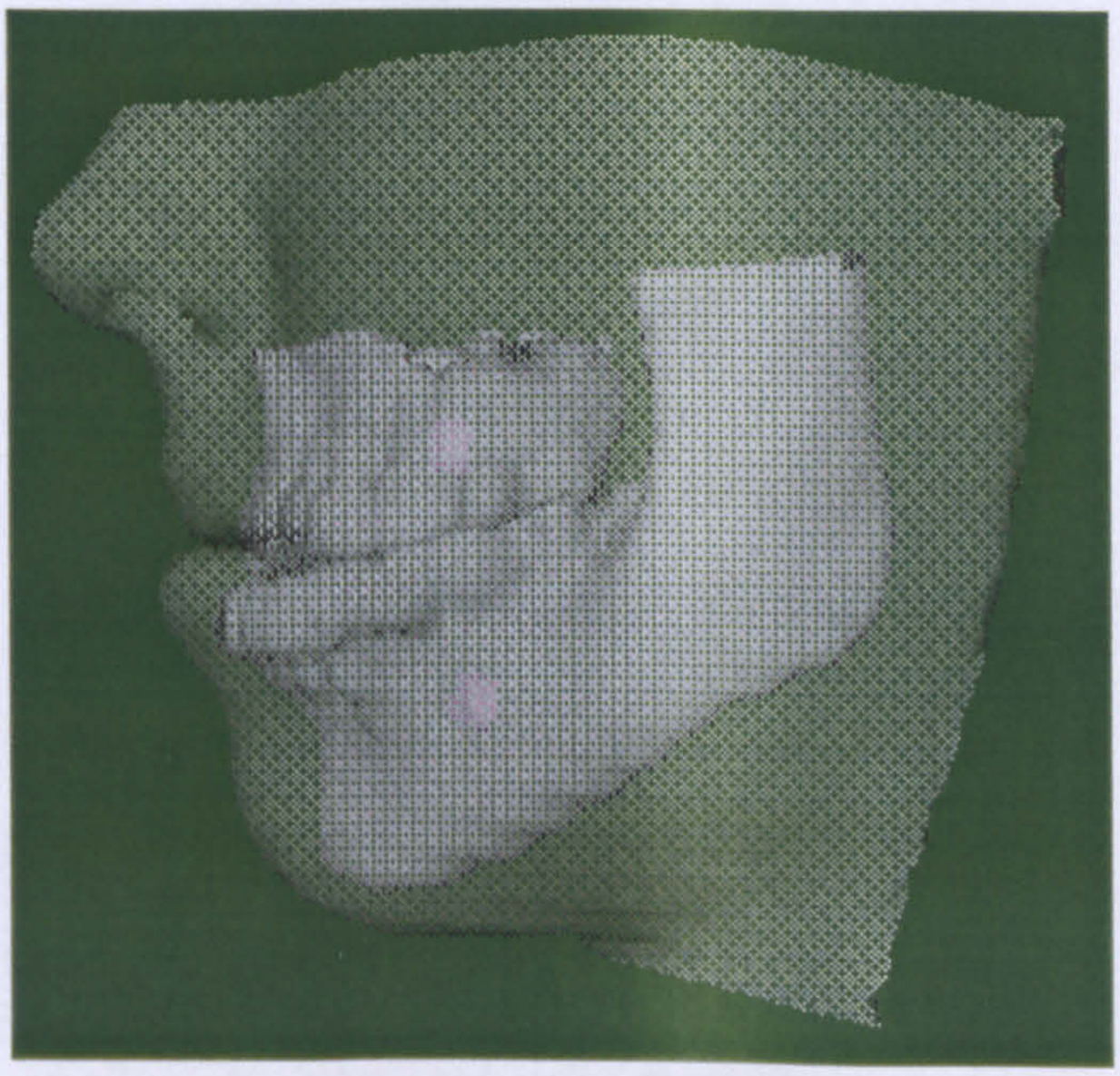


Figure 3.17 Facial surface, bone surface and two center points shown together.

The ray tracing algorithm was examined in chapter 2. Using this algorithm, prism elements are obtained. A typical prism element between the skin and bone surfaces is shown in figure 3.18. The number of prism elements is equal to the number of skin triangles. It is important to note that none of the skin vertices above the bone level are used in this process. The skin and bone surfaces as well as the bone level can clearly be seen from figure 3.17. Therefore the number of skin vertices involved in finding the prism element is 2628 and the number of the skin triangles is 4606. The next section explains how prism elements are used in the mass-spring simulation.

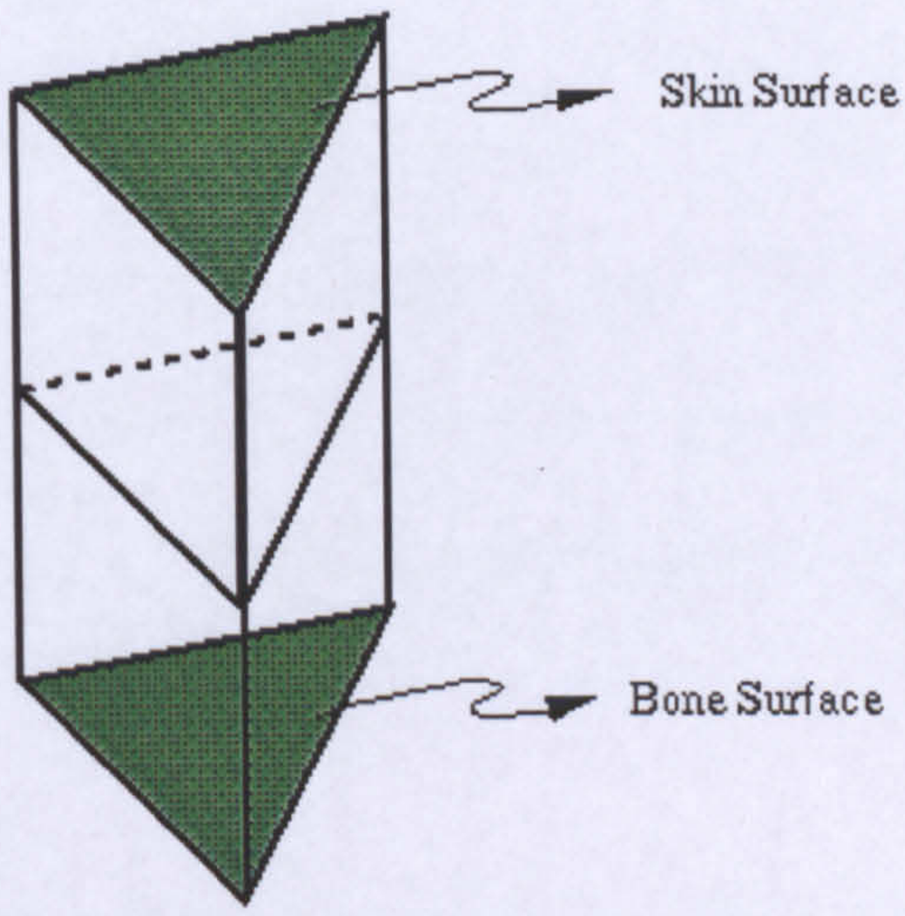


Figure 3.18 A prismatic element between the skin surface and the bone surface.

3.4.1.2 Assigning Mass-Points and Springs

Once the prism elements representing different tissue layers between the skin and bone surfaces are obtained, it is necessary to explain their roles in the mass-spring systems simulation. Their roles are determined by defining spring types and system parameters. Some of the springs are classified as boundary springs, whose lengths are not affected by any movements. Springs on the bone surfaces and springs on the edges of the face fall in this category. Springs on the face surface are called skin-

springs, while springs connecting tissue layers are defined as layer-springs. Springs diagonally connecting different tissue layers are called shear-springs.

Different springs in the mass-spring representation will use different parameters, simulating various visco-elastic representations of tissue characteristics. It is very difficult to set the unknown parameters of the springs. In many of the previous applications, the parameter values used were not reported. This is mainly because they were set by trial and error to produce plausible results. They were not physically validated. In our work we adapted the values given in (Koch et al. 1996). Skin springs were set at high values such as 200. The stiffness value of the shear springs was chosen as 100 and finally, layer springs were assigned a value of 80. We set the damping value of all the spring to 0.2 and their mass value to 0.001. This essentially corresponds to 2.6 kg of weight for the entire face. The parameters of the boundary springs were set to zero, to prevent any movement at defined boundaries.

3.4.2 Bone Realignment

Once the volume elements between the two surfaces (skin and bone) are found, bone realignment, can take place. A simple operation aligning the upper and lower jaw is performed. As explained in section 3.2.3.2, the lower jaw is cut and pushed back by about 8mm as shown in figures 3.10 and 3.11. The very same operation is performed on the triangulated bone model shown in figure 3.19. Part of the lower jaw is cut according to the measurements found in section 3.2.3.2. The front part of the jaw is pushed back to cover the gap. This operation is performed on vertices found on the bone structure by the ray tracing algorithm. In other words, vertices connecting tissue layers to bone structure are translated to close the gap, created by bone cutting operation. Other vertices on different tissue layers or on the skin surface are not

affected by this operation. This operation will start the simulation, since the initial configuration of the mass-spring systems is now changed. Each part of the model given in figure 3.19 can easily be moved individually. We took advantage of this and also produced various bone animations by moving or rotating the lower jaw. The resulting facial tissue changes were then simulated. An example for this animation experiment is given in figure 3.20, where the lower jaw is rotated by a small angle.

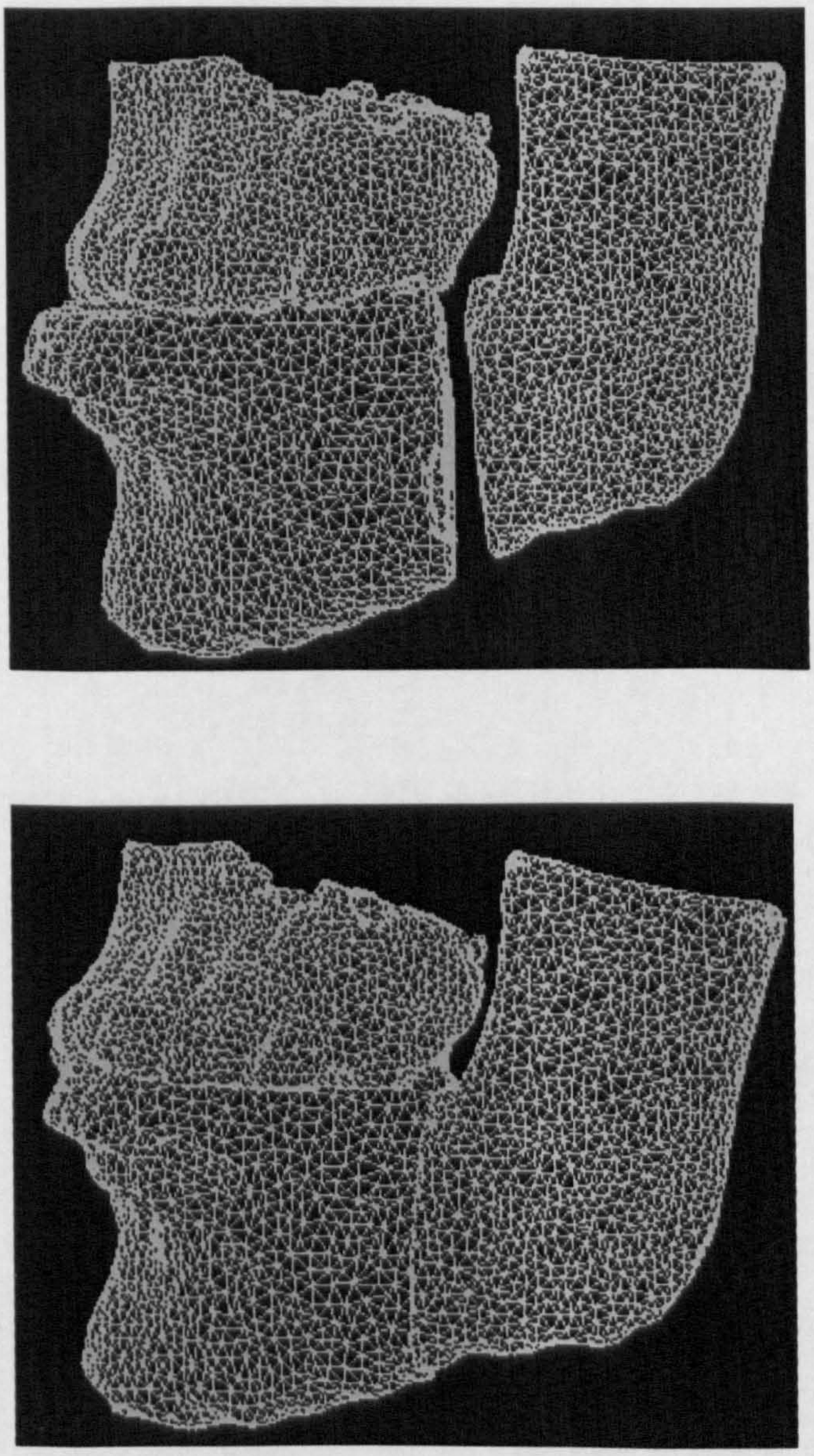


Figure 3.19 The upper and lower jaws are aligned using the triangular representation of skull model, which is used in the actual simulations.

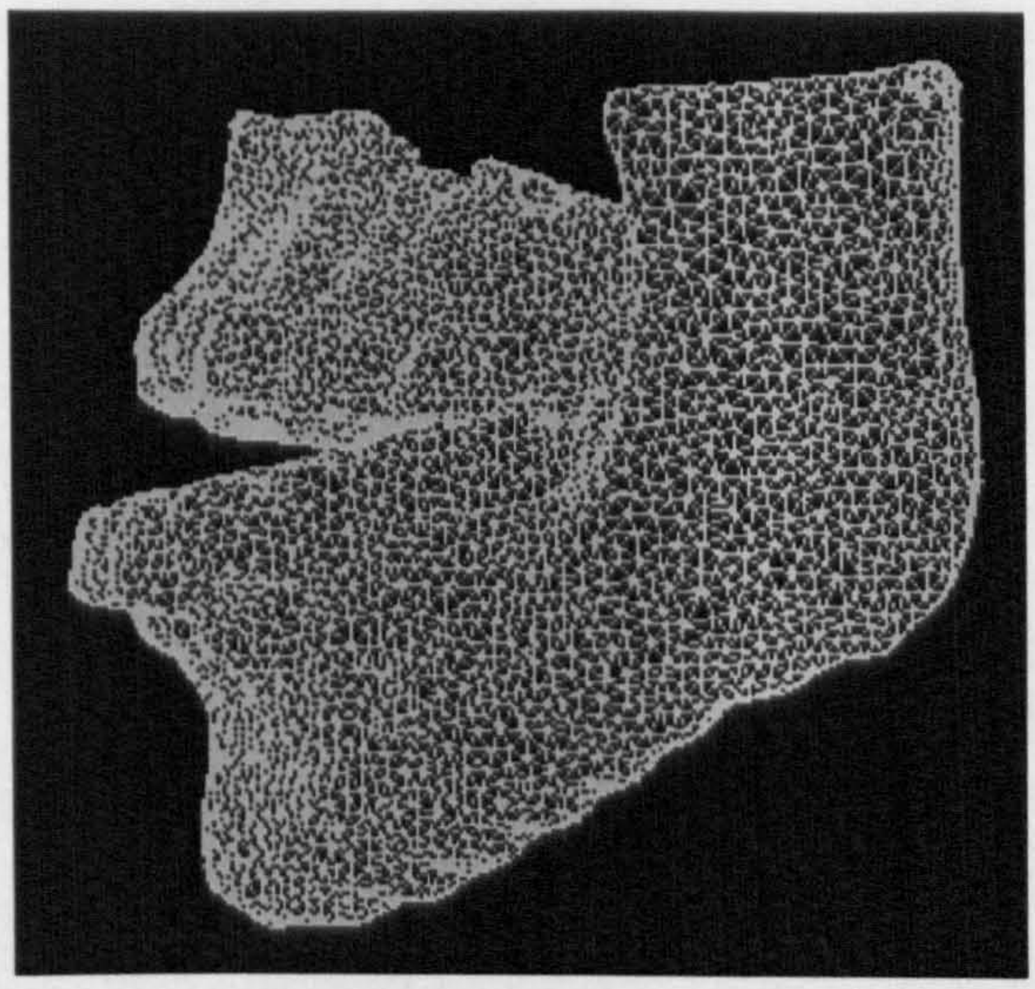


Figure 3.20 Lower jaw is rotated representing mouth opening.

3.5 Simulation Results

A mass-spring network is formed based on the vertices of the prisms acting as mass-points and the edges acting as springs. The lower jaw is cut and pushed back to align it with the upper jaw as illustrated in figure 3.19. The intersection points connecting tissue layers to bone structure were translated to close the gap created by the cutting operation. Changes in the bone structure cause the deformation of the soft tissue. This is because these changes will alter the spring rest lengths, defined in the original settings at the edges of the prism elements. There were no external forces applied in this simulation.

We carried out our simulations with a number of different tissue layers. Using one layer of tissue clearly did not work. The smoothness of the skin surface is compromised. Our experiments revealed that using 2 or 3 tissue layers works quite well. For performance reasons, we decided to use 2 layers of tissue volumes between the skin and bone surfaces. The number of iterations was determined after running the simulation several times.

The first part of the experiment simulates tissue deformation, whose original shape is given in figure 3.21. The face after the simulation is given in figure 3.22, while figure 3.23 gives both pre and post images together. Figures 3.24 and 3.25 represent the frontal view of the face before and after the simulation, respectively. As figure 3.22 reveals, a smooth deformation is achieved, where the lower and upper lips are aligned as well.

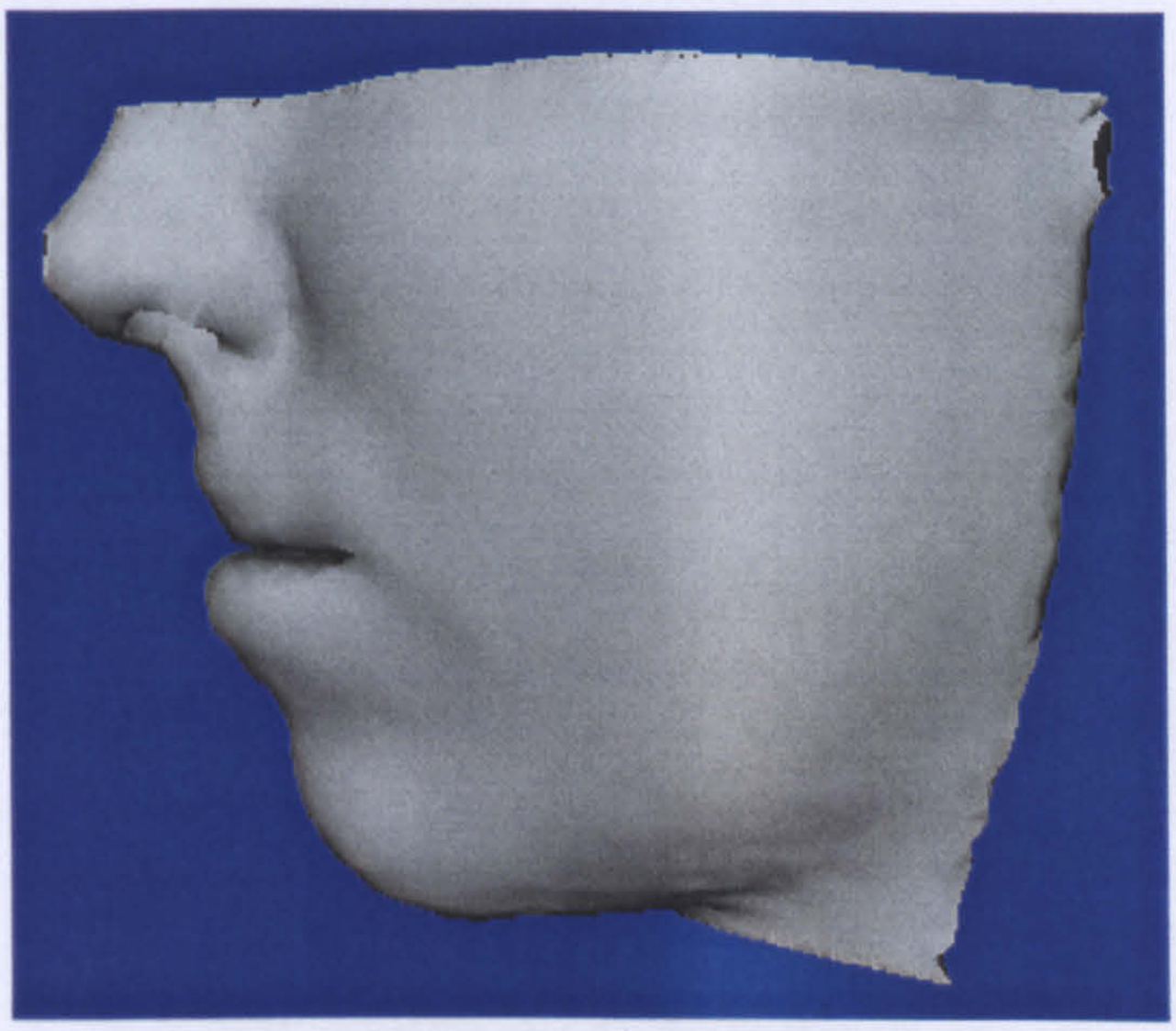


Figure 3.21 Face surface before the bone realignment and simulation.

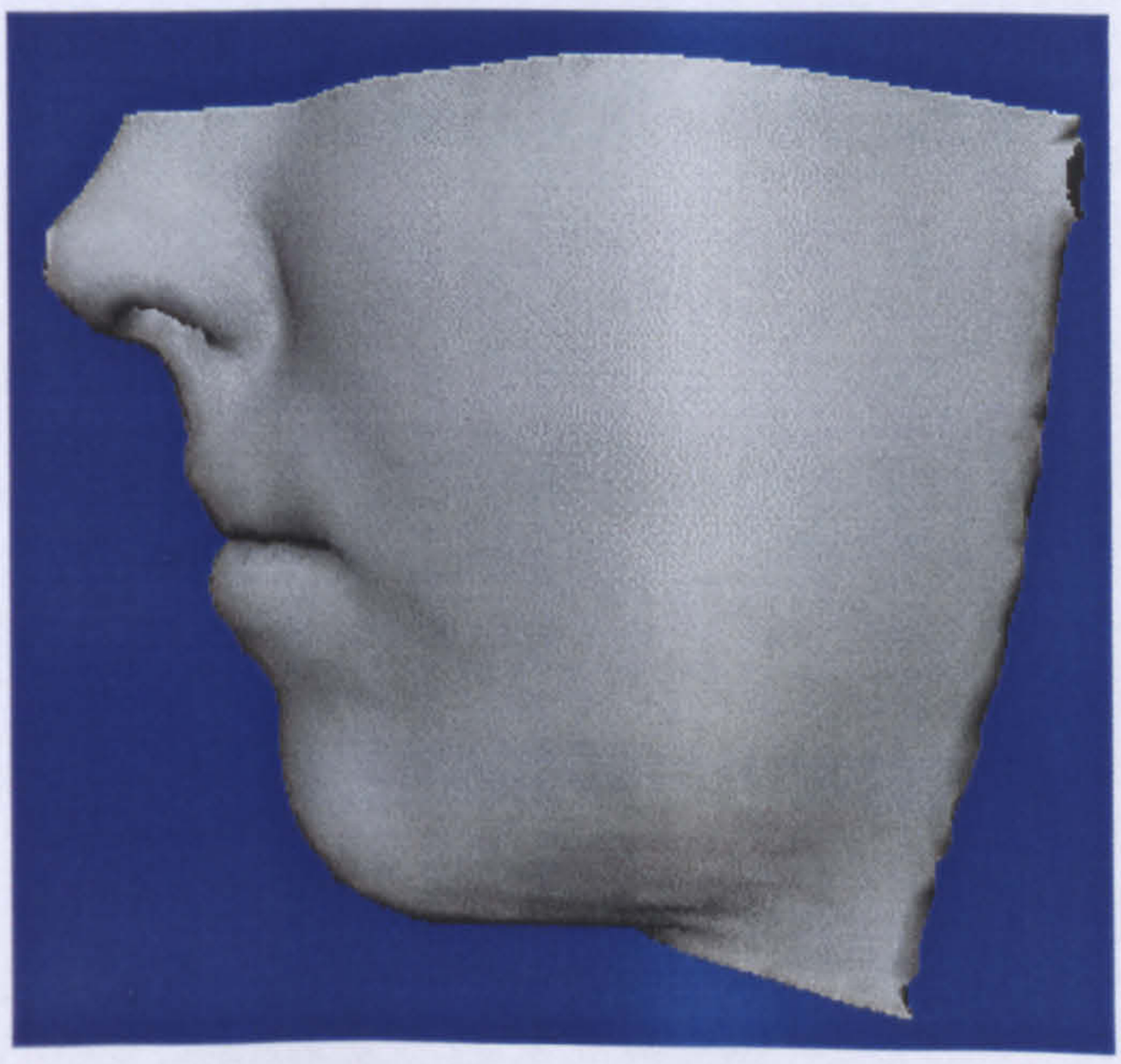


Figure 3.22 Face after the surgical simulations.

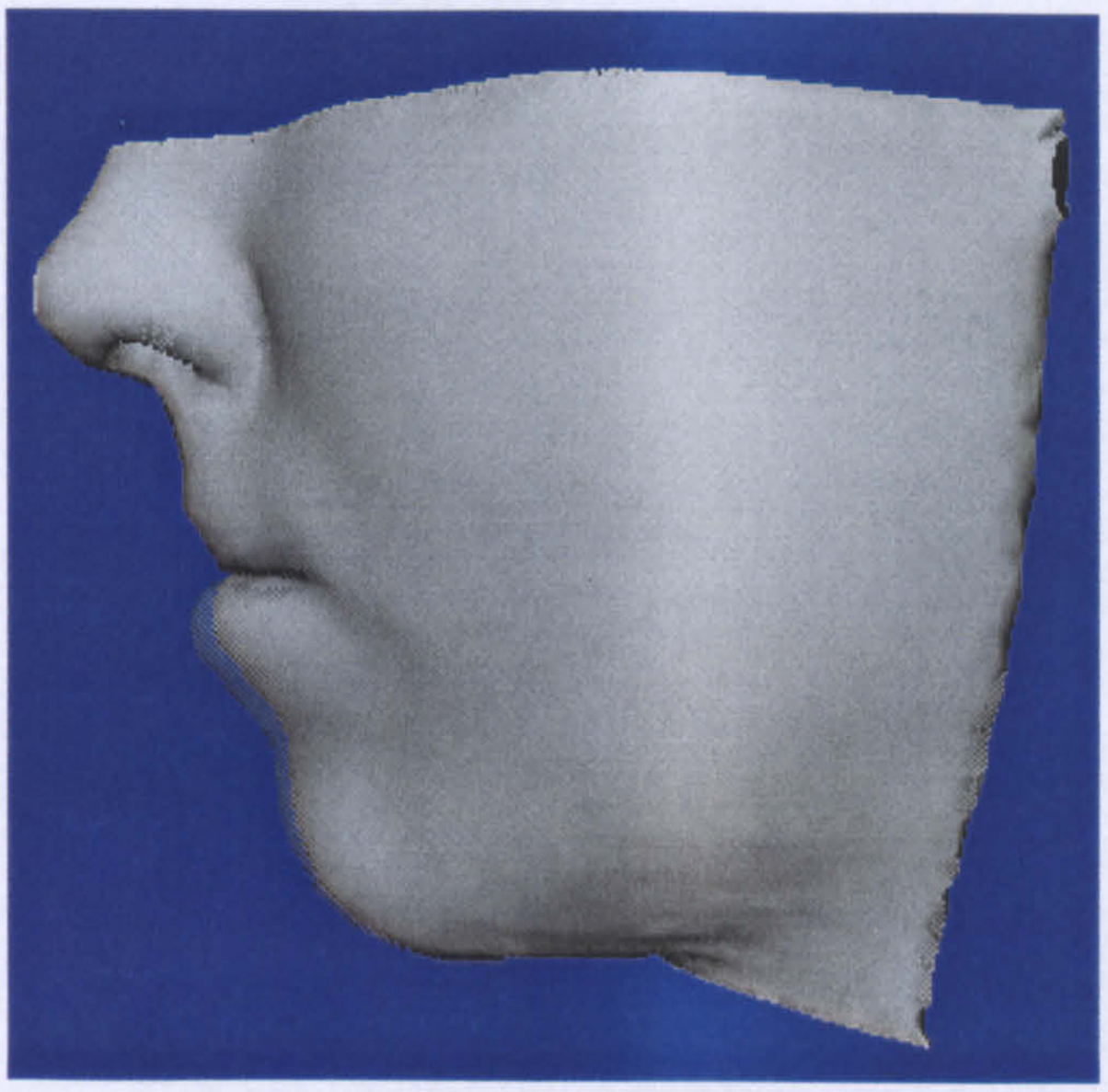


Figure 3.23 Both pre and post surgery images superimposed together showing the changes on facial tissue.

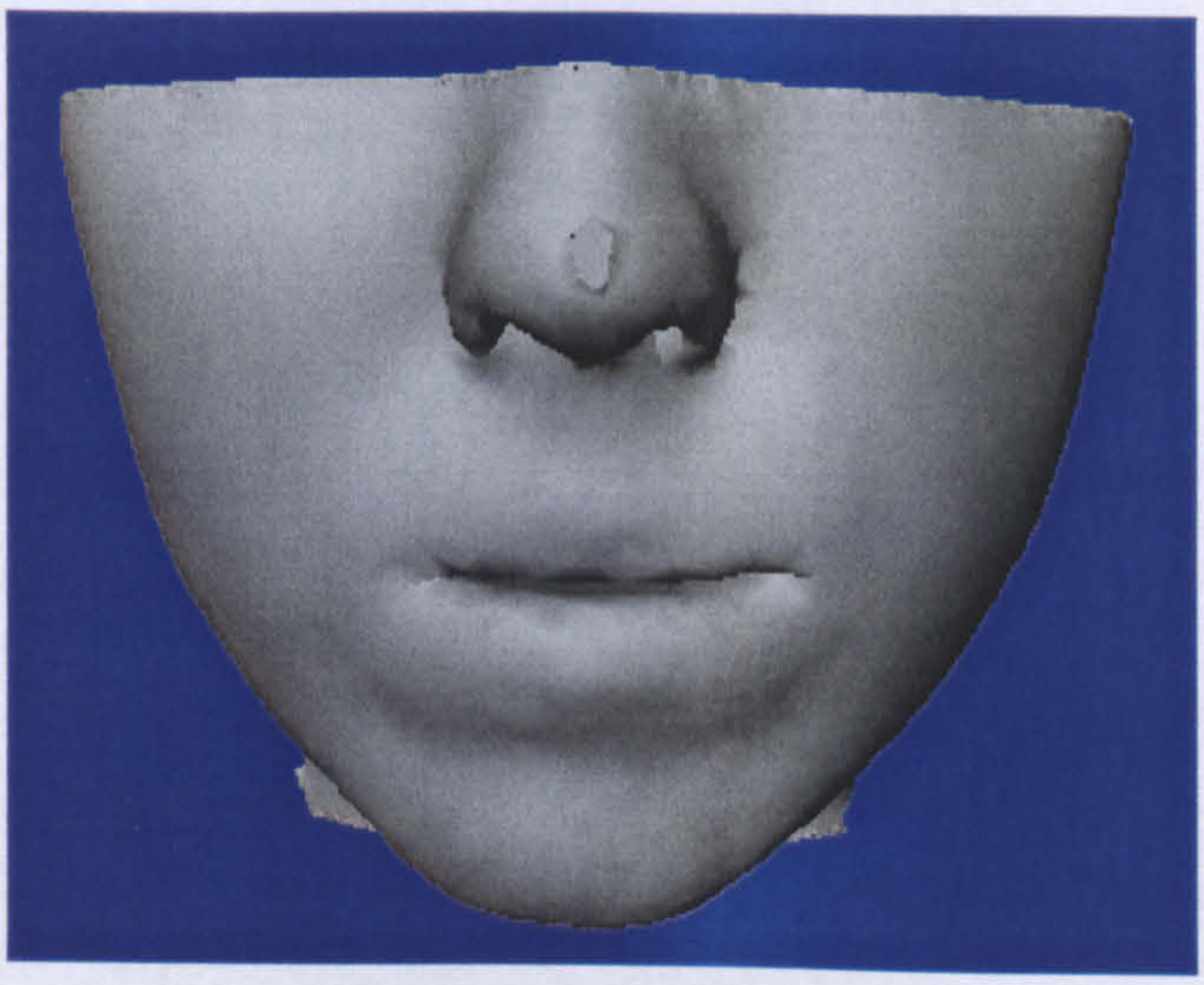


Figure 3.24 Facial image frontal view before the surgery.

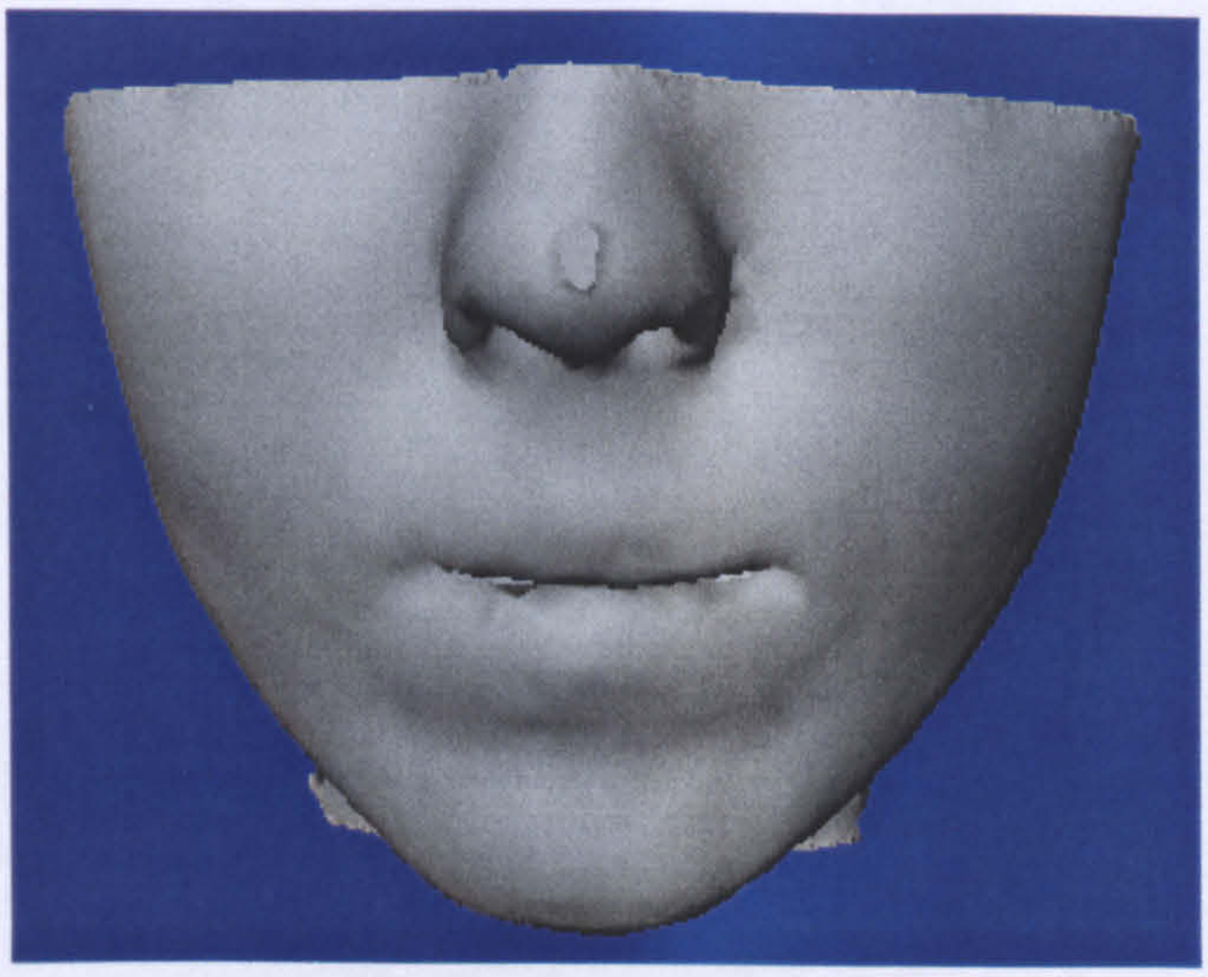


Figure 3.25 Facial image front view after the simulation.

The second part of the experiment demonstrates various facial expressions caused by bone movement. One such expression is given in figure 3.26, where the lower part of the jaw is pushed forward. This is an opposite movement to the one performed above for the craniofacial surgery simulation. The resulting image shows that the alignment of the upper and lower lips gets worse.

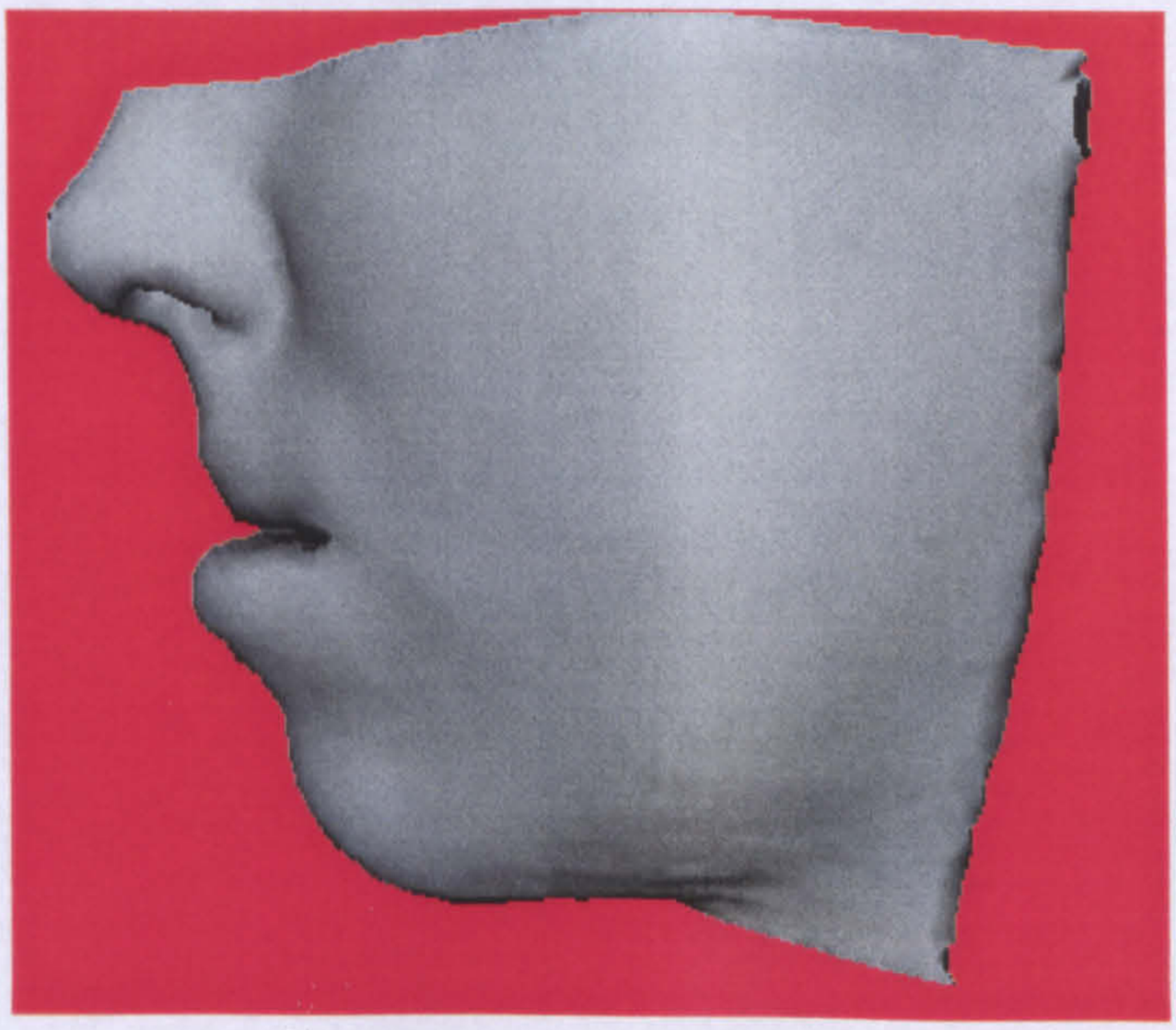


Figure 3.26 Instead of pushing the lower jaw backward to align with upper jaw, we push it forward by about 8 mm.

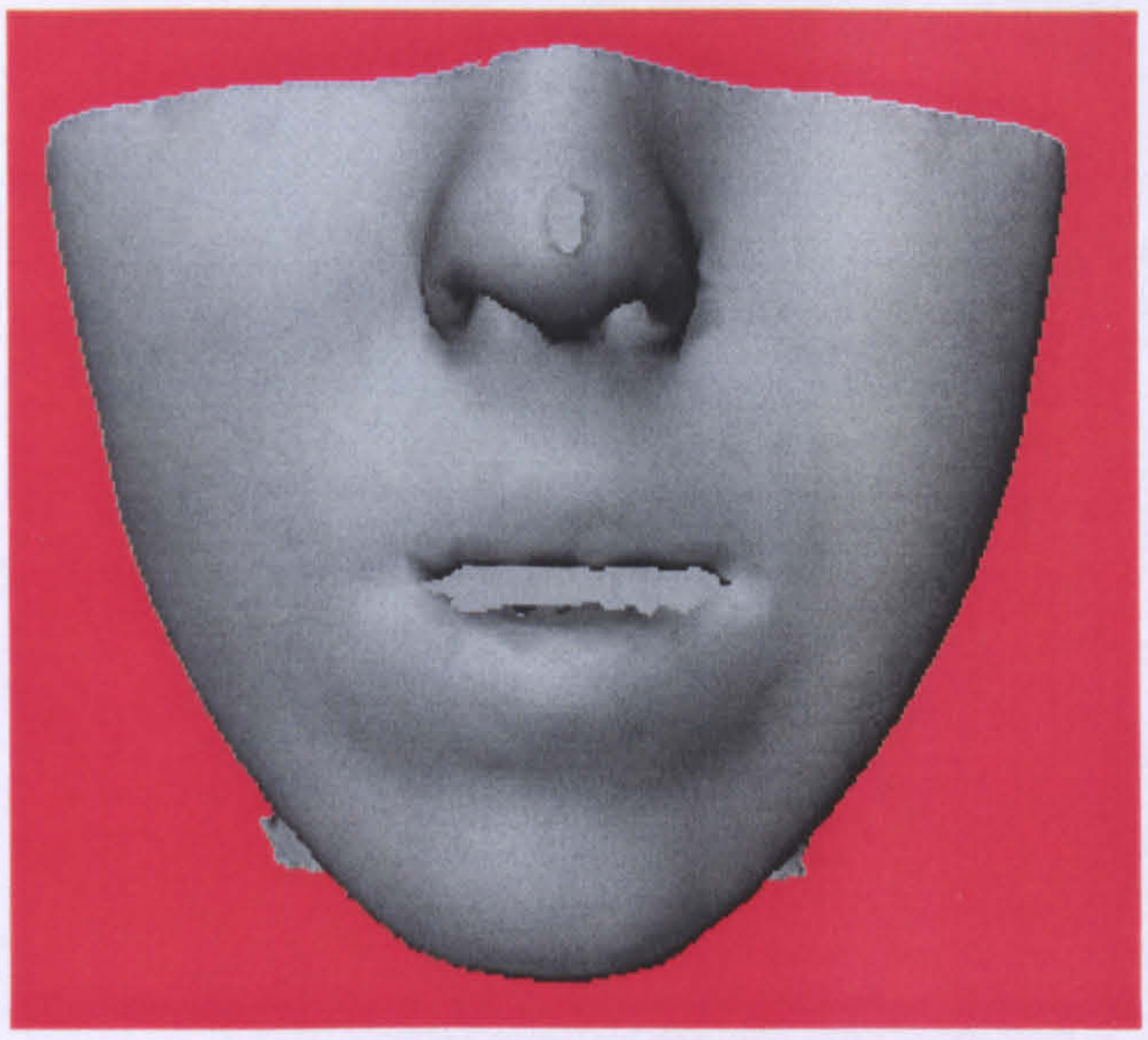
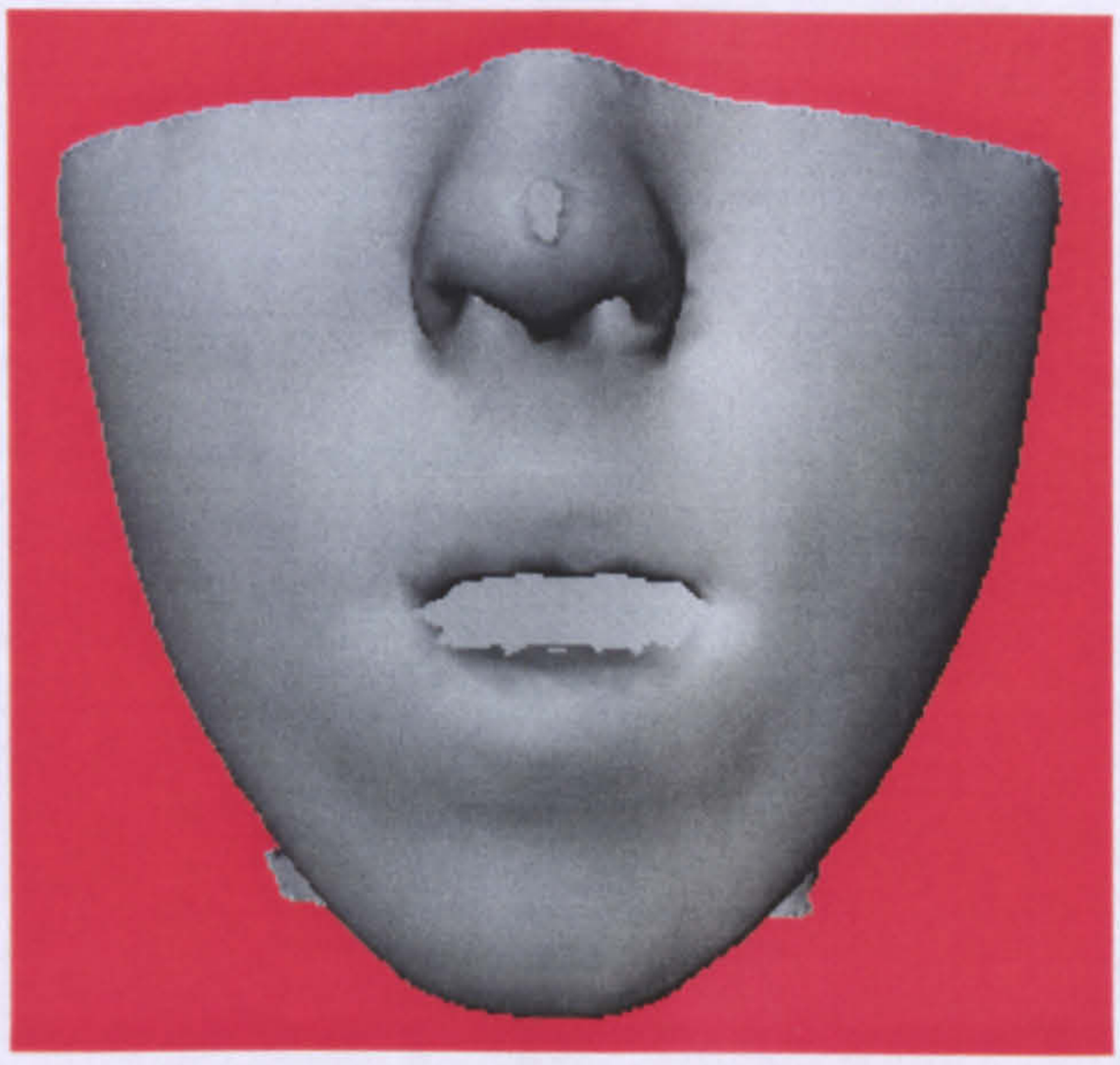


Figure 3.27 Lower jaw is rotated by various degrees to represent the opening of the mouth and the resulting facial expression is simulated.



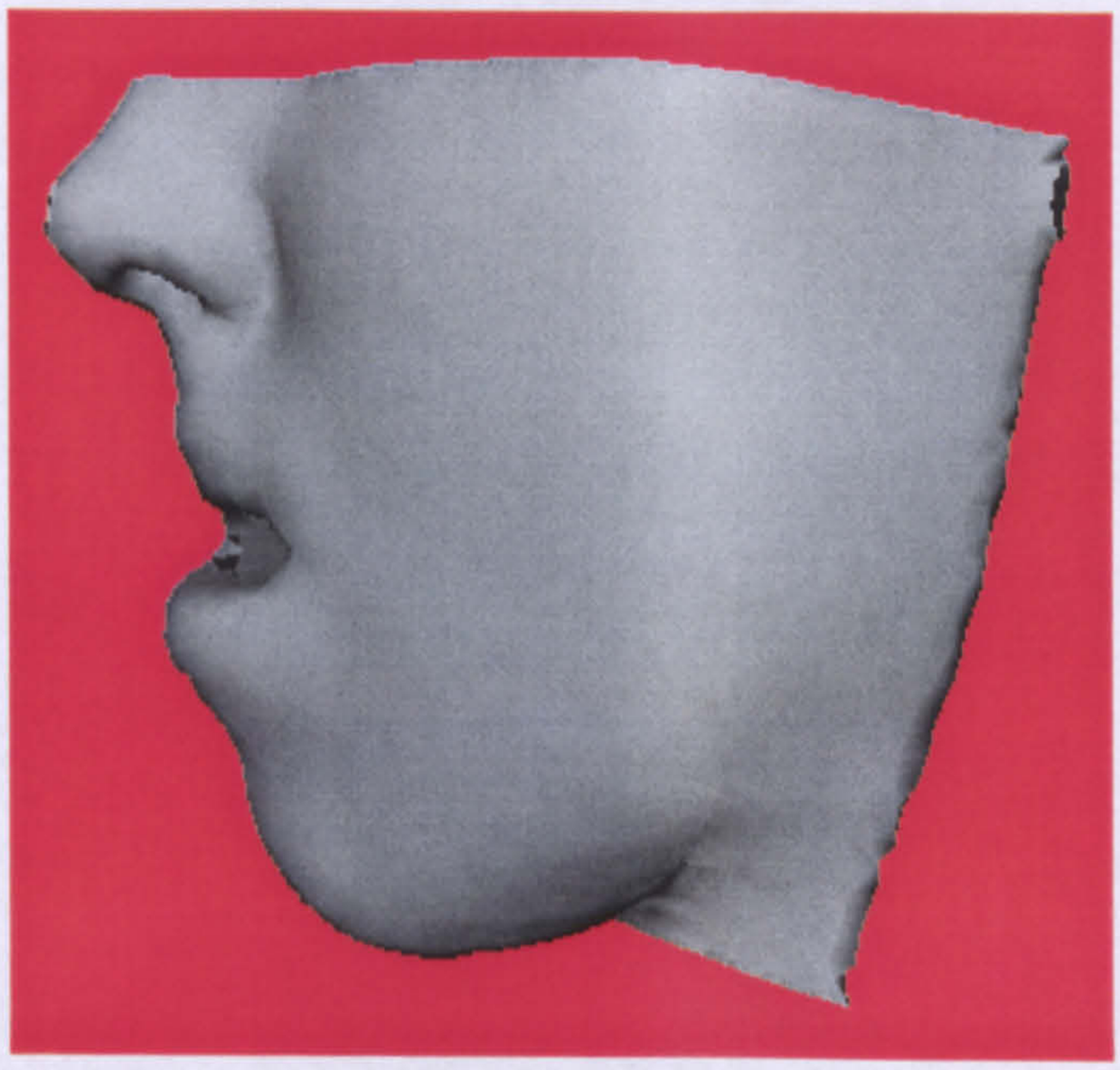


Figure 3.28. The mouth is opened more widely.

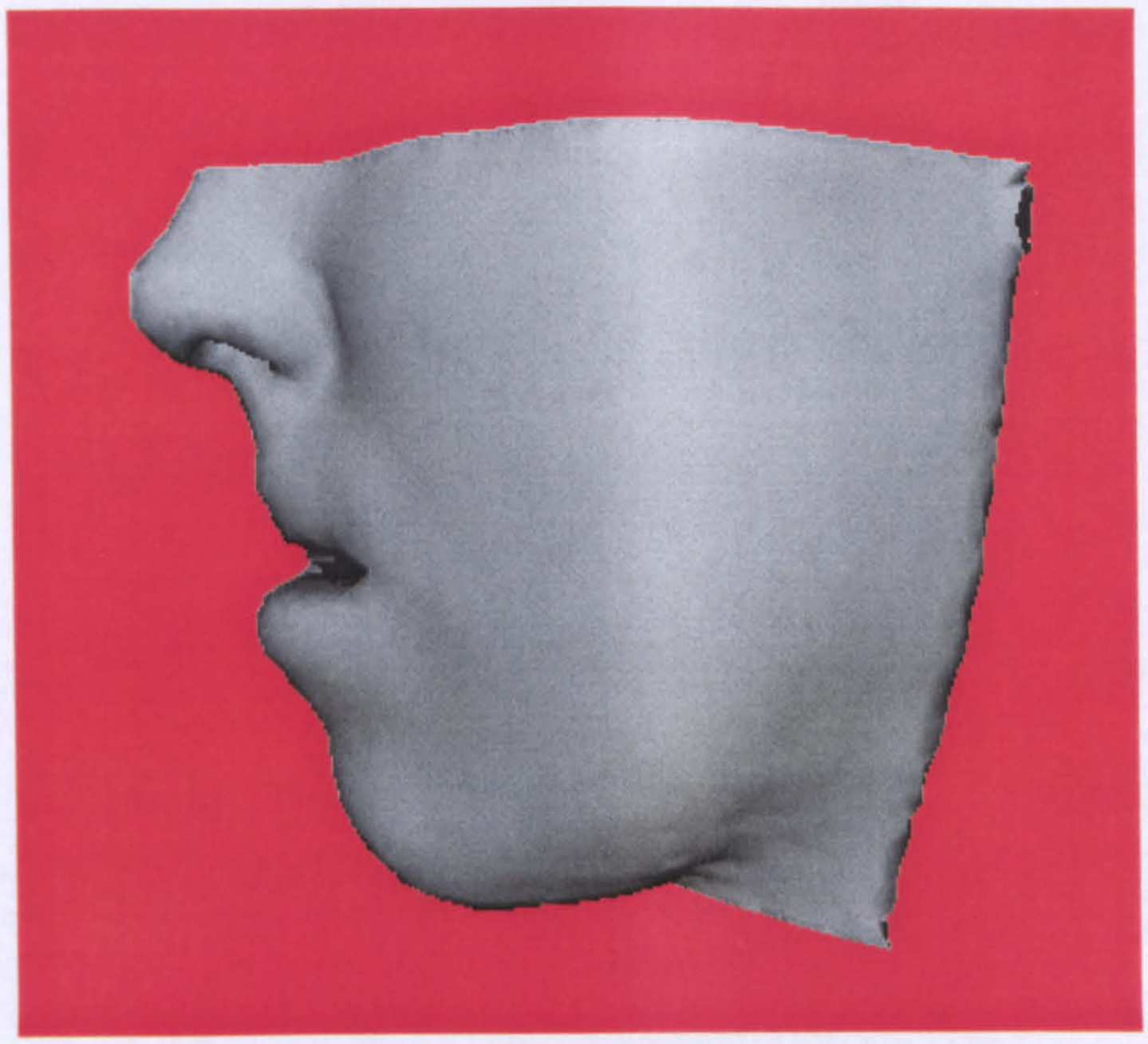


Figure 3.29 The open mount pose before the surgery.

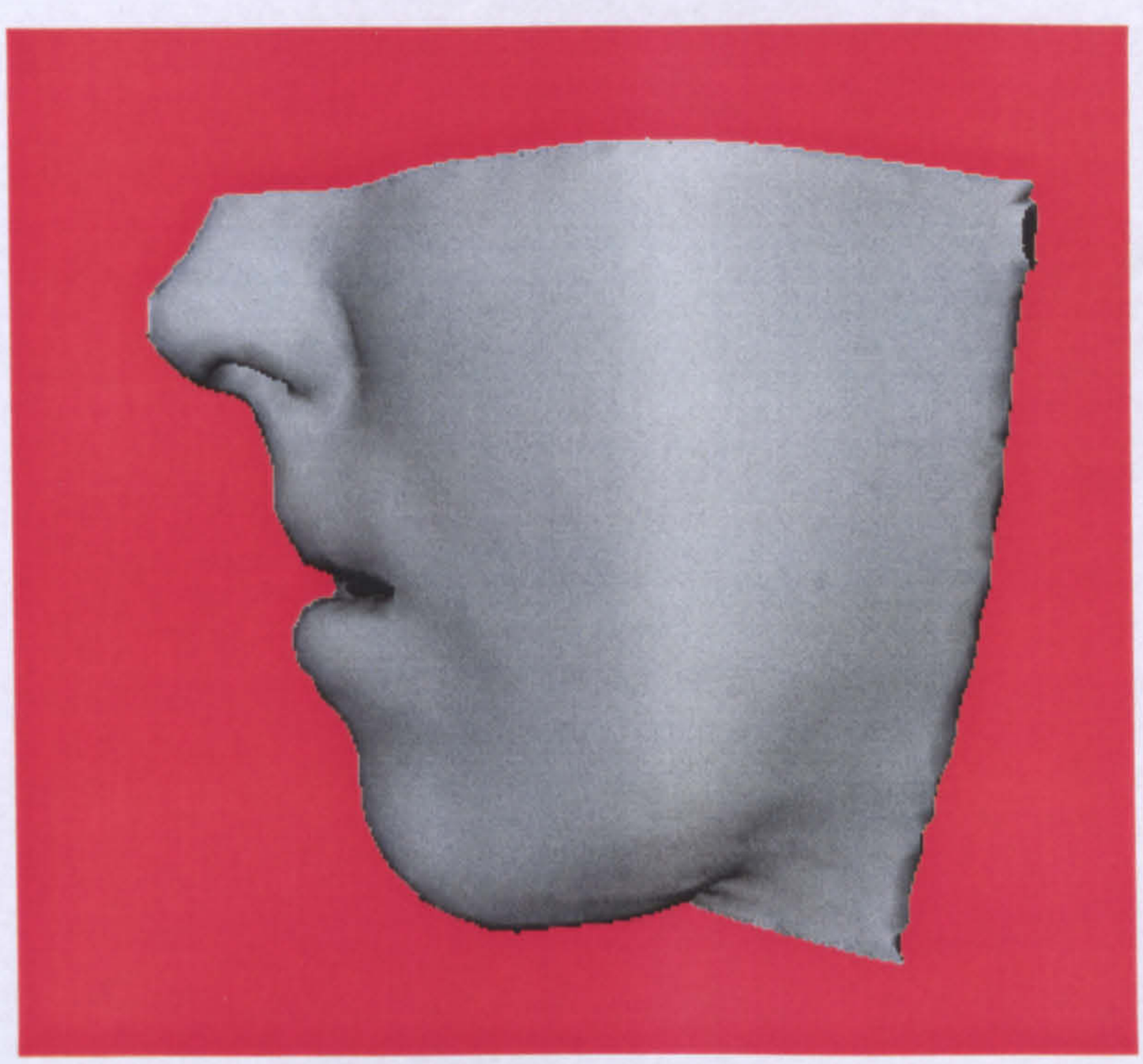


Figure 30 The open mount pose after the surgery.

Another experiment was carried out by rotating the lower jaw by various degrees. It must be noted that in the original medical image the lips are completely closed, as

shown in figure 3.5. In order to animate the opening of the mouth we cut the lips at the middle with a small line. This may cause an unrealistic appearance of the lips during the animation. Figure 3.27 and 3.28 show the mouth opening simulation from the frontal and side views. Figures 3.29 and 30 show the open mount before and after the surgery simulation, respectively.

3.6 Summary

In previous applications, the head model, consisting of tissue layers between face and bone surfaces, was generated using only one center point (assuming the face is a sphere). The tissue layers are represented with prism elements, which may overlap. In addition, some skin vertices were connected to wrong bone parts causing unwanted face deformation. These problems were addressed by assigning two center points in the determination of the tissue layers. An average tissue thickness was also found and was used for establishing the depth of the prism elements. In our model possible overlaps are minimized and correct intersection with accurate tissue thickness is achieved. The simulation algorithm then successfully predicts the soft tissue changes due to bone manipulations (Duysak et al. 2003). Using this model, simulation of facial expressions is also possible. The results, however, need to be verified by comparing them with the post-operation image. Although verification using real data was not done due to the lack of the post-operation images, the results were verified visually by the surgeons. The most difficult part in simulating soft tissue deformations, using mass-spring systems, is choosing appropriate simulation parameters. Given the correct parameter values (for the spring stiffness and damping) any type of tissue deformation can be simulated. There is, however, no straightforward way of determining these parameter values. In the next chapter we will develop a specific method that provides a means of selecting such parameter values.

CHAPTER 4

NEURAL NETWORK SYSTEM IDENTIFICATION

Mass-spring systems are an approximation to their real life counterparts. However, due to their simplicity to construct and economy to compute, they have been widely used in numerous applications of computer graphics, virtual reality and medical simulation. A key to achieving an accurate simulation result is to use appropriate system parameters, such as spring stiffness and viscosity. In practice, as these system parameters are not available, trial-and-error is the most common means for their determination. Although they are usually significantly nonlinear, for simplicity they are assumed to be linear in almost all existing systems, which often leads to unrealistic simulation results. In this chapter we employ neural networks to identify and determine, simulation parameters, which are nonlinear functions.

The determination of simulation parameters in mass-spring systems still remains a challenge. Appropriate parameters need to be assigned not only for reasons of accuracy but also for reasons of stability. Although there is a mathematical relation between simulation parameters, which will maintain stability, there is no

straightforward way of choosing simulation parameters, which will achieve accurate results.

In the literature such parameters are determined based on the outcome of simulations. Clearly this is very time consuming and nonlinear parameters leading to physically correct simulation are almost impossible to determine. Some researchers use intensity based approaches, which evaluates intensity values as spring stiffness. There is no physical foundation between static intensity values and nonlinear parameters representing dynamic deformation behavior. Some researchers use genetic algorithms to minimize a cost function leading to the identification of simulation parameters. This is again done based on an assumed constitutive model and only one parameter is identified at a time. Neural networks on the other hand have been successfully used in the identification of unknown systems as well as parameters in engineering applications.

Neural networks can be considered as a data processing system consisting of a large number of simple but highly connected elements capable of learning to do many tasks by training, which use input-output data. Since neural networks themselves are nonlinear black-box models, they are often trained to replace other unknown systems (system identification). If the structure of the system is known with some undetermined parameters, neural networks can also be used in identifying those parameters. In chapter 2 a dynamic structure for mass-spring systems was derived based on the explicit-Euler integration method. This structure is further investigated in this chapter to obtain a general mass-spring model, which includes unknown system parameters. An identification method is then developed to fit the mass-spring model and to identify its unknown coefficients. In this chapter we also explain the type of

neural network used in detail. A cerebellar model articulation controller (CMAC) neural network is used in our work for the following reasons. CMAC neural networks are extremely fast to converge compared to other networks. CMAC neural networks also require very little computation time, therefore they do not impose a substantial computational burden on the simulation system. They therefore are suitable for on-line applications.

4.1 The Cerebellar Model Articulation Controller (CMAC) Neural Networks

Anatomical and neurophysical studies of the brain have motivated a number of researchers to propose mathematical models that explain the information-processing characteristics of the cerebellum, which is the structure responsible for learning and execution of reflexive and voluntary motions (Albus 1972). Based on the known human memory and neuromuscular control principles, Albus (1972, 1975a 1975b, 1979) proposed a neural network algorithm called the cerebellar model articulation controller (CMAC). The CMAC is a feedforward neural network and can be considered as an associative memory performing two consecutive mappings from input space to output space.

Before giving a detail analysis of the CMAC neural networks we provide some of their attractive properties, which include:

- Good generalization. In many cases, the performance of neural networks is degraded when a given input is not part of the training data. In contrast, the CMAC neural network has better performance in this regard (Miller 1988). CMACs are said to have good generalization, because they generate reasonable output even when their input is not part of the training data.

- Good training properties. In comparison with other networks, CMAC training requires less iteration (Miller et al. 1990). Furthermore, it has been shown that for off-line training, the CMAC parameters converge to a least-squares solution.

- Easy software and hardware implementation. Realization of a CMAC network in software is easily accomplished even for large networks (Miller 1988). In addition, it has also been shown that CMAC algorithms are well suited for implementation using integrated circuits (Wen et al. 1996).

- Fast computation time. CMACs need fewer computations and less time for weight update and output generation than other types of neural network models (Miller 1988). Therefore CMACs are preferred in applications which are required to work in real-time.

4.1.1 The Operation of a CMAC Neural Network

A CMAC network is used to approximate a nonlinear function of the form:

$$\mathbf{y}_i = f(\mathbf{x}_i) \tag{4.1}$$

where $\mathbf{x}_i$ is a multidimensional input vector, and $\mathbf{y}_i$ is a multidimensional output vector.

A CMAC network is an associative neural network, in that the input $\mathbf{x}_i$ serves as an address key to memory locations whose contents are summed to form the network output. Figure 4.1 shows that a CMAC network performs the mapping from $\mathbf{x}_i$ to $\mathbf{y}_i$ in several stages. The individual stages are now considered in detail.

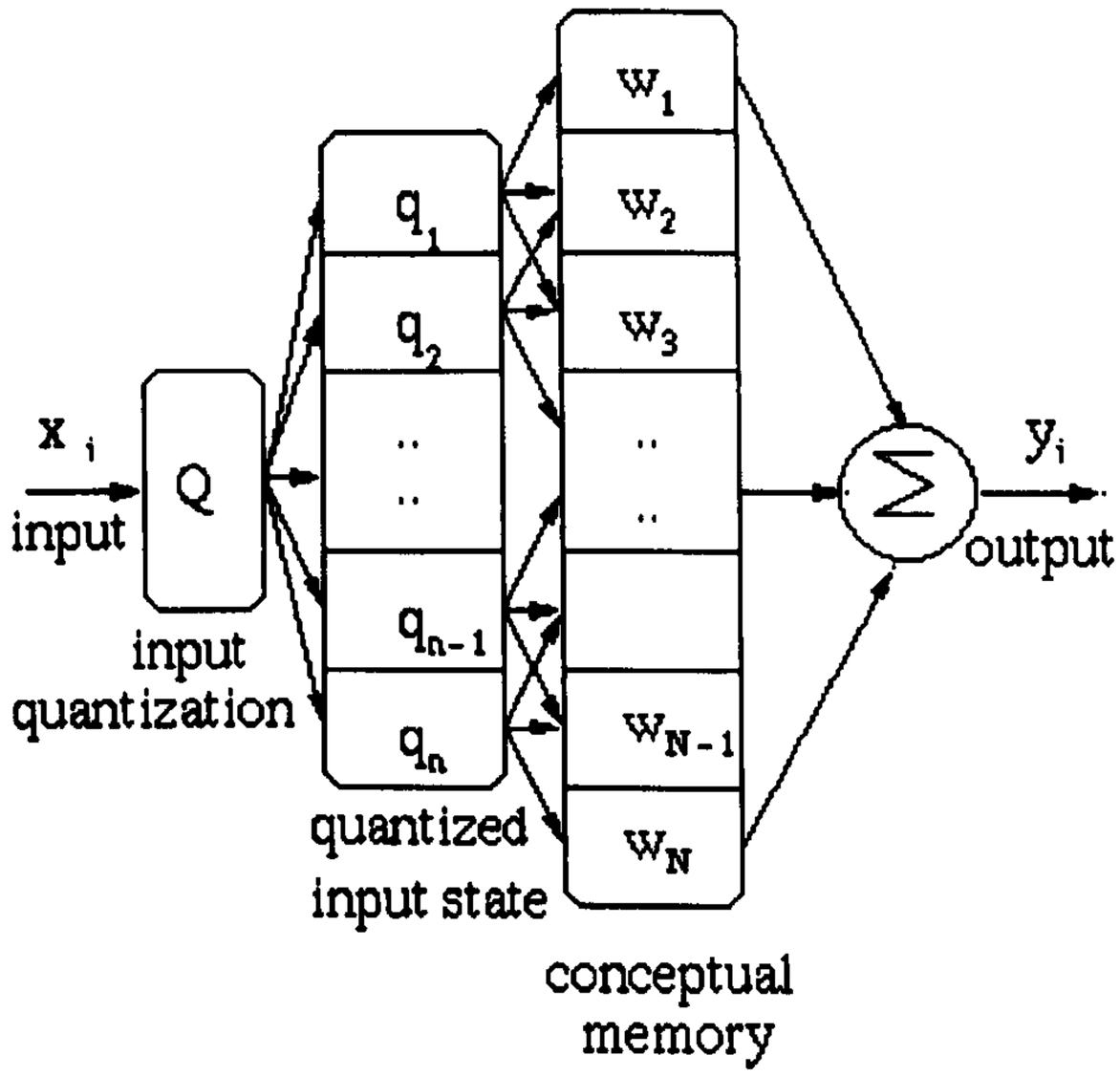


Figure 4.1 The operation of CMAC neural networks.

4.1.1.1 Input Quantization

For simplicity, this section considers the case where both the input x_i and the output y_i of the CMAC are scalars. It is assumed that x_i is bounded and the range of x_i is known. The input x_i is first quantized into one of n possible values, q_1 to q_n, which span the input space. The quantization levels can be fixed or variable. As an example, figure 4.2 shows a function that represents the input decoder (quantization) of Figure 4.1. The input is x_i and the output is the quantization level q_k :

$$q_k = Q(x_i : x_{\min}, x_{\max}, q_{\max}), \tag{4.2}$$

where k represents the quantization levels, $x_{\min}$ and $x_{\max}$ are the minimum and maximum values of the input, respectively, and $q_{\max}$ is the total number of available quantization levels.

As shown in figure 4.2, the distance between adjacent quantized levels can vary depending upon the range of expected input. It is reasonable to use a small distance

for the ranges of x, which are more likely to occur. For simplicity, in this thesis, the quantization resolution is fixed as:

$$r = \frac{x_{\max} - x_{\min}}{q_{\max}} \tag{4.3}$$

where r is the quantization resolution.

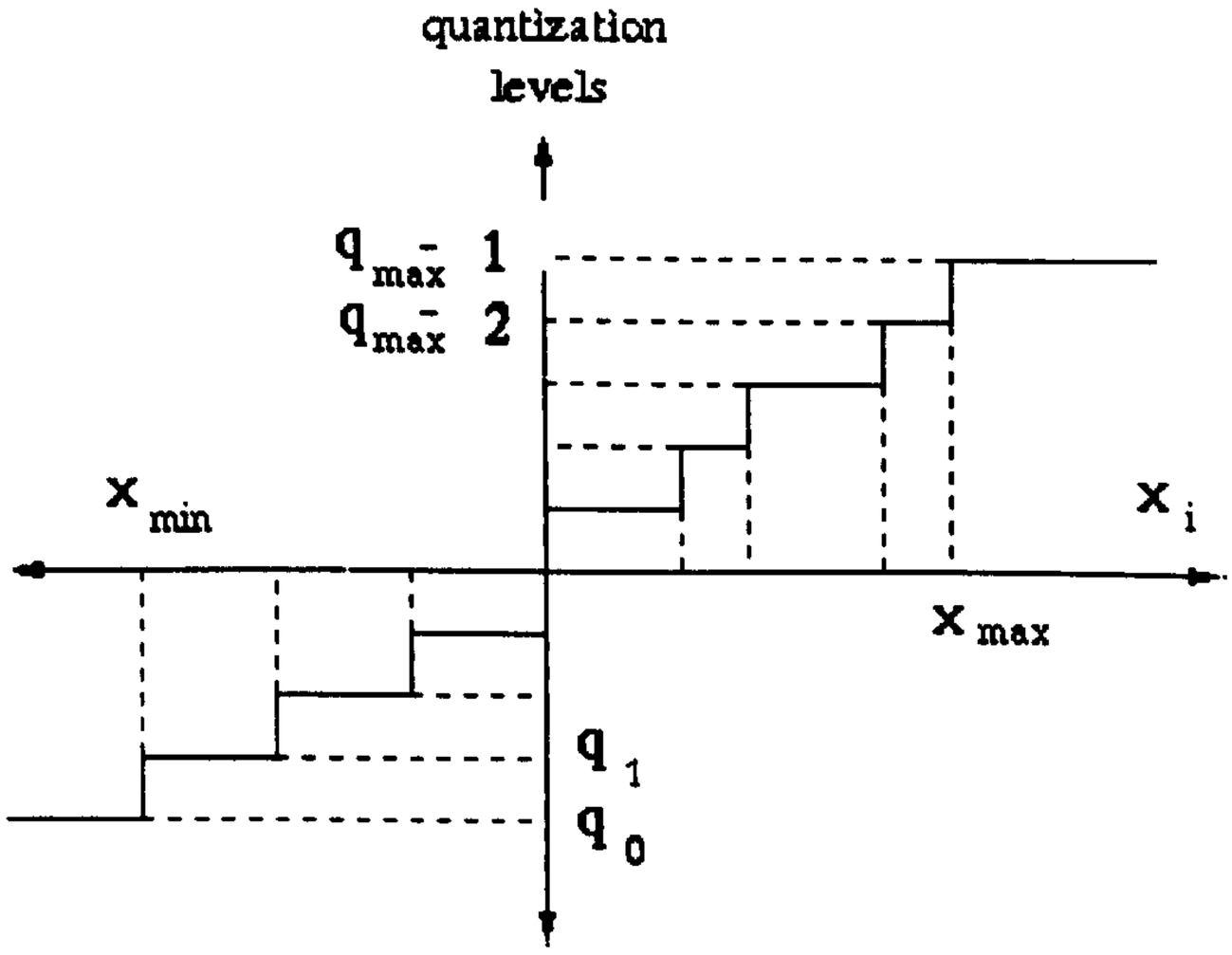


Figure 4.2 Input quantization.

For situations where the quantization resolution is variable, the resolution r is adjusted adaptively. References (Kim and Lin 1992) and (Kawamato et al. 1995) examine the use of adaptive quantization in CMAC neural networks. The quantization function in equation 4.2 is defined as:

$$q_k = Q(x_i) = \left\lfloor \frac{x_i - x_{\min}}{r} \right\rfloor \tag{4.4}$$

where $x_i \in [x_{\min}, x_{\max}]$ and $\lfloor p \rfloor$ is the largest integer less than the real number p. The function $Q(x_i)$ returns an integer q_k for the input x_i and q_k ranging from 0 to $(q_{\max} - 1)$.

4.1.1.2 Conceptual Memory and Memory Mapping

A CMAC neural network contains a conceptual memory whose contents are summed to form the CMAC output. In section 4.1.1.1, it was shown that each input x_i is represented by quantization level q_k. Quantization level q_k maps into A^* memory cells of the conceptual memory. A^* represents the number of memory cells that are activated at any time by a particular quantization level. Any two adjacent quantization levels activate memory cells that overlap by A^*-1. It follows that if there are $q_{\max}$ quantization levels, then the conceptual memory has:

$$N_C = q_{\max} + A^* - 1 \tag{4.5}$$

memory cells.

It is convenient to introduce a row vector θ that indicates which memory cells in the conceptual memory are activated by a given quantization level q_k. The number of columns in θ is equal to the size of the conceptual memory. For example, using $q_{\max} = 9$ and choosing the number of active weights as $A^* = 4$, equation 4.5 shows that the conceptual memory contains a total of twelve memory locations. The vector θ associated with each quantization levels q_k in this example is shown in table 4.1.

By observation, the sum of the elements in θ_k is equal to A^*. In addition, from table 4.1, whenever the input changes by one quantization level, one element is dropped and another is added. This feature provides the CMAC neural network with a good generalization property. That is, inputs that have adjacent quantization levels are likely to produce similar outputs. When considering the multidimensional input vector x_i, it is useful to represent addresses of the conceptual memory using lower case

letters. Each element of θ_k in table 4.1, can then be represented by a unique letter representing its position in θ_k. For example, the vector $\theta_1{}^T$ is represented as:

$$\theta_1{}^T = [1\ 1\ 1\ 1\ 0 \ldots 0] = [a\ b\ c\ d\ 0\ \ldots 0]. \tag{4.6}$$

1	1 1 1 1 0 0 0 0 0 0 0 0
2	0 1 1 1 1 0 0 0 0 0 0 0
3	0 0 1 1 1 1 0 0 0 0 0 0
4	0 0 0 1 1 1 1 0 0 0 0 0
5	0 0 0 0 1 1 1 1 0 0 0 0
6	0 0 0 0 0 1 1 1 1 0 0 0
7	0 0 0 0 0 0 1 1 1 1 0 0
8	0 0 0 0 0 0 0 1 1 1 1 0
9	0 0 0 0 0 0 0 0 1 1 1 1

Table 4.1 The row vector θ indicates which cells in the conceptual memory are activated by the quantization level q_k.

Using this approach table 4.1 is rearranged by eliminating the locations that are not activated. Memory cells not activated by a given quantization level are represented by zeros in table 4.1. Table 4.2 shows the addresses of the quantization levels.

As shown in table 4.2, the twelve memory addresses in the conceptual memory are labeled as a, b, c, ..., l. The weights stored in the memory cells are contained in a column vector w

$$w = [\ w_a, w_b, w_c, w_d, \ldots, w_l\ ] \tag{4.7}$$

q_k	address
q_1	a b c d
q_2	e b c d
q_3	e f c d
q_4	e f g d
q_5	e f g h
q_6	i f g h
q_7	i j g h
q_8	i j k h
q_9	i j k l

Table 4.2 Conceptual memory addresses activated by the quantization levels q_1 through q_9 for A^*=4.

As mentioned earlier, consecutive quantization levels q_k and q_{k+1} activate memory cells that overlap by $A^* - 1$. In general, the amount of the overlap is $A^* - H_{ij}$, where H_{ij} is the Hamming distance between quantization levels q_i and q_j and is defined as (Albus 1972):

$$H_{ij} = \sum_{k=1}^{n} | \theta_{ij} - \theta_{ik} | \tag{4.8}$$

where θ_{ik} is the k^{th} element of θ_i, n = 1,2,..., N_C and j = 1,2,..., N_C. If two inputs have a small Hamming distance, there will be more overlap. For example, when A^*=4 and $H_{k,k+1} = 1$, there is an overlap of $A^* - H_{k,k+1} = 3$ between the memory

cells activated by the quantization levels q_k and q_{k+1}. If two inputs are separated by a large Hamming distance, there will be less overlap between their corresponding association cells. No overlap occurs if the Hamming distance is larger than A^*. This fact gives CMAC neural networks their ability to dichotomize inputs separated by a large Hamming distance.

4.1.1.3 Calculation of the Output

It has been shown that any given input x_i activates A^* memory locations in the conceptual memory. The weights stored in these memory locations are summed to produce the output:

$$y_i = \theta_i w \tag{4.9}$$

where i=1,2,..., $q_{\max}$, and w is a $N_C \times 1$ column vector. As an example, assume that the input excites the second quantization level, q_2 in table 4.2. In this case, the activated weights are w_e, w_b, w_c, w_d and the output y_2 is the sum of these weights. The next section shows how we obtain the values of the weight vector w.

4.1.2 CMAC Training

In order to determine the weight vector w the response of the CMAC network is compared to a desired response for a number of different inputs x_i. The weights are adjusted so that the output of the CMAC network approximates the desired output. In the training algorithm presented by Albus, the network is presented with a series of m data pairs (x_k, d_k) where x_k is the input and d_k is the desired response. For the k^{th} training sample, the error between the desired output d_k and the network output y_k is $e(k) = d_k - y_y$. Albus proposed the iterative training law (Albus 1975a 1975b):

$$w(i+1,k) = w(i,k) + \beta \frac{\theta \times e(k)}{A^*} \tag{4.10}$$

where β is the learning factor, $k = 1,2,...,m$ and i is the iteration. This method evenly distributes the error $e(k)$ to the weights that are excited by the k^{th} input.

The designer must carefully choose both the numbers of data points and their values. It is not necessary to provide a training pair (x_k, d_k) at every point in the input space because of the network's generalization property. However, it may be necessary to repeat the training several times if two inputs are separated by small Hamming distance to produce different outputs. This may require a large difference in some weights that are not in common for those two inputs. The training iterations are completed when the magnitude of $e(k)$ falls below a specified value for each data pair (x_k, d_k).

The final weight vector w is determined by many parameters including the number of active weights A^*, the size of the CMAC memory N_C, and the learning factor β. A large conceptual memory typically requires more iteration to complete the training. This can be overcome by choosing a smaller value of A^*, but this may cause insufficient generalization. On the other hand, if A^* is large and N_C is small, CMAC may fail to converge using the training law shown in equation 4.10. A large value of A^* causes the CMAC to lose its generalization ability. Albus recommended that $N_C > 100\,A^*$. In (Xu et al. 1994), it was shown that $N_C \cong 10A^*$ also works well. There are several ways to train a CMAC network, but only two of these methods are considered here.

4.1.2.1 The Batch Algorithm

In this algorithm, the weights are updated using the following rule:

$$w(i+1) = w(i) + \sum_{k=1}^{m} \frac{\beta}{A^*} \theta_k^T (d_k - \theta_k w(i))$$

$$= \frac{\beta}{A^*} \Theta^T (d - \Theta w(i)) \qquad (4.11)$$

where $\Theta = [\theta_1^T, \theta_2^T, ..., \theta_{q_{\max}}^T]$ and $d = (d_1, d_2, ..., d_{m-1}, d_m)$. The selection of the range of the learning factor was examined in (Duysak 1997), which also showed that this algorithm always converge to a LMS solution for a specified range of β. If β is one, the process is called one-shot error correction and the training algorithm does not always converge.

4.1.2.2 The Non-Batch Algorithm

In this algorithm the weights are updated each time an input is applied. The following rule is used:

$$w(i+1,k) = w(i,k) + \frac{\beta}{A^*} \theta_k^T (d_k - \theta_k w(i,k)) \qquad (4.12)$$

where $k = 1,2,...,m$ is the sample number and i represents the iteration. For iterative learning, it was shown (Duysak 1997) that the non-batch CMAC learning converges only to a limit cycle whose radius is determined by the learning factor β. In this case a small β leads to a smaller radius of the limit cycle, but more iterations are required. This method has the advantage that any learning factor in the range $0 < \beta < 2$ will cause the algorithm to converge. Although the CMAC non-batch algorithm converges only to a limit cycle, this method provides the opportunity to train a CMAC neural network on-line.

4.1.3 The Multidimensional Input Case

The CMAC network described in section 4.1.1 is easily modified to account for multidimensional input signals. For example, suppose the input x_i is a vector with two components x_i^1 and x_i^2. In general, each component is quantized using a different rule. For example, consider the case where x_i^1 is quantized into one of five levels q_k^1, while x_i^2 is quantized into one of seven levels q_k^2.

As described in section 4.1.1, for multidimensional inputs it is desirable to specify addresses in the conceptual memory using letters. In this example, addresses activated by the quantization levels q_k^1, are indicated by upper case letters, while addresses activated by q_k^2 are indicated by lower case letters. For simplicity, we use $A^* = 4$ for both inputs. Tables 4.3 and 4.4 show the addresses activated in the conceptual memory by the quantization levels q_k^1 and q_k^2.

q_k^1	address
q_1	ABCD
q_2	EBCD
q_3	EFCD
q_4	EFGD
q_5	EFGH

Table 4.3 The addresses of weights for input x_i^1.

q_k^2	address
q_1	a b c d
q_2	e b c d
q_3	e f c d
q_4	e f g d
q_5	e f g h
q_6	i f g h
q_7	i j g h

Table 4.4 The addresses of weights for input $x_i{}^2$.

If the CMAC is implemented using two conceptual memories, each associated with different inputs; a total of 18 weights are needed. For a given input $x_i = (x_i^1, x_i^2)$, it is necessary to address $A_1^* + A_2^* = 8$ memory locations and sum the corresponding weights to produce the output. In order to decrease the number of cells that must be addressed, as well as to decrease the number of addition operations required, Albus proposed to concatenate the two separate conceptual memories into a single large conceptual memory. The disadvantage of this approach is that it requires a large number of memory locations.

As seen from tables 4.3 and 4.4, if input one and input two both excite the first quantization level, then the corresponding address of the weights is [AaBbCcDd], which is the concatenation of [ABCD] and [abcd]. If input one excites the first

quantization level while input two excites the third quantization level, then the corresponding weight address is [AeBfCcDd]. The complete mapping is shown in table 4.5.

ijgh	7	AiBfCgDh	EiBjCgDh	Ei,Fj,Cg,Dh	EiFjCgDh	EiFjGgHh
ifgh	6	AiBfCgDh	EiBfCgDh	EiFfCgDh	EiFfGgDh	EiFfGgHh
efgh	5	AeBfCgDh	EeBfCgDh	EeFfCgDh	EeFfGgDh	EeFfGgHh
efgd	4	AeBfCgDd	EeBfCgDd	EeFfCgDd	EeFfGgDd	EeFfGgHd
efcd	3	AeBfCcDd	EeBfCcDd	EeFfCcDd	EeFfGcDd	EeFfGcHd
ebcd	2	AeBbCcDd	EeBbCcDd	EeFbCcDd	EeFbGcDd	EeFbGcHd
abcd	1	AaBbCcDd	EaBbCcDd	EaFbCcDd	EaFbGcDd	EaFbGcHd
-	-	1	2	3	4	5
-	-	ABCD	EBCD	EFCD	EFGD	EFGH

Table 4.5 Conceptual memory found by address concatenation.

As expected, while 18 weights are needed to implement two individual mappings, 20 weights are necessary to implement a single mapping from the input quantization levels to an address in the conceptual memory. In general, if each element of the input vector has R quantization levels, then an n-dimensional input vector has R^n points that must be mapped to the conceptual memory. It follows that the number of required memory locations is of the same order of magnitude as R^n. For many applications, the size of the conceptual memory is too large to be physically implemented. Albus solved this problem by using hash-coding, where the large conceptual memory is mapped into a smaller physical memory. Readers are referred to (Wang et al. 1996) for details of hash-coding method.

Instead implementing the hash-coding technique, which causes collision problems, we employed an address mapping formula.

4.1.5 An Address Mapping Formula

In order to avoid problems associated with collisions, in this thesis we do not use hash coding. It follows that the conceptual and physical memories are identical. This section presents an equation that maps a multidimensional input into addresses of activated weights in the memory. The algorithm considered was first introduced by (Wen et al. 1996):

$$A_J = [1 + \left\lceil \frac{q^1_{\max} - J}{A^*} \right\rceil][1 + \left\lceil \frac{q^n_{\max} - J}{A^*} \right\rceil]\left\lceil \frac{q^1_k - J}{A^*} \right\rceil + \ldots$$

$$+ [1 + \left\lceil \frac{q^n_{\max} - J}{A^*} \right\rceil]\left\lceil \frac{q^{n-1}_k - J}{A^*} \right\rceil + \left\lceil \frac{q^n_k - J}{A^*} \right\rceil \qquad (4.13)$$

where A_J is an address in the J^{th} memory bank and $\lceil p \rceil$ is the largest integer less than the real number p.

Equation 3.13, divides the CMAC memory into A^* memory banks based on the values of J, where J=1,2,..., A^*. A given quantization level q^i_k activates exactly one address A_J given by equation 3.13 in each of the A^* memory banks. The A^* activated weights are then summed to form the CMAC output.

q_k^1	q_k^2	A_1	A_2	A_3	A_4
1	1	0	0	0	0
1	2	1	0	0	0
1	3	1	1	0	0
1	4	1	1	1	0
1	5	1	1	1	1
1	6	2	1	1	1
1	7	2	2	1	1
2	1	3	0	0	0
2	2	4	0	0	0
2	3	4	1	0	0
2	4	4	1	1	0
2	5	4	1	1	1
2	6	5	1	1	1
2	7	5	2	1	1
3	1	3	3	0	0
3	2	4	3	0	0
3	3	4	4	0	0
3	4	4	4	1	0
3	5	4	5	1	1
3	6	5	4	1	1
3	7	5	5	1	1
4	1	3	3	2	0
4	2	4	3	2	0
4	3	4	4	2	0
4	4	4	4	3	0
4	5	4	4	3	1
4	6	5	4	3	1
4	7	5	5	3	1
5	1	3	3	2	2
5	2	4	3	2	2
5	3	4	4	2	2
5	4	4	4	3	2
5	5	4	4	3	3
5	6	5	4	3	3
5	7	5	5	3	3

Table 4.6 Addresses of the memory locations obtained using equation 3.13.

The mapping equation is now explained with an example. For the two-dimensional input case examined in section 4.1.1, the addresses of the conceptual memory are given in table 4.5. Because A^*=4, there are four memory banks A_1 through A_4. Memory banks A_1 and A_2 have six weights which are addressed by 0 to 5, while memory banks A_3 and A_4 have four weights addressed by 0 to 4. Using the equation 4.13, for the same number of quantization levels and number of active weights, the addresses of memory locations in the conceptual memory are given in table 4.6.

Recall that for the quantization level $q_1 = q_2 = 1$, the addresses given in table 4.5 were [AaBbCcDd]. These addresses are found using the equation 4.13 as [0,0,0,0]. For quantization level $q_1 = q_2 = 2$, the addresses obtained by address concatenation are [EaBbCcDd], for the same quantization level the addresses obtained using equation 4.13 are [1,0,0,0]. As a result, the size of the conceptual memory obtained by address concatenation is the same as the size of the conceptual memory found by the equation 4.13.

4.2 System Identification

Neural networks have been widely used in the identification of completely unknown systems (Sinha 2000, Lu et al. 2003, Almedia and Voit 2003). In some cases, the structure of a dynamic process can be obtained by using first principles of physics but some unknown system parameters may exists. If the system structure is known, specific models can be developed to identify the unknown system elements. Narendra (1996) proposed neural network identification models for known and unknown structures for general cases. A more specific parameter identification model for a mass-spring system is proposed by Nurnberger et al. (1998, 1999, 2001). Based on

both studies we have developed a parameter identification method for determining the coefficients of the mass-spring systems examined in chapter 2, where a dynamic model is given by figure 2.4 This model includes unknown parameters such as spring stiffness and damping. We briefly describe the method developed by Neirnberger in the next section before describing the new method.

4.2.1 Previous Methods

There have been several investigations in the literature for the identification of simulation parameters of mass-spring systems (Louchet et al. 1995, Koch et al. 1996, Joukhadar and Laugier 1997, Ishikawa et al. 1998, Bhat et al. 2003). None of these, however, provides a model or detailed information on how the identification was carried out. The most detailed work on this subject was reported by Nurnbergel et al. (1998, 1999, 2001) who developed a problem specific neural network model and identification method.

Their method is illustrated here in figure 4.3 using the basic structure of mass-spring systems where two mass-points are connected by a spring. The spring between the two mass-points is considered to have three dynamic elements representing spring stiffness, damping and total spring force. Each point is also thought to have three dynamic parts generating acceleration, velocity and position. The model shown in figure 4.3 can best be understood by analyzing the dynamic model developed in chapter 2 and shown in figure 2.4.

Each dynamic part of figure 2.4 is replaced by a neural network model whose task is to learn the dynamic it replaces. The neural network given by N_{fs} is replaced with the dynamic parts responsible for generation the spring forces due to the spring

stiffness, therefore this neural network learns the spring stiffness. Similarly, N_{fd} learns the spring damping. Using particular activation functions these coefficients are converted into forces. The total spring force is represented by N_F. The neural network N_A is used for the acceleration whereas, N_V gives the velocity. The positions of the mass-points are used to train the neural networks in their model, which has a total of six neural networks.

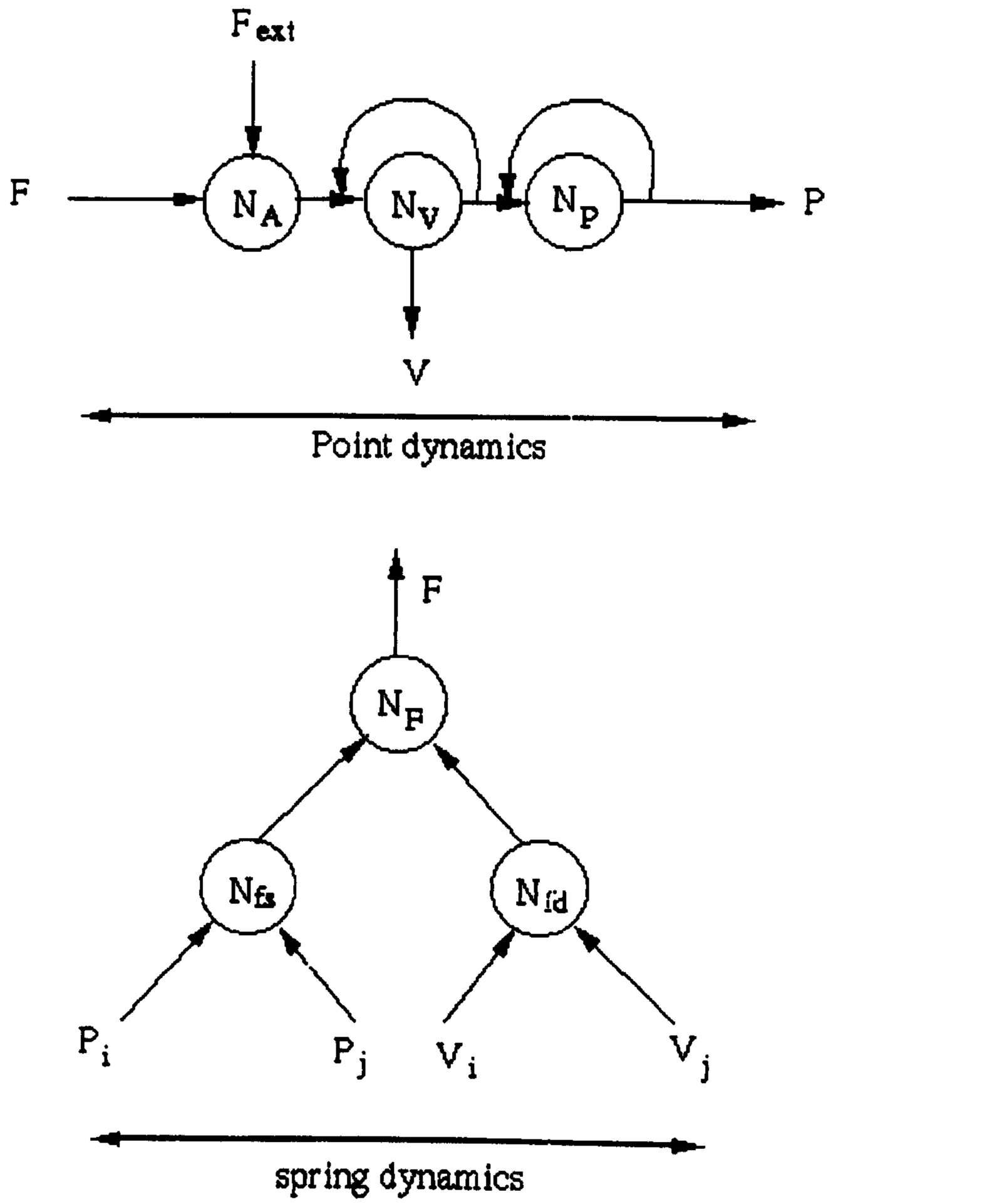


Figure 4.3 Nurnberger's identification model.

This caused some problems. The determination of the teacher signal is very difficult, if not impossible, for each neuron. This is because the only accessible information is the position. Error accumulation causes convergence problems since the outputs of the neural networks are used to feed other neural networks. In addition portability is not guaranteed if the position neural network model, N_P, is trained. As indicated in their latest work (Nurnberger et al. 2001), the proposed network does not converge to the original parameters but minimizes the error metric. This may cause significant problems if the trained neural network model is used in different applications where the input spectrum is different from the training data.

We, therefore, develop a new identification model that is specific to mass-spring systems. This model aims to reduce the number of neural networks used, and addresses the converge and portability problem along with a wider input space for training data.

4.2.2 A New Parameter Identification Model

Our model assumes that most of the nonlinearities come from the spring dynamics where the spring parameters are unknown. There exist many well-developed methods, given in Chapter 2 section 2.1.2, for calculating point dynamics. Therefore it is logical to focus on spring dynamics. In addition, the only unknown parameter in point dynamics is m (the mass of each point), which can be determined using various methods. The forces going into point dynamics (figure 2.4) are forces from the environment ($\mathbf{F}_{ext}$) and forces generated by the springs themselves ($\mathbf{f}_K$ and $\mathbf{f}_D$). A typical mass-spring system force dynamics are then given by the following equation. The total force exerted on each mass-point at time instant (k) is:

$$\mathbf{f}_s(k) = \mathbf{f}_K(k) + \mathbf{f}_D(k) + \mathbf{F}_{ext}(k)\,. \tag{4.14}$$

Since the external force is assumed to be known, this force can be added into the point dynamics leaving the unknown spring forces. The remaining forces are considered as spring forces, which are expressed as follows:

$$\begin{aligned}\mathbf{f}_s &= \mathbf{f}_K(k) + \mathbf{f}_D(k) \\ &= K * dr * \mathbf{u}_K + D * dv * \mathbf{u}_D \end{aligned} \tag{4.15}$$

where $\mathbf{f}_K(k)$ and $\mathbf{f}_D(k)$ are nonlinear spring forces due to spring stiffness and damping. The formula governing spring dynamics, given by equation 2.4 and 2.7 in chapter 2, can be represented in the following nonlinear form:

$$\mathbf{f}_s = K(dr) * dr * \mathbf{u}_K + D(dv) * dv * \mathbf{u}_D \tag{4.16}$$

where $K(dr)$ and $D(dv)$ are nonlinear functions that represent spring stiffness and spring damping, respectively. The known values $\mathbf{u}_K$ and $\mathbf{u}_D$ are vectors associated with spring stiffness and damping given in chapter 2 by equations 2.2 and 2.6.

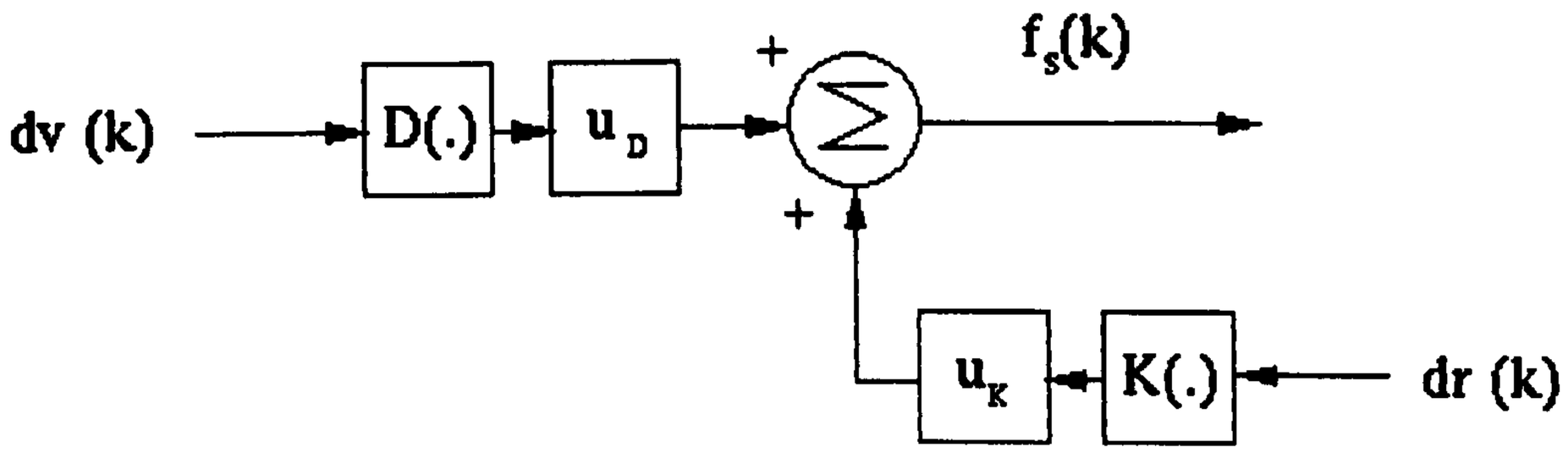


Figure 4.4 Forces generated by springs.

The spring dynamics, in terms of spring forces, is then given by figure 4.4. The inputs to the dynamics are examined in the next section. Equation 4.16 can then be expressed in the following representation using neural networks:

$$\hat{\mathbf{f}}_s = N_K(\cdot) * \mathbf{u}_K + N_D(\cdot) * \mathbf{u}_D \tag{4.17}$$

where neural networks $N_K(\cdot)$ and $N_D(\cdot)$ are used to learn nonlinear parameter functions $K(dr)$ and $D(dv)$, respectively. The forces generated by the springs can then be given using the neural network representations shown in figure 4.5.

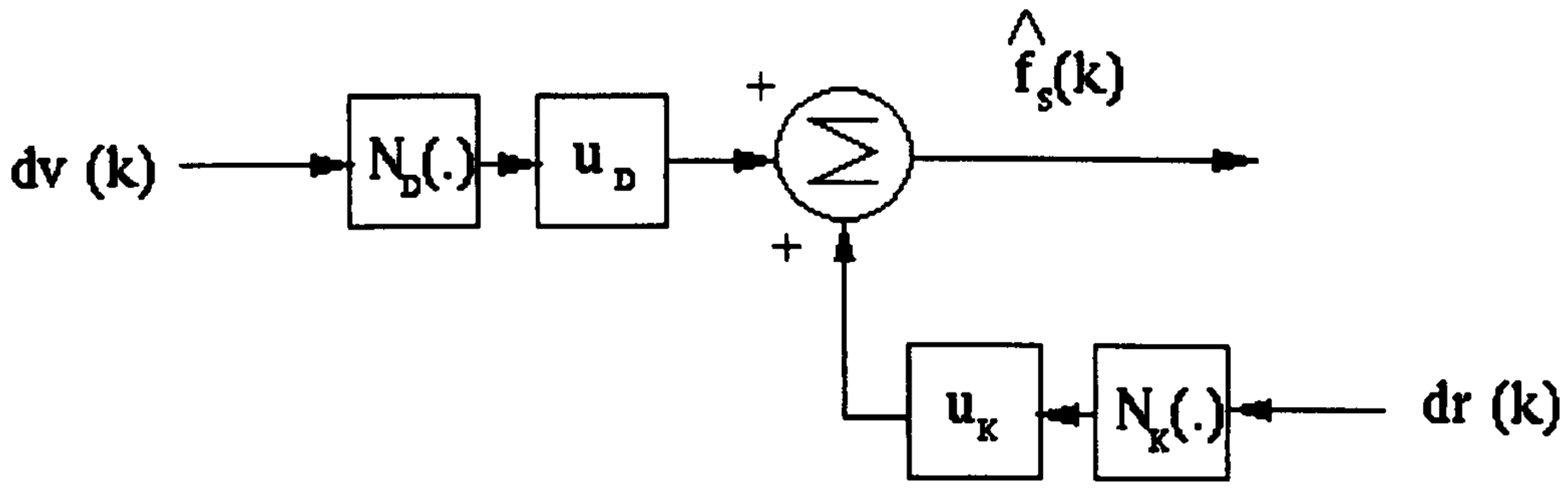


Figure 4.5 The neural network representation of the spring forces.

The next step is to establish proper input-output signals for training the two neural networks.

4.2.2.1 Input-Output Data

In previous applications positions and velocities were chosen as inputs to neural networks. In our work we use a different approach. Since spring force is driven by the changes of the velocity and the length of springs (see Chapter 2), the inputs are therefore defined to be:

$$dr = \left|\mathbf{X}_j - \mathbf{X}_i\right| - r_0 \tag{4.18}$$

where dr is the length change of the spring and

$$dv = \left|\mathbf{V}_j - \mathbf{V}_i\right| \tag{4.19}$$

where dv is the velocity change across the spring. Here, $\mathbf{X}$ and $\mathbf{V}$ define the current position and velocity of each mass-point and r_0 is the original spring length.

Using positions and velocities as inputs, the neural network model will have a very large but sparse input space, which will result in poor learning. This will also limit the generalization abilities of the neural networks used. A wider input spectrum for the neural networks can be established by using changes in the spring length and velocity. This solution will also address the portability problem as well. Positions of mass-points, in our method, do not effect the input space as much as in previous methods. Therefore trained networks can be used in different applications, where similar types of springs are used.

Since the point dynamics are calculated using one of the well-known techniques presented in chapter 2, the choice of the output (teacher signal) is also different in our model. The output signal is the total spring force given as the sum of two forces from the spring stiffness and damping (see equation 4.15). This force can be determined in terms of known variables using the dynamic structure shown in figure 2.4. The output for the next time step is:

$$\mathbf{X}(k+1) = \mathbf{X}(k) + dt \times \mathbf{V}(k+1). \tag{4.20}$$

The velocity for the same instance is:

$$\mathbf{V}(k+1) = \mathbf{V}(k) + \frac{dt}{m}\mathbf{f}_s(k). \tag{4.21}$$

Equations 4.20 can be rearranged to give:

$$dt \times \mathbf{V}(k) = \mathbf{X}(k) - \mathbf{X}(k-1). \tag{4.22}$$

Equation 4.20 can be rearranged by substituting equation 4.21, 4.22 and 4.14:

$$\mathbf{X}(k+1) = \mathbf{X}(k) + [\mathbf{X}(k) - \mathbf{X}(k-1)] + \frac{dt}{m}[\mathbf{f}_s(k) + \mathbf{F}_{ext}(k+1)]. \tag{4.23}$$

The above equation 4.23 is then rearranged once again to obtain the total spring forces in terms of the positional changes and external forces as:

$$\mathbf{f}_s(k) = \frac{m}{dt^2}[\Delta\mathbf{X}(k+1) - \Delta\mathbf{X}(k)] - \mathbf{F}_{ext}(k+1) \tag{4.24}$$

where $\Delta\mathbf{X}(k+1) = \mathbf{X}(k+1) - \mathbf{X}(k)$ and $\Delta\mathbf{X}(k) = \mathbf{X}(k) - \mathbf{X}(k-1)$ represent the positional changes of each mass-point.

As the above formulation (4.24) reveals, the external forces are included in the point dynamics instead of the spring dynamics (examined in the previous section). Based on the neural network representation of spring dynamics (see equation 4.17) and the input-output choice, a training model is presented in the next section.

4.2.2.2 The Training Model

Given the input-output data dr (equation 4.18), dv (equation 4.19) and $\mathbf{f}_s$ (equation 4.24), the nonlinear functions $K(\cdot)$ and $D(\cdot)$ are approximated using neural networks $N_K(\cdot)$ and $N_D(\cdot)$, respectively. The neural network model output shown in figure 4.5 is compared to the actual mass-spring system output shown in figure 4.4. The neural networks $N_K(\cdot)$ and $N_D(\cdot)$ are trained using the error between the neural network representation and the mass-spring systems to minimize the cost function:

$$J = \sum_{1}^{M}[\mathbf{f}_s(k) - \hat{\mathbf{f}}_s(k)]^2$$

$$= \sum_{1}^{M} e(k) \tag{4.25}$$

where M represents the number of training samples.

The neural network model integrated into the spring dynamics shown in Figure 4.6 is used to train the two neural networks $N_K(\cdot)$ and $N_D(\cdot)$. The neural network

responsible for learning the spring stiffness takes the length changes of the springs as input and its output is multiplied by the appropriate vectors associated with the springs. The second neural network, approximates spring damping, receives velocity changes as inputs and its output is also multiplied by a measurable vector. These multipications vectorize the output of both neural networks. It is important to note that both inputs to the neural networks are scalars and consequently the neural networks also produce scalar outputs representing the spring coefficients. The outputs of neural networks are vectorized to obtain the total spring forces in the x, y and z directions.

The sum of both outputs is then compared to the desired output resulting in an error signal. This error signal is used to train both neural networks. To ensure better learning and convergence, we propose a new learning rate called "adaptive learning rate" that is examined next.

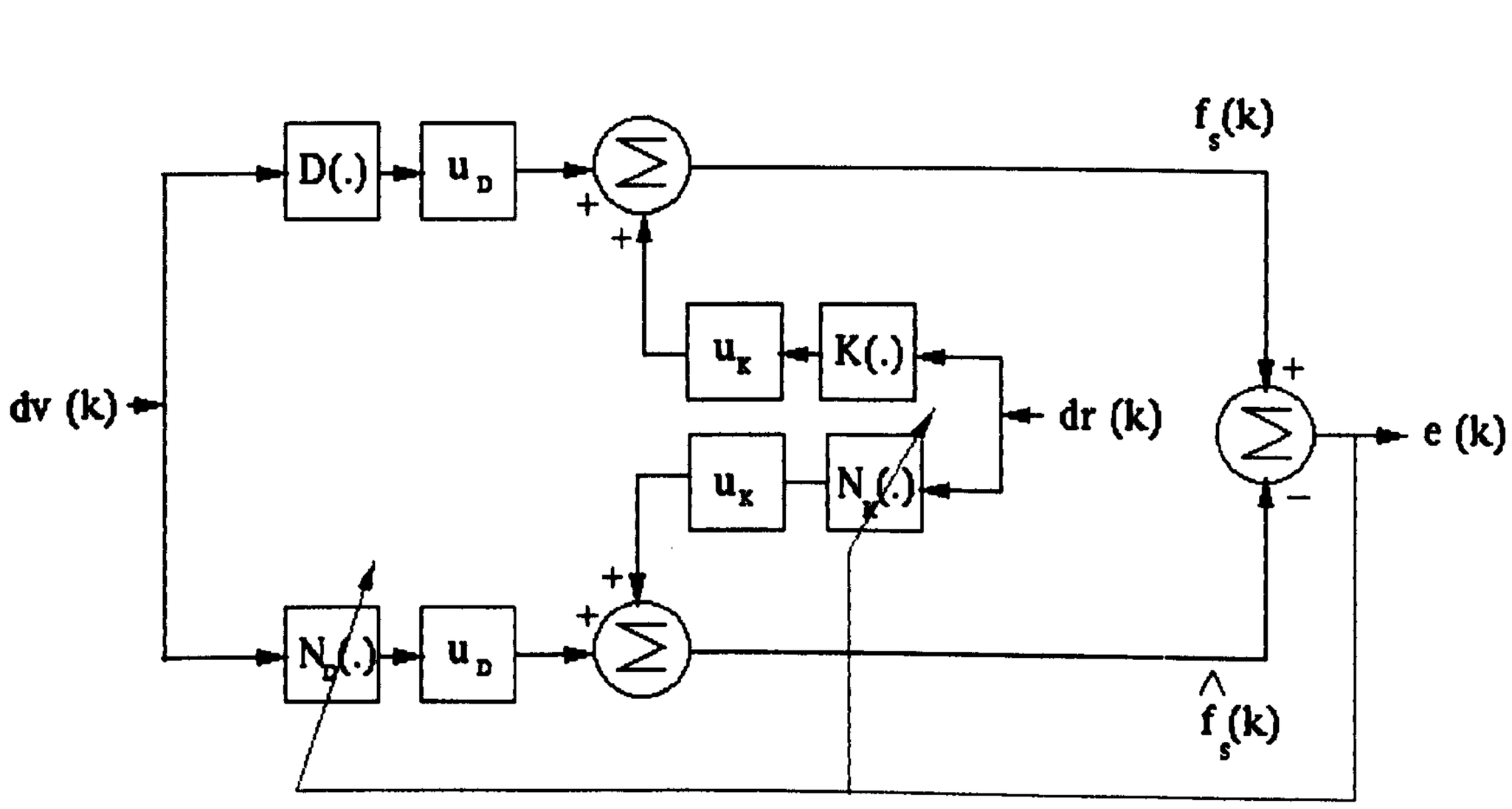


Figure 4.6 Training neural networks $N_K(\cdot)$ and $N_D(\cdot)$.

4.2.3 Learning Rate

The conventional learning algorithms are derived based only on the force errors at the mass-points. Although this appears correct, it does not produce satisfactory results. This is because it only considers the aggregate effect of all the springs incident at the mass-point rather than the contributions from these individual springs. It is understandable that the same displacement of a mass-point will result in different length change and velocity change for each spring depending on the structure of the mass-spring system. Therefore each of these springs will contribute a different amount to the movement of the mass-point. In the existing literature, however, the weights of neural networks are updated without acknowledging this important fact. This implies that the neural network is forced to produce the same output for different inputs. Consequently the errors may be minimised but the exact physical parameters may not be identified, resulting in unrealistic simulations.

It is well known that the spring force is a function of the changes of the spring length and velocity. In linear cases it is given by equations 2.8 and 2.9. The error given by equation 4.25 is also a function of the spring length and velocity changes. In the training process this error should be distributed to each state of the neural network. As there is no access to individual spring forces or errors, to get around this problem we introduce an adaptive learning rate. For the spring stiffness this rate is related to spring length changes and it is given by the following formula:

$$\beta_K(dr) = \beta_O(1 - e^{-(|dr| / \Sigma|dr|)}) \tag{4.26}$$

where β_O is a constant learning rate and $\sum|dr|$ is the total length change of the springs incident at each mass-point. The additional term in the formula effectively accounts for the contribution of the length change of each individual spring.

A similar adaptive learning rate can be chosen for the damping coefficient based on the velocity changes of the springs:

$$\beta_D(dv) = \beta_O(1 - e^{-(dv/\sum dv)}) \tag{4.27}$$

where $\sum dv$ is the total spring velocity at each mass-point.

4.2.4 The Weight Update Law

The weight update rules of the CMAC neural network are given by equation 4.11 and 4.12. The weights of the CMAC neural network can be updated using either non-batch (where the weights of the networks are adjusted after each training pair of inputs and outputs is presented to the network) or batch methods (where the weights are updated after a block of data is presented to the network). Non-batch training is used here because, as it is shown in (Sayil and Lee 2002), the non-batch error correction presented by Albus is the fastest method. It is therefore very suitable for on-line training. This method has the advantage that any learning factor in the range $0 < \beta < 2$ will cause the algorithm to converge.

The weight update law for both spring stiffness and damping can be given by the following formulations adapted from equation 4.12:

$$w_K(i+1,k) = w_K(i,k) + \frac{\beta_K}{A_K^*}\theta_K^T[e(k)] \tag{4.28}$$

$$w_D(i+1,k) = w_D(i,k) + \frac{\beta_D}{A_D^*}\theta_D^T[e(k)] \tag{4.29}$$

where w_K and w_D represent the weight vectors for spring stiffness and damping, the learning rate β is defined by equation 4.26 and 4.27, $k = 1,2,...,m$ is the sample number and i represents the iteration. The error function in equation 4.28 and 4.29 can be given as:

$$e(k) = |\mathbf{f}_s| - |\mathbf{u}_K \theta_K w_K(k) + \mathbf{u}_D \theta_D w_D(k)| \tag{4.30}$$

We make the following assumption in the weight update law: the output of the neural networks is compared with initial set values and if this output is found to be less than initial value, then the weights are not updated. This criterion is used to prevent learning negative coefficients. This is especially the case where there is lack of training data, which produces negative values on the right hand side of equations 4.28 and 4.29. This type of unwanted weight update produces negative learning (i.e. it produce negative values for stiffness and damping). In order to avoid this, the problem learning algorithm prevents updating the weights in those situations. The pseudo-code description of the weight update law is given as follows:

Initialize the weights: C_K, C_D

Loop

- *Find NN model output,* $\mathbf{f}_s$

Find the error, $\mathbf{e} = \mathbf{f}_s - \hat{\mathbf{f}}_s$

Find the learning rates, β_K *and* β_D

Modify the weights using, Eq.4.28, 4.29

Find the outputs $N_K(\cdot)$ *and* $N_D(\cdot)$

If $(N_K(\cdot) < C_K)$

$$w_K(k+1) = w_K(k)$$

If $(N_D(\cdot) < C_D)$

$$w_D(k+1) = w_D(k)$$

If (error< δ*), loop terminates.*

End of loop

The weights of the neural networks can be initialized randomly or set to known values. It is logical to set the initial values of the weights by best guest. The learning factors can also be chosen as constant values (i.e. the traditional way) or by using equations 4.26 and 4.27. The number of quantization levels and the number of active weights are determined by the designer's experience and the requirement of the

application. In our application we used (A^*) 80 active weights and 1000 quantization levels. The learning factor β_0 for both networks was set to 0.15.

4.2.5 The Training Data

In this section, we examine how training data for our neural network based identification method is obtained. In our experiment, we assign the values for the stiffness and damping coefficient. Then using the neural network based identification system developed above, we try to identify these given values. The training data for a mass-spring system should be obtained by experimentation. In order to test the accuracy of the proposed method, we generated the training data mathematically. This is because it is difficult to obtain real data. This is in fact an open research subject. Our experiment was carried out using the CG head model, which was generated in chapter 3.

Here we assume that all springs used have the same coefficients (i.e., we assume the same non-linear function for the coefficients). As discussed in (Teschner et al. 2000), spring stiffness is a non-linear function of length change, and it is defined as:

$$K(dr) = k_O(1 + k_1 * dr * dr) \tag{4.31}$$

where k_O and k_1 are constants. In this example, we set k_O=1 and k_1=10. Similarly, the damping is also represented by a function:

$$D(dv) = d_o(1 + d_1 * dv) \tag{4.32}$$

and we set d_o to be 0.1 and constant d_1 to be 0.01. We initialized the weights of the neural networks randomly, and let $N_K(\cdot) = 0.2$ and $N_D(\cdot) = 0.02$. The simulation was run for number of iterations and the positional changes were recorded. The total spring forces effecting each mass point were found using equation (4.24). Using the

configuration depicted in Figure 4.6 two neural network models were trained. These results are examined in the next section.

4.3 Results

CMAC networks (see section 4.1), were utilized in the parameter identification method (see section 4.2) for learning spring stiffness and damping. The training data generated in Chapter 3 was used. The parameters used in the data generation are known values so that we can compare the learnt coefficients with their original values. We carried out the training process using the developed identification model with both the conventional and the proposed weight update law.

The error function defined by equation 4.25 was plotted against the number of iterations for both neural networks. Figure 4.7 shows the graph for a constant learning rate and figure 4.8 shows the graph for an adaptive learning rate. In the first case the learning rate was 0.15 and for the second case it was defined by equations 4.26 and 4.27. The neural network model with the adaptive learning rate clearly minimises the error function with a final value of 0.043, while the neural network with the constant learning rate minimises the error to some constant with an average final value of 0.6. The adaptive learning rate ensures better learning, therefore the neural network model is able to generate almost the exact forces as the original mass-spring systems. Another issue that we have considered for the identification process is whether the neural network converges to the original coefficients. In Figures 4.9 and 4.10, we have plotted the original values of the coefficients and the neural network outputs. The x axis of the graph represents changes in spring length and the y axis represents the corresponding value for spring stiffness. As can be clearly seen, the adaptive learning rate improves the neural networks approximation significantly.

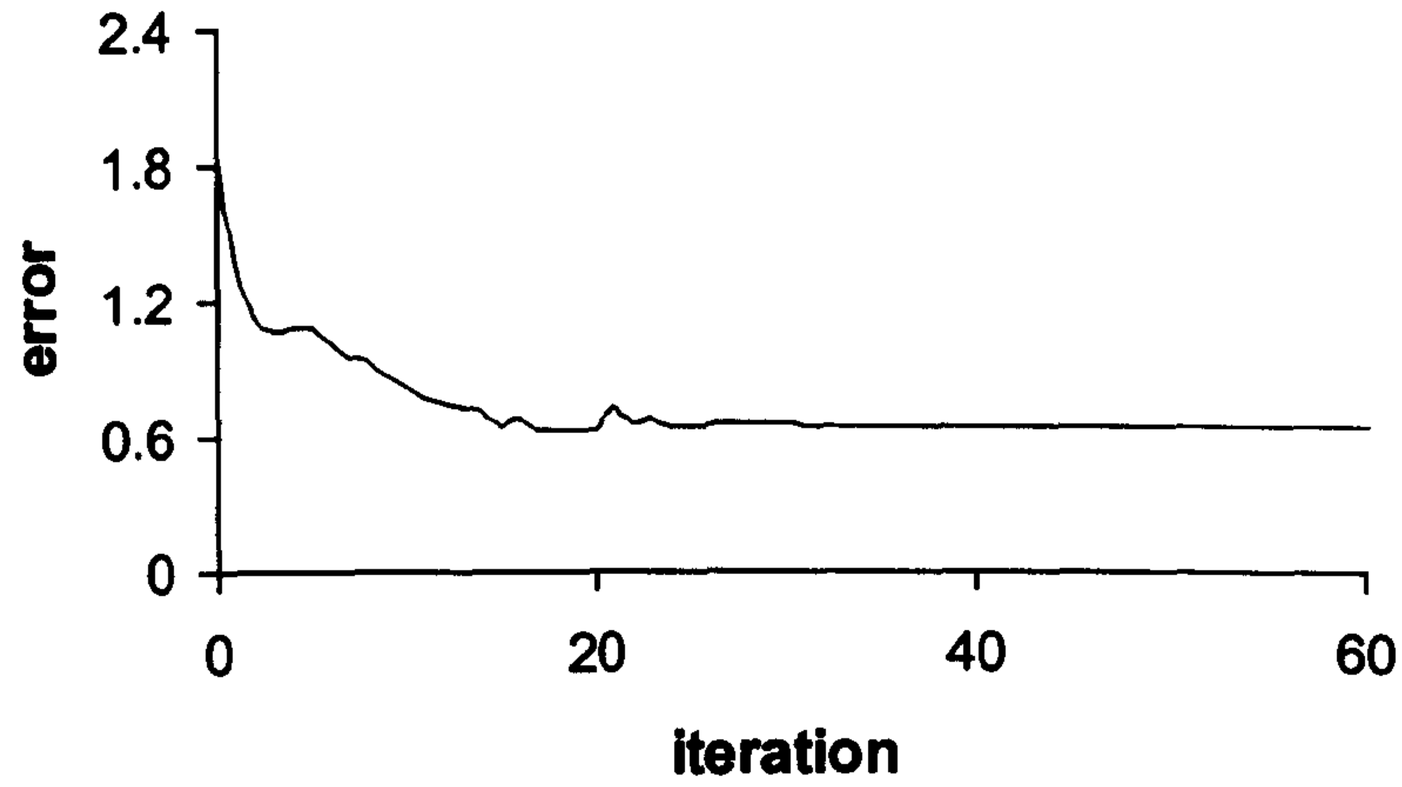


Figure 4.7 Error function versus number of iterations using a constant learning rate.

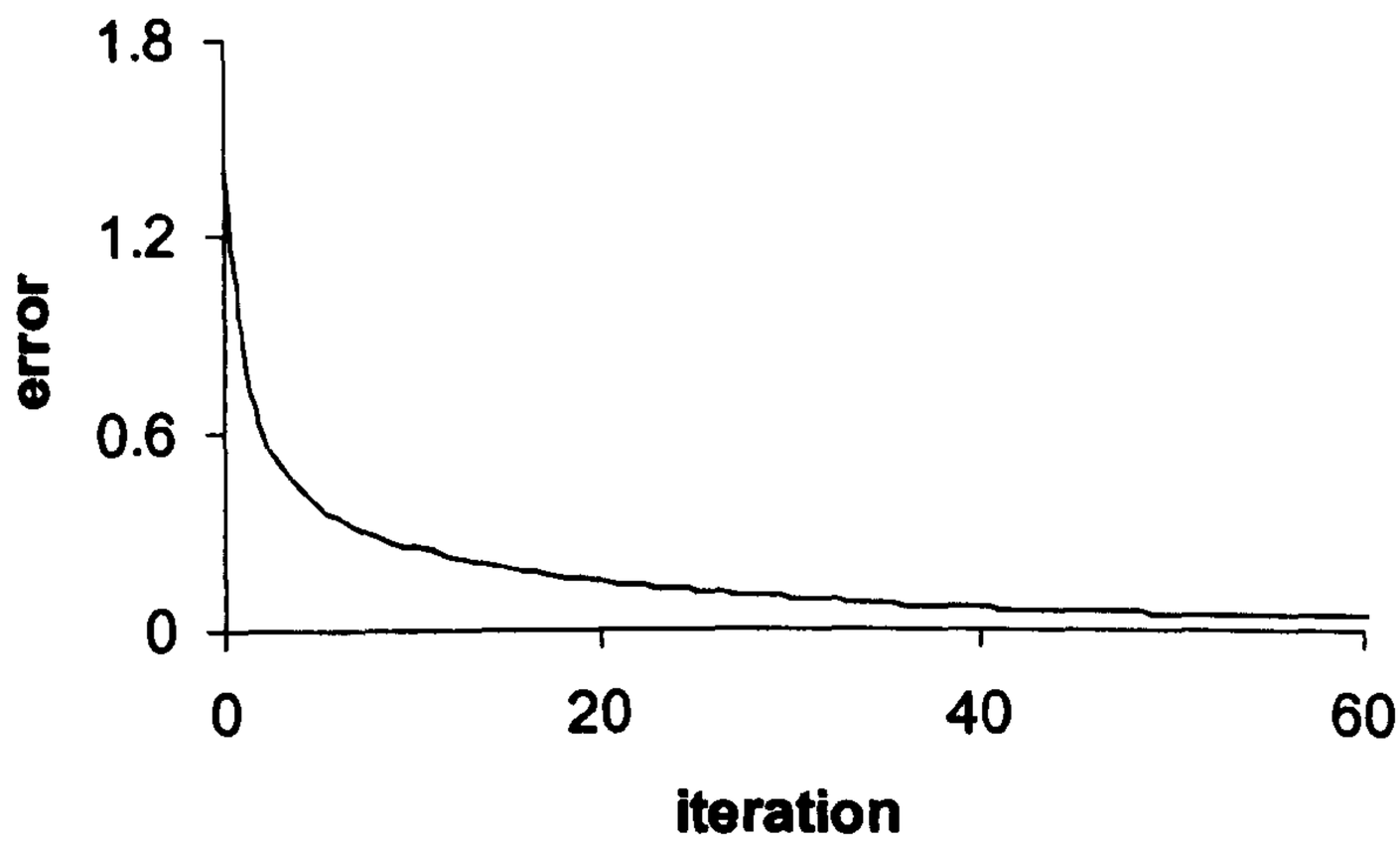


Figure 4.8 Error function versus number of iterations using an adaptive learning rate.

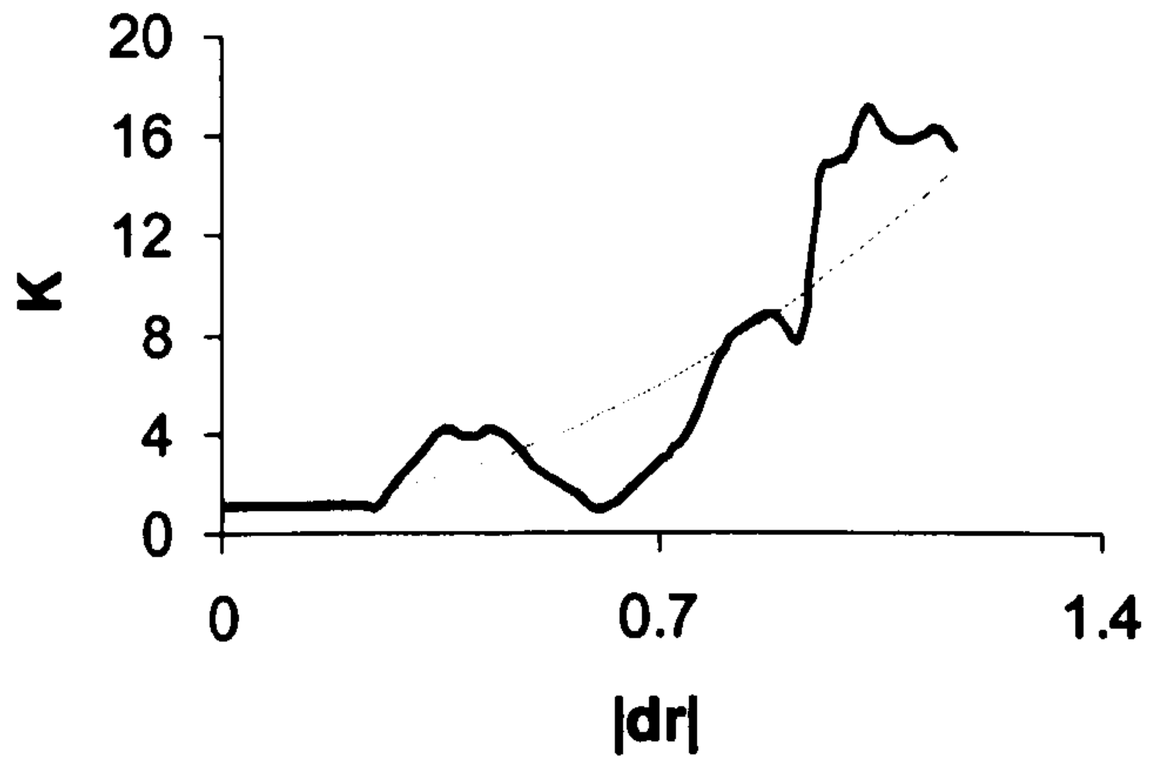


Figure 4.9 The original and the neural network approximation of parameter K, with a linear learning rate.

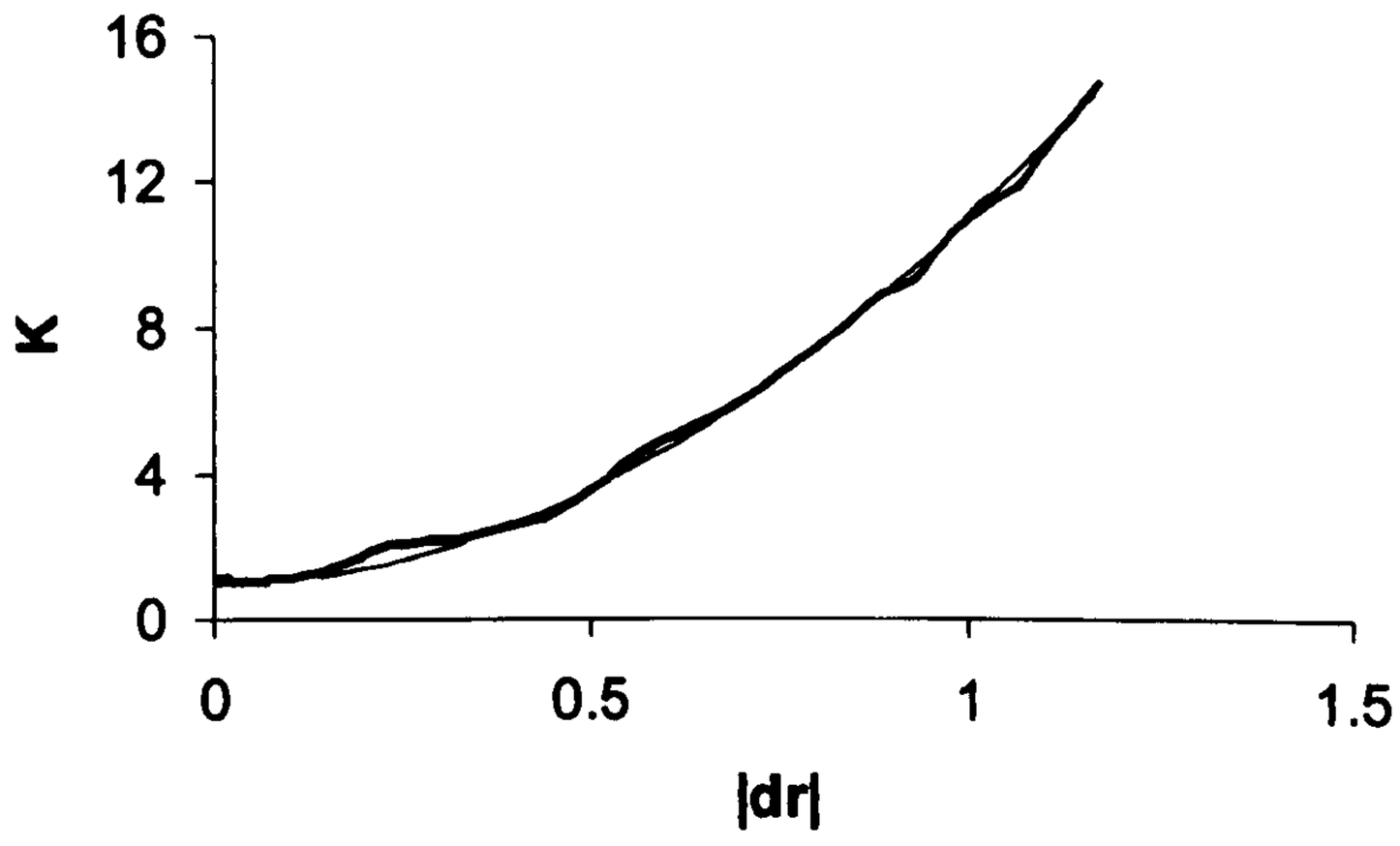


Figure 4.10 The original and the neural network approximation of K, with an adaptive learning rate.

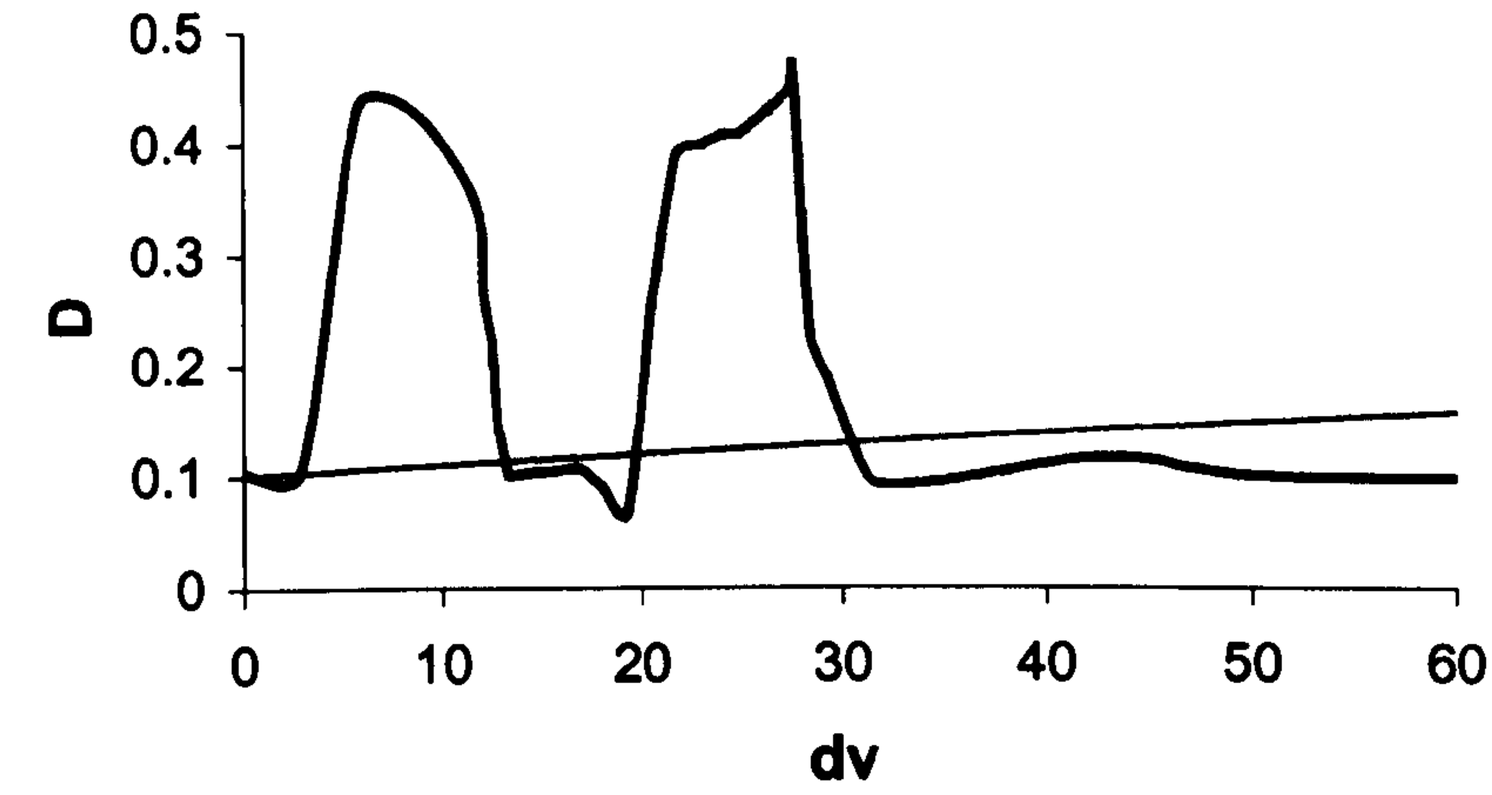


Figure 4.11 The original and the neural network approximation of parameter D with a linear learning rate.

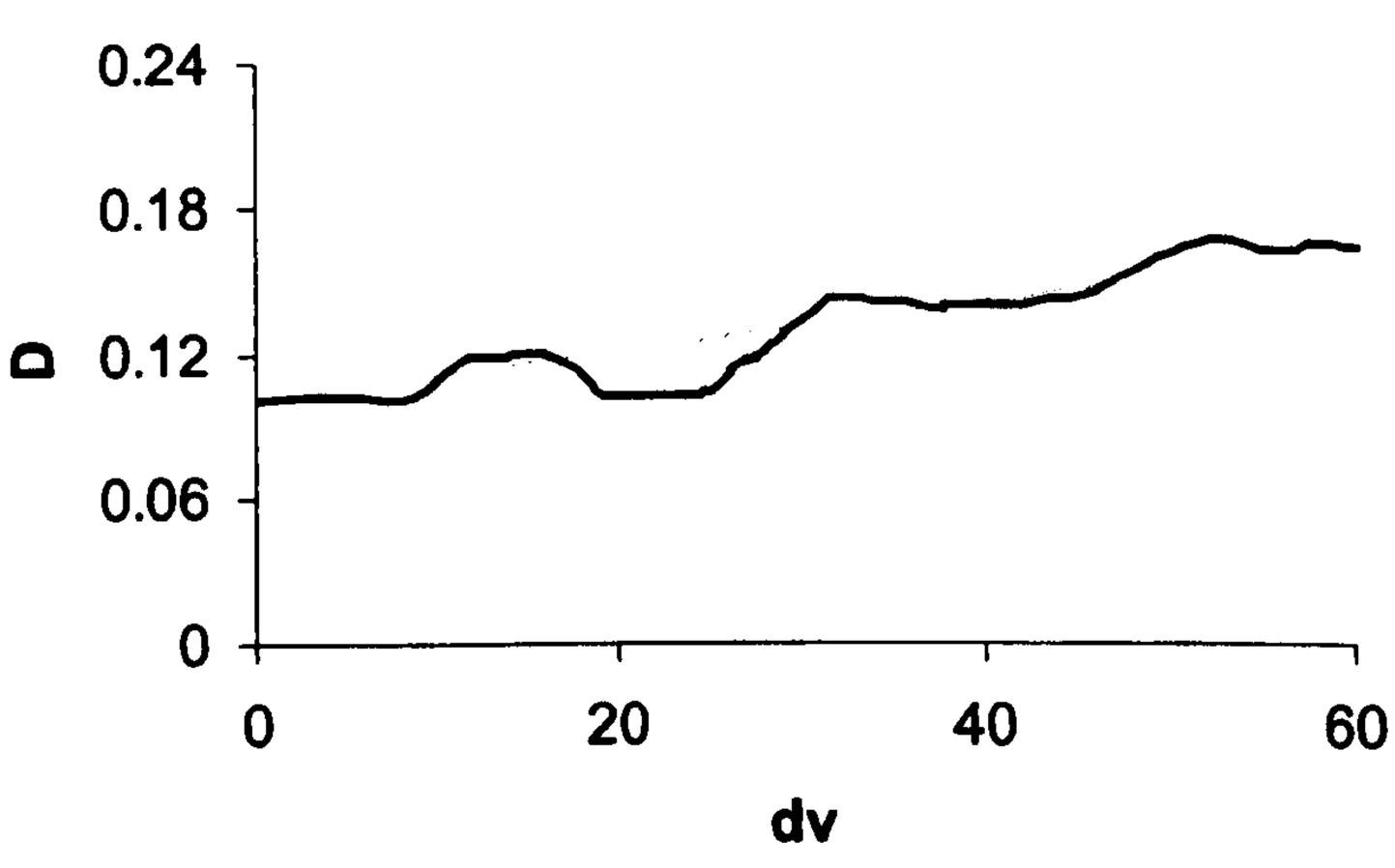


Figure 4.12 The original and the neural network approximation of parameter D with an adaptive learning rate.

Figures 4.11 and 4.12 show the original values and the neural network outputs for spring damping. The first figure is obtained using a constant learning rate, while the second figure illustrates the damping coefficient obtained using an adaptive learning rate. As seen from figure 4.11, the conventional learning algorithm fails to learn the damping successfully. Use of the adaptive rate in our learning algorithm, on the other hand, once again significantly increases the neural network learning.

In addition, to quantify the training error, the mean-squared error (see equation 4.32) of all the springs are shown in Table 4.7. This table indicates how closely these parameters are learnt. The mean-squared error is calculated by the following formulas:

$$MSE_K = \frac{1}{m}\sum_{1}^{m}(K(dr) - N_K(\cdot))^2$$

$$MSE_D = \frac{1}{m}\sum_{1}^{m}(D(dv) - N_D(\cdot))^2 \qquad (4.32)$$

where m is the number of training samples.

	Constant β	Adaptive β
MSE_K	0.9749	0.0092
MSE_D	0.0157	0.00004

Table 4.7 The mean -Squared-Errors between the neural network outputs and the original coefficients.

4.4 Summary

The previous chapter revealed that the choice of simulation parameters is the most important part of the mass-spring simulation. In this chapter we investigated ways of determining these parameters. System identification using neural networks is a well-

established method in engineering applications but little effort has been expended in computer graphics. In (Nurnbergel et al. 1998, 1999, 2001), six neural networks were used in identification process. This caused some problems since the only accessible information is the positions of the mass-points and the "teacher" signals for all neurons can not be determined. Error accumulation also presents a problem because the output of some of the neurons are used to feed other neurons. If the output neuron in previous model is trained, portability is not guaranteed. In addition, they admitted that their identification process did not always convergence to the original parameters. We, therefore, have developed a specific identification technique for the mass-spring algorithm (Duysak and Zhang 2003). We only used two neural networks to identify spring stiffness and damping coefficients. Changes in the spring length and velocity are used as inputs to neural networks in order to increase the generalization of the networks as well as to overcome the portability issue. The proposed adaptive learning rate improves the convergence of the developed technique.

Experiments show that the developed model is able to identify system parameters with very high accuracy. The developed model was tested using data generated by mass-spring system simulations. We anticipate that this model will find an effective use in the simulation of real life systems, where the deformation data is obtainable using experimental means (Brouwer et al. 2001, Kerdok et al. 2001). It is also possible to use this method with the implicit integration scheme (Kang et al. 2000) and the quasi-static formula (Brown et al. 2002) to improve the simulation efficiency. Current identification systems use data generated mathematically from mass-spring systems or from finite element modeling and testing our model with real data is part of future work.

CHAPTER 5

A NEW DEFORMATION ALGORITHM; MASS-SPRING-CHAIN (MSC)

Depending on the requirements of the applications, researchers may use different simulation techniques for soft tissue simulation. The finite element method is a common choice if accuracy is the main concern while mass-spring systems may be preferred if speed is essential. But even with the mass-spring system, real-time performance, which is essential in many surgical simulation applications, is difficult to achieve, since most real applications involve a large number of soft tissue (e.g. muscles) elements. In addition the mass-spring system is an iterative method, which uses numerical iteration to perform deformation. Surgical simulations that requires 30Hz update rates and further more real-time simulations with haptic interactions, which require 1kHz update rates are still a challenge to realize. Therefore, whilst much research effort has been spent on improving such techniques in the area of physical accuracy and performance, other methods, such as the ChainMail algorithm (which is not physically-based), have been proposed for real-time interactive frame rates.

Unlike the finite element model or mass-spring system techniques that use reduced resolution but complex formulations to calculate deformation, the ChainMail algorithm operates on full or very high resolution and uses very simple calculations. In addition, the ChainMail algorithm is not an iterative method that takes many steps to reach the final state. It only needs one or two simple steps, hence it is very fast. The ChainMail algorithm models the object by a large number of vertices, which are interconnected with links. A cubic cell is formed around each vertex defined by its six immediate neighboring vertices in three mutually perpendicular directions. When a point is moved, its neighboring vertices may be moved according to the defined rules. The affected vertices may in turn move their neighbors and so on. The movement is thus spread to the whole model, leading to the deformation of the object. According to this algorithm's deformation law, vertices are allowed to move within set limits representing the minimum and maximum distances from each other.

In this chapter we propose a new method for deformation simulation in order not just to achieve real-time performance but also to reach the required haptic rate. The method is non-iterative and its structure is suitable for neural network design. Before going into the detail for combining physically and non-physically based methods, our motivation is described next.

5.1 Motivation

We observe that the ChainMail algorithm determines the deformed positions using defined geometric constraints with one or two simple steps, while mass-spring systems use an iterative approach (with hundreds of steps). Therefore, the performance of the ChainMail algorithm is considerably faster than that of mass-spring systems. Mass-spring systems on the other hand use a more sophisticated

deformation algorithm and therefore produce more accurate results. Finding a deformation technique with one or two simple and fast steps while improved accuracy at the same time motivated us to study closely both the mass-spring systems and ChainMail algorithms. A new method is developed to combine the desirable properties from both these algorithms while eliminating or at least ameliorating their drawbacks. Here we list some of the undesirable properties of both the mass-spring systems and the ChainMail algorithms and finally at the end of this chapter we provide the properties of the new algorithm.

The ChainMail algorithm possesses some disadvantages including:

- It does not work on triangulated models. Since most of the applications use triangulated data or models, the ChainMail algorithm is not suitable for these applications.
- It assumes that each neighbor of any point lies in the vertical, horizontal, top, or bottom directions. Thus it works only on voxel-type structures.
- It uses a very simple deformation formula that is difficult to justify physically.
- It is difficult to determine geometric constraints representing the physical characteristics of material.
- It does not take rotation into account.
- Some special cases examined in following sections are not handled by this algorithm.

The mass-spring systems algorithm on the other hand, also has some disadvantages:

- It uses an iterative approach, therefore it is not suitable for real-time applications in many cases.

- Some modifications are employed (e.g. the approximated implicit method in (Kang et al. 2000a) or the quasi-static method (Brown et al. 2001)) to achieve real-time performance at the expense of simulation accuracy.
- Although the mass-spring system's algorithm uses a well defined deformation algorithm, it is very difficult to obtain its simulation parameters (Chapter 4).
- It does not handle rotation and special cases are not addressed.
- It causes spring elongation, which can be avoided using additional computations but causes accuracy and speed problems.

In the following sections we will describe our algorithm in detail. First a pattern is developed to process deformations on a 3D mesh. The boundary limits are established for the spring movements and deformations. A surface where the deformation occurs is then identified. A method is proposed to find the exact locations of the deformed points on this surface. We also address multiple vertex movements and cell conversion that is likely to occur in such applications. Finally, the algorithm is summarized and applied to simulate various objects, including simple geometric figures and complex soft tissue deformations due to craniofacial surgery.

5.2 The Mass-Spring-Chain (MSC) Algorithm

The Mass-Spring-Chain algorithm models the object in a similar way to the mass-spring systems algorithm in that the object consists of a number of mass-points connected with springs. As in the mass-spring system's algorithm, springs perform a deformation by stretching or compression. The deformation starts from the moved mass-points and propagates through the entire 3D lattice of springs. The movements and deformations of the springs are constrained by pre-set conditions. Spring movement is limited between two extremes; rigid movement and elastic movement.

The spring length is also constrained between the allowed maximum compression and stretching. The deformation algorithm is then responsible for finding the necessary movements and the amounts of deformation of the springs within these set limits. We begin by introducing some definitions that will aid us in the description of the overall deformation process of this algorithm

5.2.1 Definitions

The 3D lattice is initially considered to be in a passive state, which implies that all points and springs in the mesh are not under the influence of any external force. Figure 5.1 illustrates such a lattice. When a point is subject to an external force (by grabbing it and moving it) this particular point becomes an active point or, in other words, a source point for the deformation. An active point is shown as black in figure 5.1. A deformation starts from an active point and travels through the rest of the lattice in every direction using the springs. The springs connected to the active point are now defined as active springs, because they are subject to movement and deformation. The other end points of the active springs are called semi-active points, because they will be repositioned (causing the deformation), and will become active points themselves in the next step of the algorithm. Semi-active points are shown in green in figure 5.1. Springs connecting semi-active points are called semi-active springs.

Some of the points in the lattice can be utilized as boundary points in order to enforce some constrains and to model connections with other objects. The springs connected to boundary points are called boundary springs. During the deformation process all the points and springs in the 3D mesh are labeled using one of the above definitions. They are all subject to deformation.

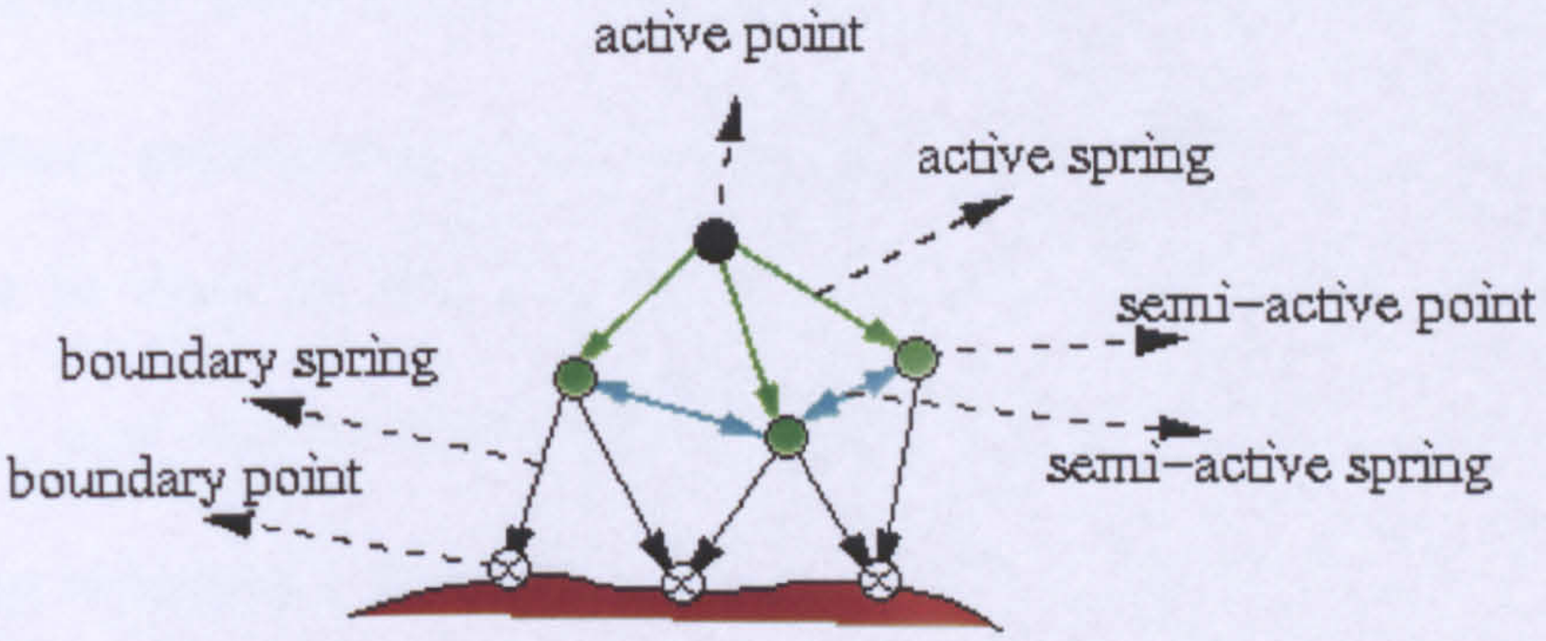


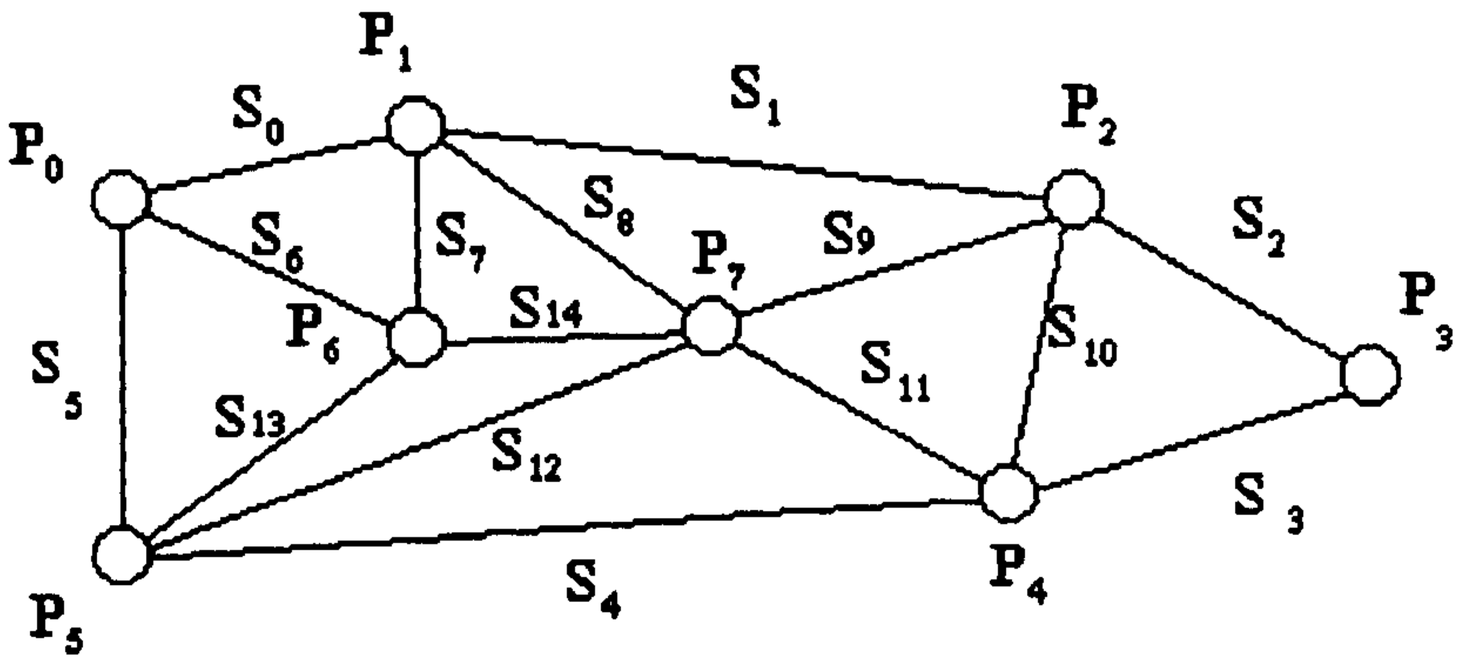
Figure 5.1 Definition of points and springs.

5.2.2 Deformation Pattern: Movement Propagation

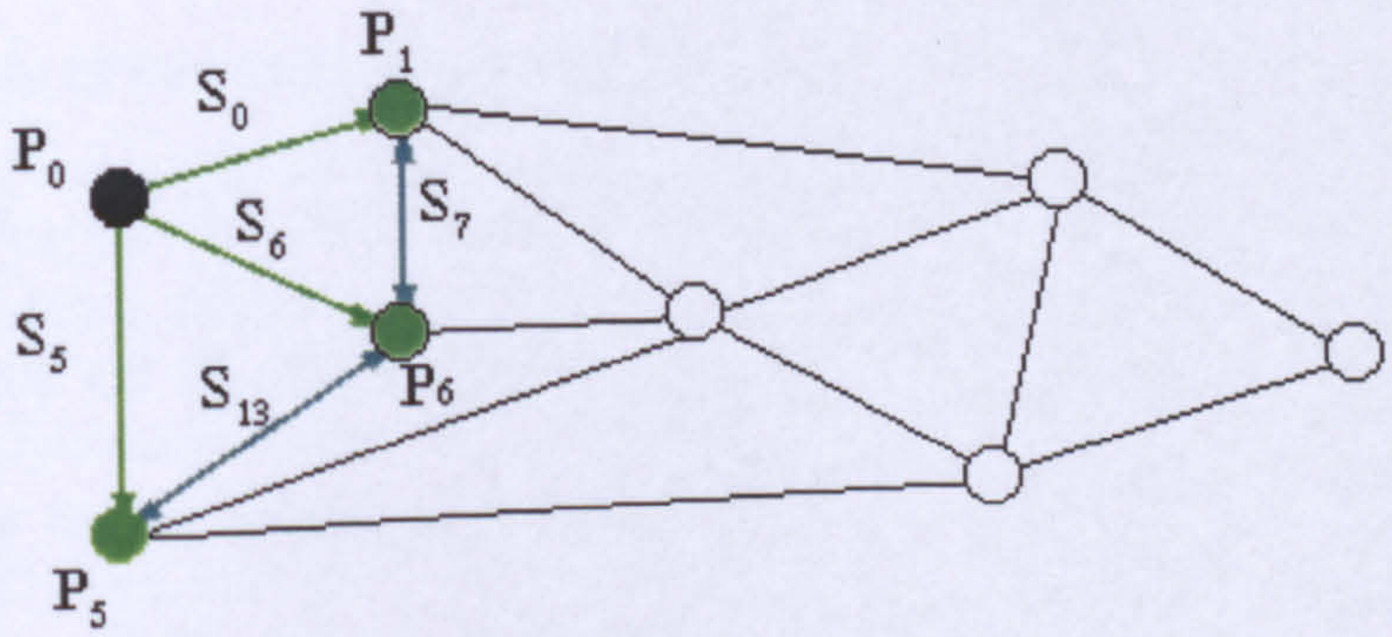
The deformation pattern can be compared to a wave on water where the wave is started by an external force (e.g. the fall of a pebble) and travels trough the entire surface of the mesh or until it diminishes. During the first cycle of the wave, active springs, semi-active points and semi-active springs are determined and the deformation takes place as explained in following sections. Before moving onto next cycle of the wave, the springs processed in the current cycle are turned off to allow only forward movement. The next cycle of the wave starts from the semi-active points of the previous cycle. Thus, these are now the source points for the movement and the deformation. Similarly, active springs, semi-active points and semi-active springs are found and deformed before they are turned off.

These cycles of waves go on until all the springs are visited and deformed. Alternatively, cut-off criteria can be set to stop the propagation and deformation as discussed in later sections. We now explain the deformation propagation through an example.

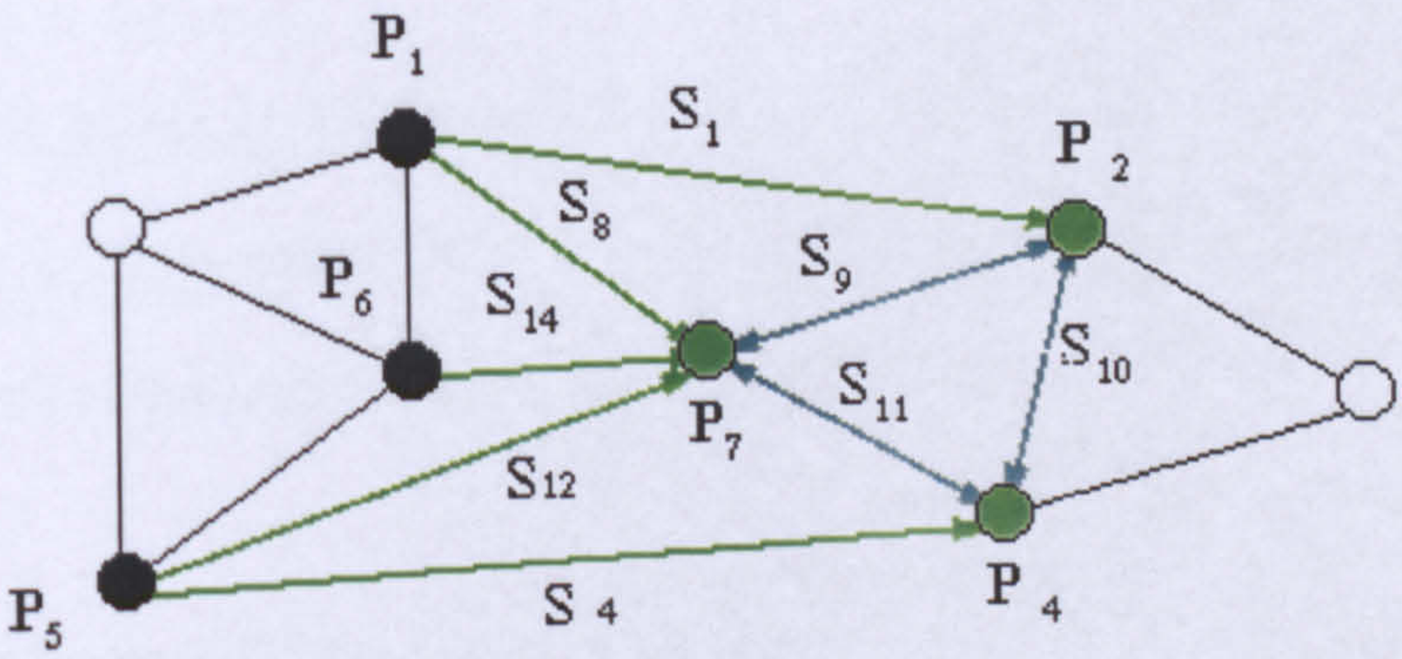
The deformation pattern of the algorithm is illustrated in figure 5.2. Figure 5.2(a) shows the initial mesh before the deformation starts. The point P_0 is moved by an external force and becomes an active point (moving point or source point). This point is shown in black in figure 5.2(b). The springs connected to the active point, s_0, s_5, s_6, now become active springs and are shown in green. These springs are subject to deformation because of the movement of one of their end points. The direction of the deformation is indicated by the arrowhead on these springs. The other end points of the active springs, P_1, P_5, P_6, are semi-active points, shown in green. The springs connecting the semi-active points, s_7, s_{13}, are semi-active springs and are indicated by double headed arrows. This implies that both ends of the springs are moving. Thus deformation takes place at both ends of the spring. After the active springs are processed (deformed), the semi-active springs have to be deformed as well. Therefore, the positions of each semi-active point are updated based on the deformation of the active and semi-active springs. The first wave is completed as shown in figure 5.2 (b).



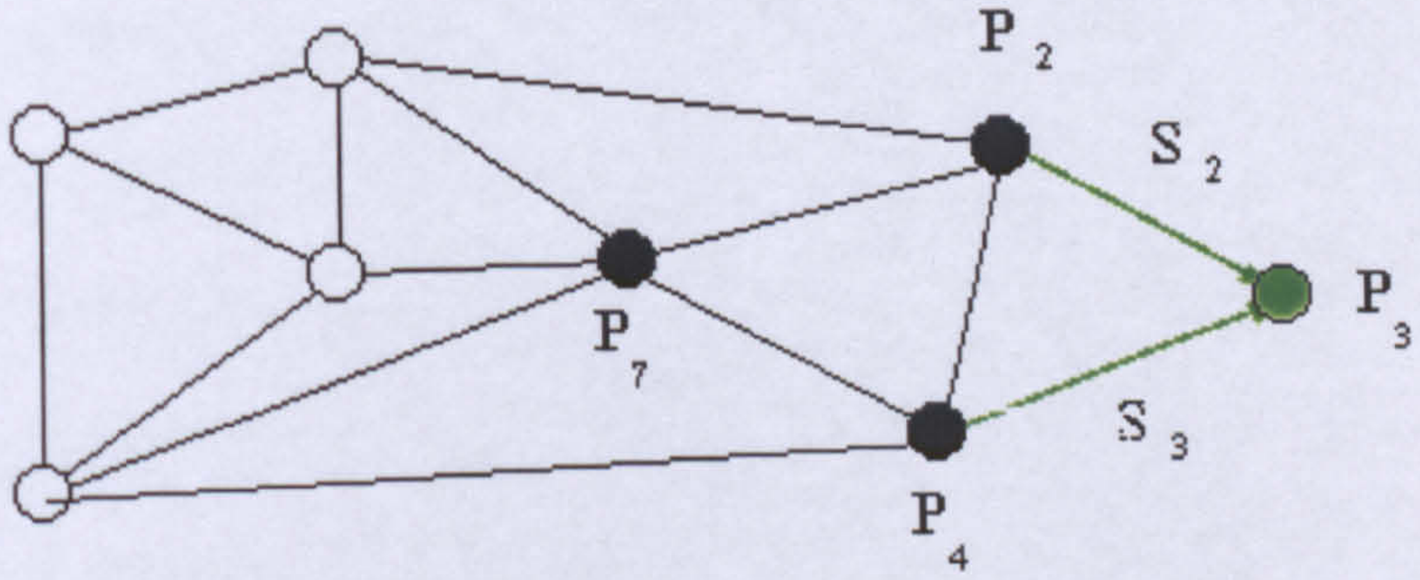
(a)



(b)



(c)



(d)

Figure 5.2. The deformation pattern (deformation propagation) of the proposed algorithm: (a) the initial mesh, (b) the first wave, (c) the second wave and (d) the last wave (the end of the propagation).

Before the first wave of the propagation ends the deformed springs $s_0, s_5, s_6, s_7, s_{13}$ are turned off. The semi-active points P_1, P_5, P_6 of the previous step now become the active points of this step, shown in black in figure 5.2 (c). The new active springs are $s_1, s_4, s_8, s_{12}, s_{14}$, the new semi-active springs are s_9, s_{10}, s_{11} and the new semi-active points are P_2, P_4, P_7. The second wave of the deformation is concluded by performing the deformation on these active and semi-active springs and finally turning them off. The last cycle of the wave is shown in figure 5.2(d) where there are only two active springs left, s_2, s_3, and P_3 is the only semi-active point.

Since all the points and springs have been processed the deformation propagation is terminated at this point. In the next section we analyze the movements and deformations of the individual springs.

5.2.3 Boundaries of Movements and Deformations

This section deals with establishing limits on the spring movements and deformations. We have taken into account the spring theory examined in chapter 2 and the ChainMail algorithm to determine the deformation constraints concerning the spring movement and deformations. Both theories suggest that neighboring points (connected by links in the ChainMail algorithm and connected by springs in mass-spring systems algorithm) have to satisfy some conditions relative to each other. In the mass-spring system's case, these conditions are embedded into the system by some parameters such as spring stiffness and damping. If the spring stiffness is high, for example, it will be harder to move and deform the springs. The ChainMail algorithm defines the conditions geometrically assigning a bonding box determining the limits of the point movements. The movements of each point are restricted to be in

this bounding box. There is not, however, a parameter given by the algorithm that controls the location of points in this bonding box.

The ChainMail algorithm only performs a deformation if a set of limits is violated. The mass-spring system's algorithm on the other hand, deforms the springs at any condition dynamically given by external forces. For example, let us assume that two mass-points are connected by a spring/link. If one of the points is moved, the spring is deformed according the theory examined in chapter 2. In the ChainMail algorithm, hovewer, the spring will only be deformed if the new spring length violates the minimum and maximum spring length criteria. A desirable method should deform the spring dynamically based on the changes of the conditions of the external forces or of the vertex movements. Next we give details on how to set these limits in the new algorithm.

5.2.3.1 Movement Limits

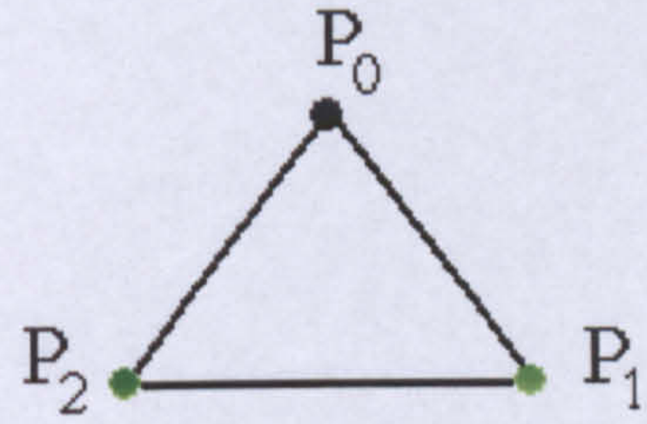
We assume that there are two extreme cases possible regarding spring movements. One is defined as a rigid movement without any rotation. When a point is moved, a connected spring moves accordingly. The initial spring and the moved spring are now parallel to each other and the distance between them is equal to the distance traveled by the moving point. A vector called the rigid movement vector, $\mathbf{u}_r$, defines this new position and sets the rigid limit or, in other words, the upper limit beyond which there will not be any movement. The opposite situation is known as super elastic movement and deformation. It is assumed that the spring offers no resistance to its movement, i.e. while one end is moving the other end stays still. A vector, represented by $\mathbf{u}_e$, from the moving end to the stationary end of the spring sets this limit. It is therefore our assumption that the moved and deformed spring will be somewhere

between the rigid and elastic limit vectors. The formation of the limit vectors is shown in figure 5.3 where (a) gives the initial triangle. The point P_0 is moved to a new location P_0^n. Figure 5.3 (b) shows the limit vectors for the spring between points P_0 and P_1. As seen from this figure point P_1 can be moved from one extreme, P_1^e to another P_1^r. Figure 5.3 (c) represents movement limits for the spring between points P_0 and P_2. In general the limit vectors are defined as follows:

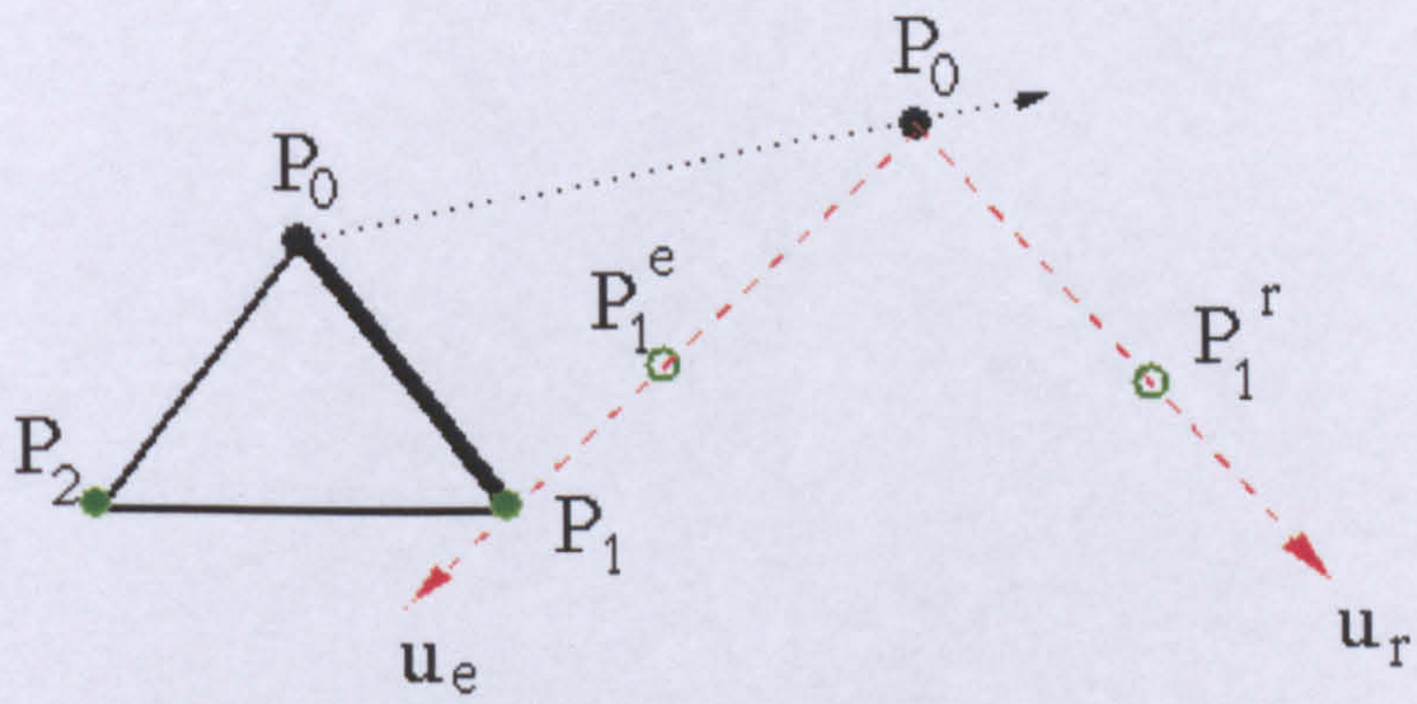
$$\mathbf{u}_r = p_m^n p_i^r$$

$$\mathbf{u}_e = p_m^n p_i^e \tag{5.1}$$

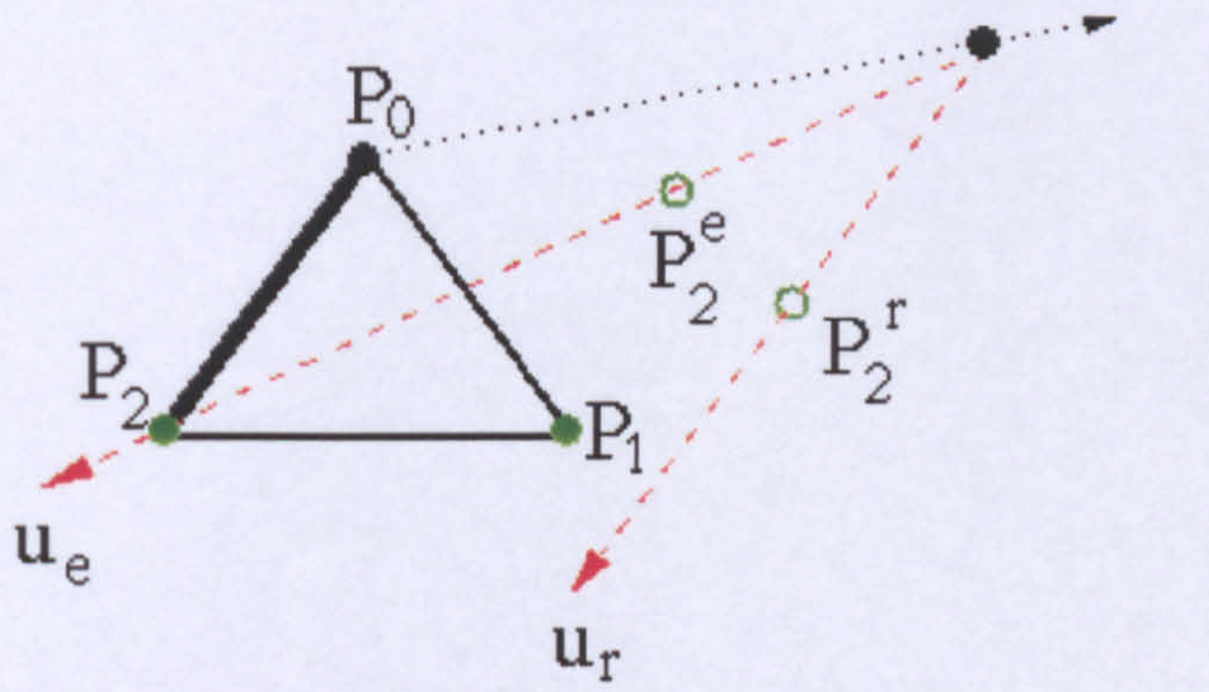
where m represents the moved point and i gives the connected semi-active points.



(a)



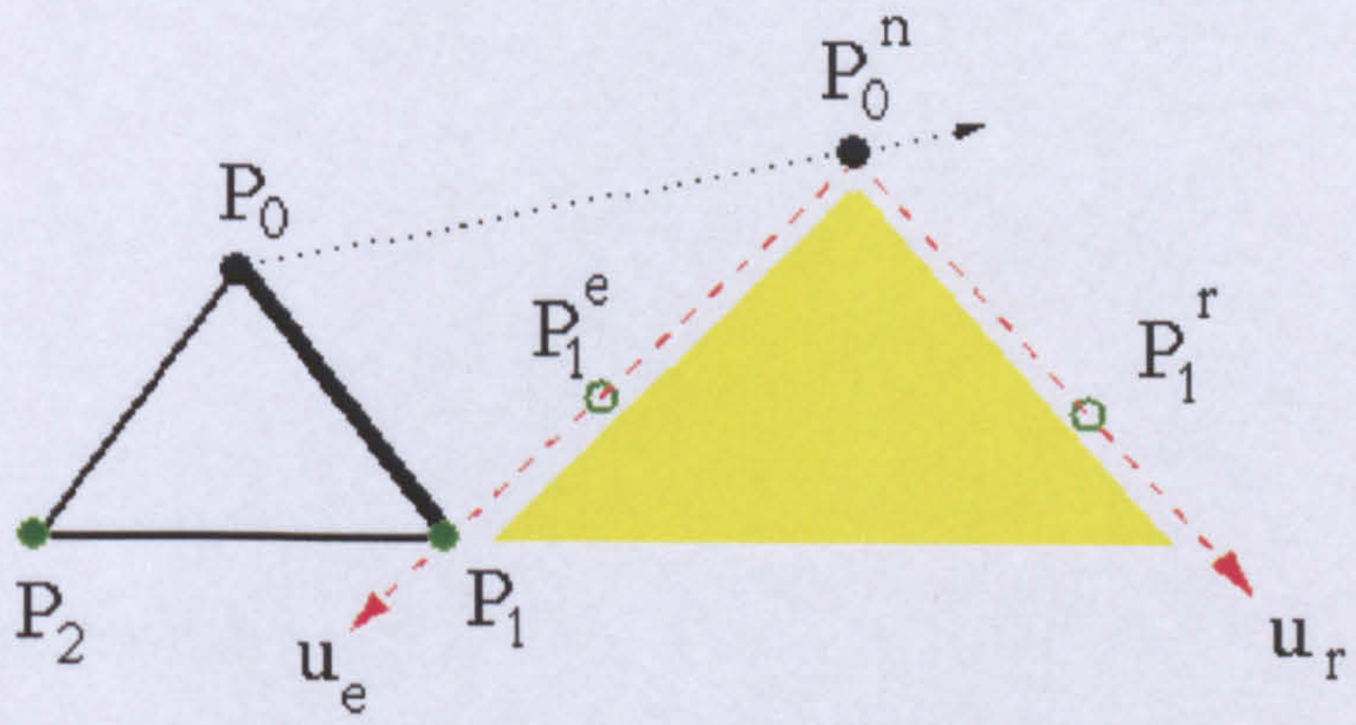
(b)



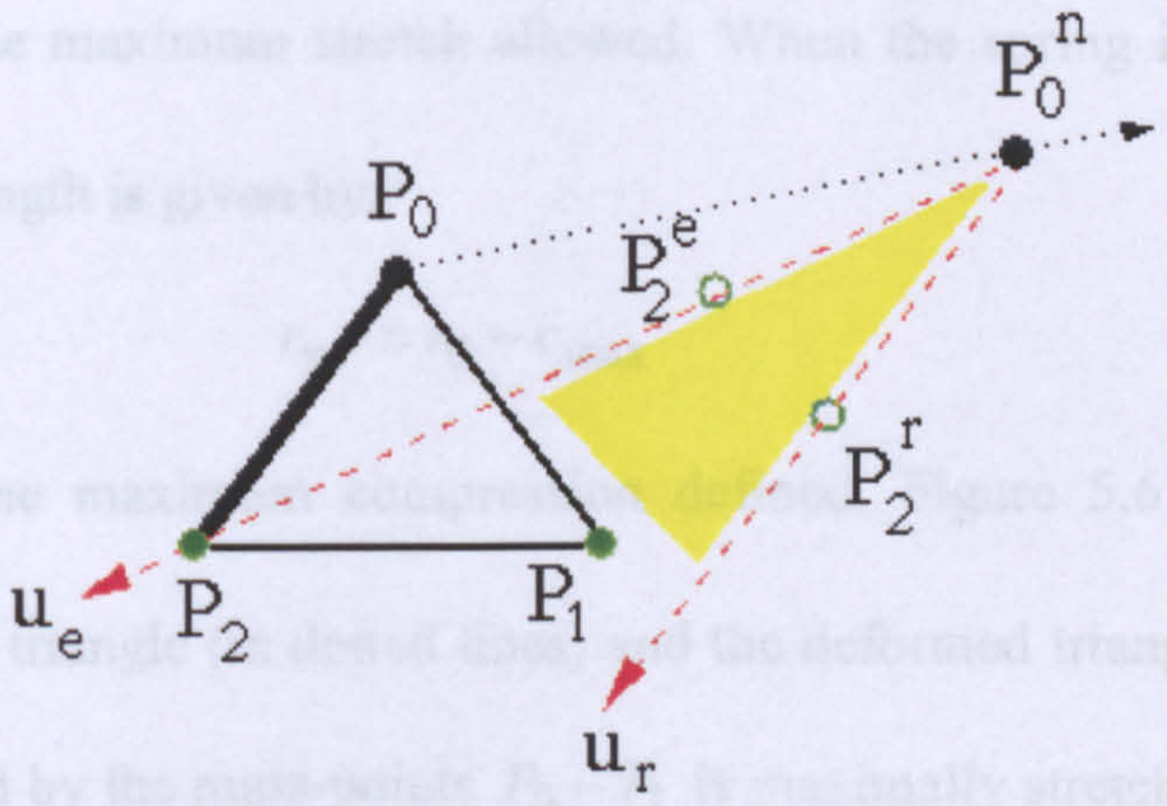
(c)

Figure 5.3 The determination of the movement limits due to vertex displacement.

While determining the movement limits, the springs are moved individually as if there were no connection to any other spring. As shown in figure 5.3 for example, the two sets of limits are formed independently from each other. It is assumed that the effects of connected springs fall into these limits. The springs are allowed to travel from the elastic limit to the rigid limit. Thus, the limit vectors form a surface called the movement surface on which locations of the springs are to be determined. Figures 5.4 (a) and (b) show movement surfaces formed by the elastic and the rigid limits (shown in figure 5.3 (b) and (c) respectively).



(a)



(b)

Figure 5.4 The movement surfaces formed by the limit vectors.

5.2.3.2 Deformation Length Limits

The previous section deals with establishing the movement limits of springs after point movement. In this section the deformation of the springs is examined. Springs are allowed to stretch or compress for a certain percentage of their original lengths r_0.

The current spring length r varies between the maximum compression length r_{mc} and the maximum stretch length r_{ms}. Figure 5.5 shows a sample spring at its rest, maximally stretched and maximally compressed states.

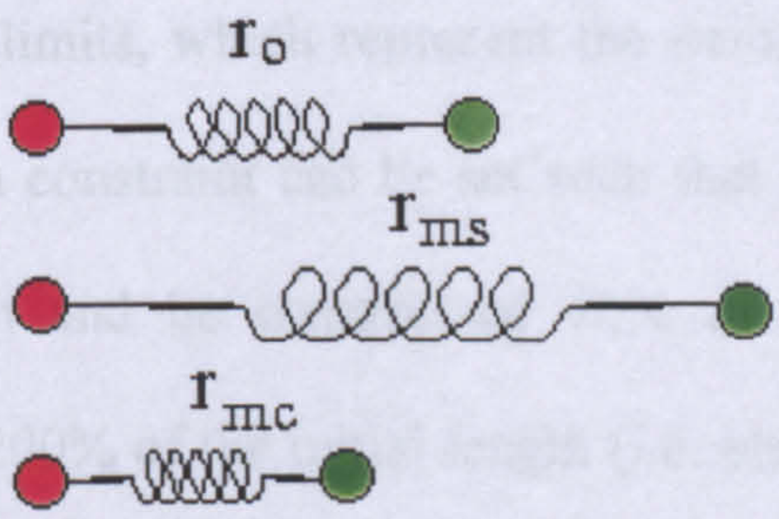


Figure 5.5 Spring length limits.

At maximum stretch, the spring length can be defined as:

$$r_{ms} = r_0 + d_{\max} \tag{5.2}$$

where $d_{\max}$ is the maximum stretch allowed. When the spring is under maximum compression its length is given by:

$$r_{mc} = r_0 - c_{\max} \tag{5.3}$$

where $c_{\max}$ is the maximum compression defined. Figure 5.6 shows the initial configuration of a triangle (in dotted lines) and the deformed triangle (in solid lines). The spring defined by the mass-points $P_0 - P_1$ is maximally stretched while the other spring between two points $P_0 - P_2$ is maximally compressed.

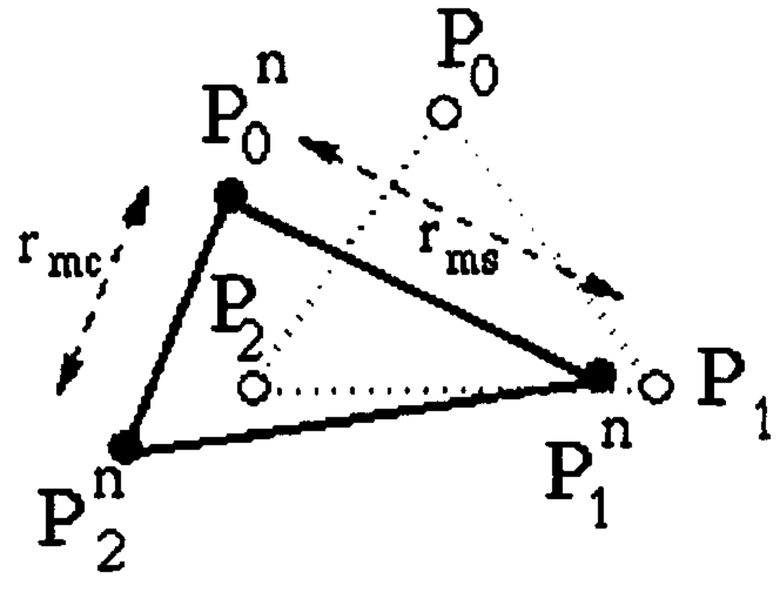


Figure 5.6 Spring length criteria.

In addition to the movement limits, varying the spring lengths sets further limits known as the deformation limits, which represent the deformation characteristics of the springs. For example, a constraint can be set such that a spring can be stretched 20% of its original length and be compressed 10% of initial length. For some materials this limit can be 200% of the initial length (i.e. plastic) based on the system requirements. During the deformation process the lengths of springs will vary between these limits. The limits eventually form a closed surface known as the deformation region. This region is also known as the search space of the deformation.

5.2.3.3 The Deformation Region

In section 5.2.3.1 we have defined the movement boundaries shown in figure 5.3 and in section 5.2.3.2 we have set deformation criteria shown in figure 5.6. The movement surface given in figure 5.4 is further narrowed down by the deformation limits resulting in a closed region, which is shown in figure 5.7 for movement surface depicted in figure 5.4 (a). The boundaries of this region therefore can be given by the elastic limit vector $\mathbf{u}_e$, the rigid limit vector $\mathbf{u}_r$, the minimum length constraint r_{mc} and the maximum length criteria r_{ms}. The deformation region (DR) can be formulated as follows:

$$\underline{\underline{\Omega}}_{DR} = \mathbf{u}_e r_{mc} \mathbf{u}_r r_{ms} . \tag{5.4}$$

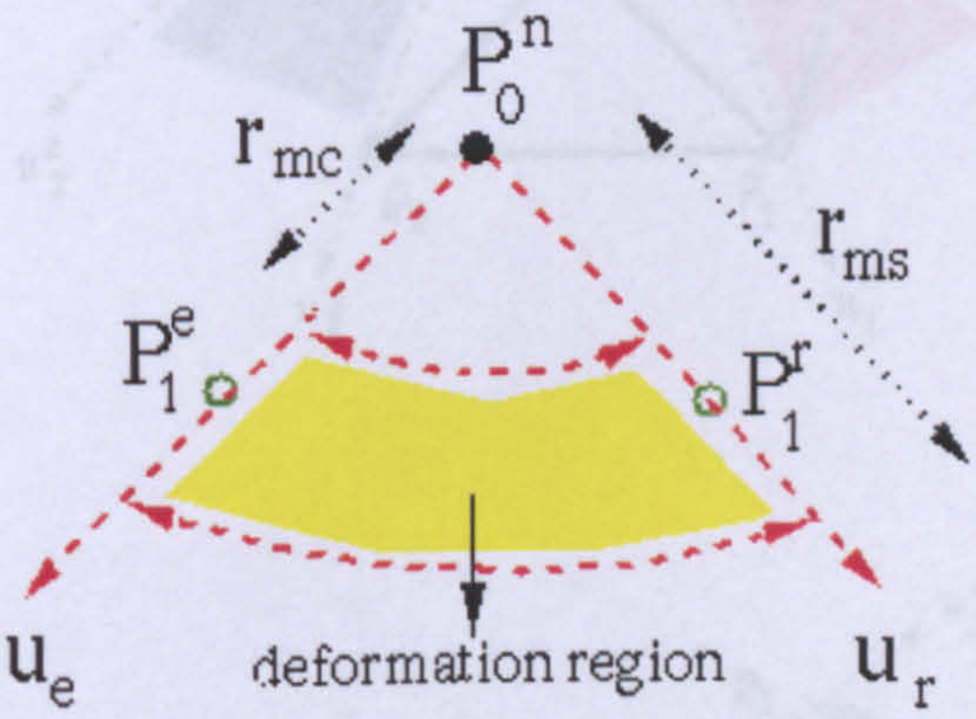


Figure 5.7 The deformation region is formed by the movement and deformation limits.

This region is called the deformation region since the deformed springs and points are contained in this region. Each spring connected to a moving point will form a similar region. In figure 5.8 we give an example of the deformation process, where the initial configuration is shown in thick black lines while red lines represent the rigid movement of the triangles without any deformation or rotation. Point P_0 is moved to

a new location P_0^n. The first part of the figure represents a pulling and the second part a pushing operation that starts the deformation. The deformation surfaces are shown in different colors for each active spring. Thin black lines represent possible deformed shape of the original figure. Our goal in the following sections is to obtain a method which will allow us to determine exactly where in this region the (semi active) points lie.

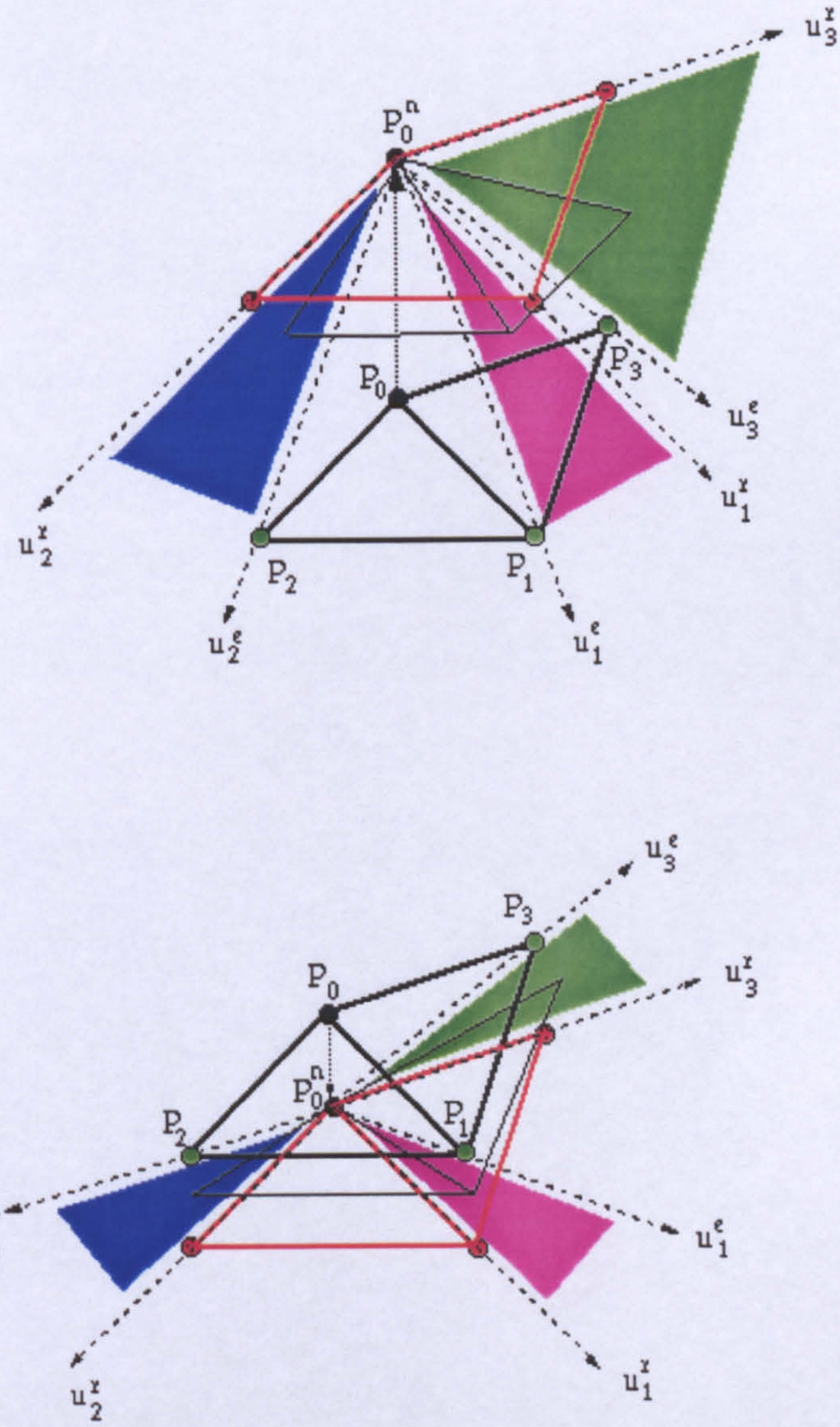


Figure 5.8 The initial position (thick black) and the rigid movement (red) are shown with the deformation surfaces in different colors for each spring.

5.2.4 Finding The Deformed Positions

Our strategy is first to find the new positions of springs due to the movement of active points. We then determine the amount of deformation and find the direction of this deformation. Finally, the new locations of the semi-active points are found. Let us now explain each of these steps in detail.

5.2.4.1 The New Orientation Vector

In order to establish the spring location after the movements we employ a vector called the orientation vector whose purpose is to indicate where the spring lies between the movement limits $\mathbf{u}_e$ and $\mathbf{u}_r$ on the movement surface. Depending on the material properties, the spring's location will vary between the two limit vectors as shown in figure 5.9 (a) for the limits given in figure 5.3 (b). The orientation vector therefore spans the entire movement surface (region). Therefore an equation for the orientation vector can be expressed in terms of the limit vectors as:

$$\mathbf{u}_o = \alpha \mathbf{u}_e + (1-\alpha)\mathbf{u}_r \tag{5.5}$$

where α represents the deformation characteristics of the object under consideration. If the object is very elastic in nature, then the orientation vector is expected to be closer to the elastic limit. Alternatively if the springs are defined with a higher stiffness then the orientation vector approaches the rigid limit. Thus, the parameter α can also be considered as a control coefficient for the spring movement.

Equation 5.5 determines the direction of the orientation vector whose magnitude is determined based on the spring's original length. In figure 5.9 (b), the initial spring is now moved to a new position, which is described as new the orientation: P_1^0. One end of the spring is already known (i.e. the active point P_0^n) and the other end is

established by the orientation vector. The semi-active point is then moved to its new position by a vector known as the movement vector $\mathbf{V}_m$. Since the original spring length is already known the new orientation can easily be established, as shown in figure 5.9 (b). The movement vector is given by:

$$\mathbf{V}_m = P_1 P_1^0 \,. \tag{5.6}$$

Finding the deformation is examined next.

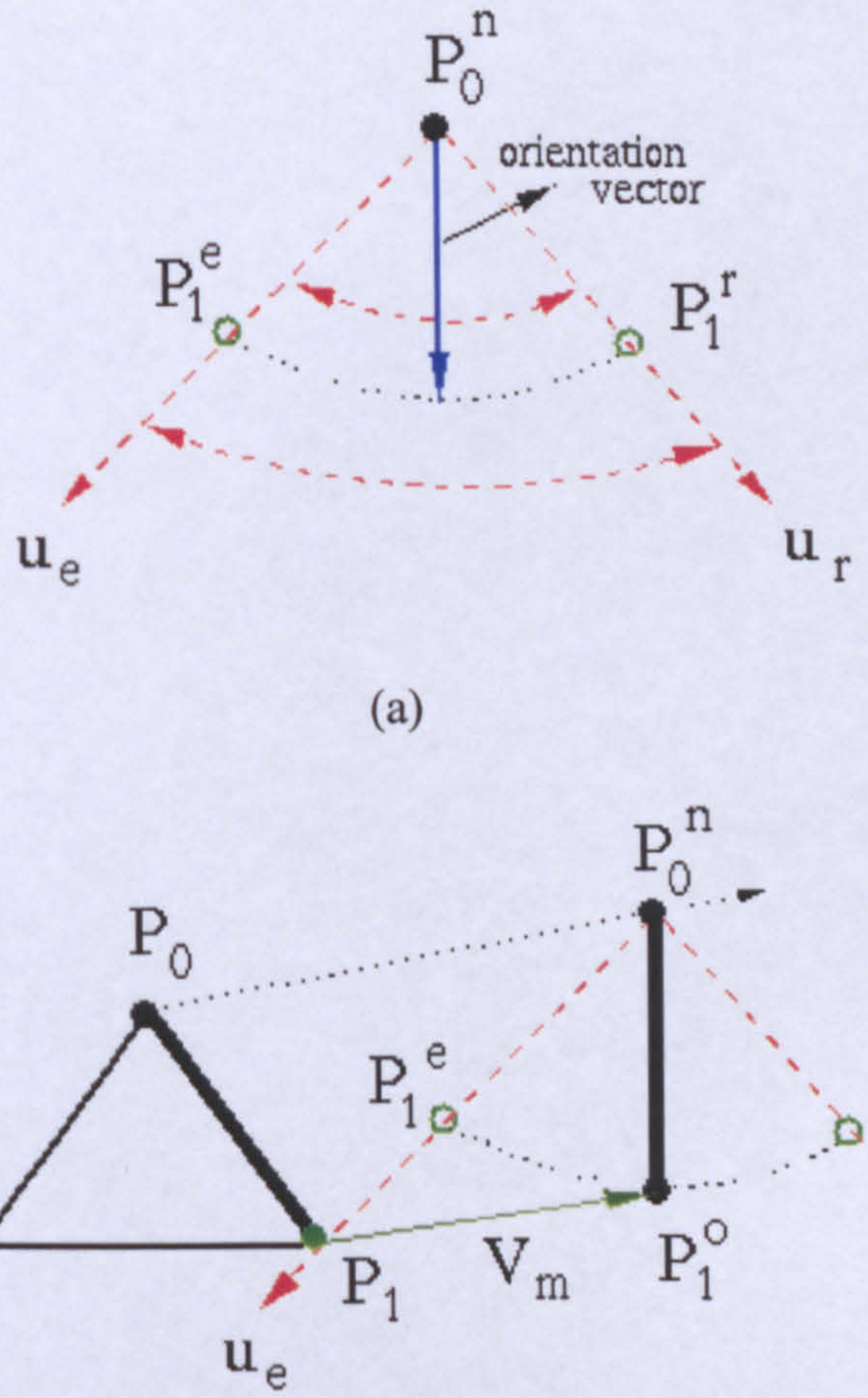


Figure 5.9. The orientation vector varies from the elastic limit to the rigid limit (a) and a new location for the spring after the movement is between these limits.

5.2.4.2 The Deformation of Active Springs

Once the new locations of the springs are determined with the help of the orientation vectors, the deformation of the spring is considered. Different types of spring will follow different sets of rules in the deformation process. Springs are classified, by definition, as active, semi-active or boundary springs. Boundary springs may follow special rules depending on the application requirements, since they form the connection between different objects. Deforming active springs is relatively straightforward because only one end of the spring is deformed. The deformation algorithm first finds out how much deformation occurs and then determines the direction of the deformation.

5.2.4.2.1 Magnitude of the Deformation

As in the mass-spring system's algorithms, the current spring lengths are found and compared with their initial (pre-set) lengths. There are three possible outcomes from this comparison. If there is no difference between them, there will be no spring deformation. The current spring length may be larger then its rest length. In this case, the spring is being stretched. In opposite case, where the initial spring length is larger than the current spring length, thus the spring is being compressed. In both cases the spring is deformed and the amount of the deformation needs to be determined.

The magnitude of the deformation may be calculated using many different formulations. Here, we use the following:

$$def = d_{\max}(1 - e^{-\beta \frac{dr}{r_0}}) \tag{5.7}$$

where $d_{\max}$ is the allowed maximum stretch or compression, dr is the difference between the current and the rest length and the slope parameter β represents the

deformation rate. A typical plot of such a function is given in figure 5.10 for nominal values of the variables.

As seen in figure 5.10, the deformation has an upper limit set by $d_{\max}$ (1 in this case) and depending on the deformation rate β, the spring's deformation will reach a maximum at varying speeds (at different values of dr). For example, when dr=1.5, (from the figure) def is either 0.7 or 0.9 depending on the function used. As can be seen from the figure 5.10, two different functions are plotted for the two different values of the β. These parameters allow us to model different objects with distinct deformation characteristics. This equation also allows continuous deformation. In the ChainMail algorithm, however, links are only deformed if their lengths violate the length constraints.

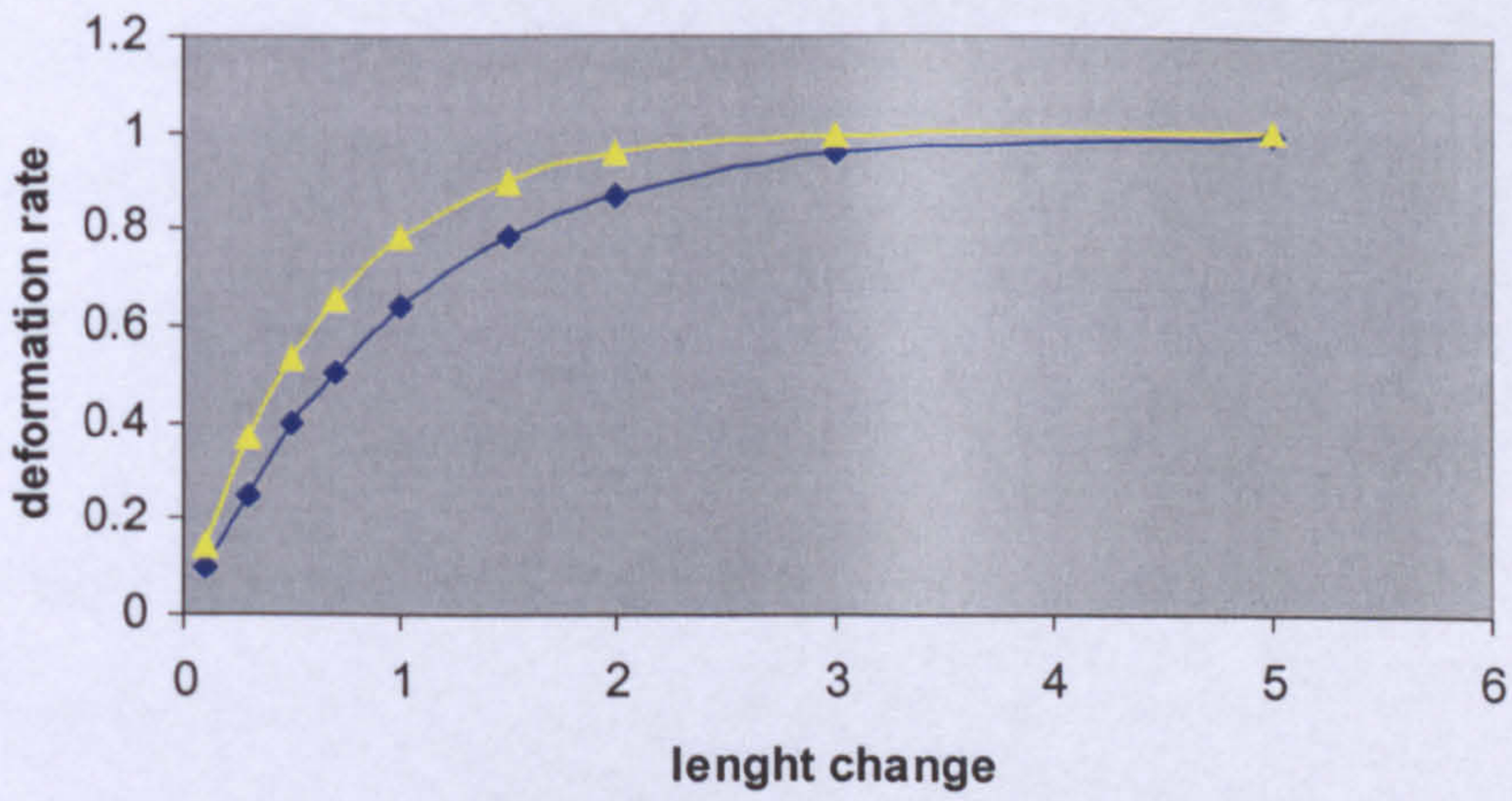


Figure 5.10. Deforming the springs, based on the parameters, $d_{\max}$ and β.

5.2.4.2.2 The Direction of the Deformation

In our work, for simplicity we choose the direction of the orientation vector as the deformation direction. If the spring is being compressed, then the deformation

direction is opposite to the orientation direction. The amount of deformation is then subtracted from the original spring length, leaving the current spring length shorter than the original length. If the spring is being stretched, then the deformation direction is the same as the orientation direction. In this case the deformation is added to the original spring length elongating the spring. The direction of the deformation, however, can be effected by many factors, such as the connected springs.

5.2.4.3 The Deformation of Semi-active Springs

Deforming semi-active springs is slightly more complex than deforming active springs. The strategy is different for semi-active springs because unlike active springs both their endpoints are moving at the same time. Semi-active springs are allowed to move and deform freely during the deformation of active springs. Then the deformation algorithm checks if the semi-active springs violate the spring length criteria. If the maximum stretch and the maximum compression conditions are satisfied, no action is taken. Otherwise, the semi-active springs are deformed to meet the set conditions. As in the deformation of active springs, here as well, the deformation magnitude and direction will have to be determined.

5.2.4.3.1 The Deformation Magnitude

The process for finding the deformation magnitude of semi-active springs is the same as that given for active springs. Since both endpoints of the spring are moving the spring is deformed at both endpoints. The distribution of the deformation to the spring endpoints depends on the distances traveled by each endpoint of the spring. If one end of the spring travels a longer distance than the other end, this implies that this endpoint is leading and it is considered as an active end. The other end, which travels a shorter distance, is known as the semi-active endpoint. The semi-active endpoint is

subject to more deformation than active endpoint. This is based on the assumption that the spring is pulled by its active endpoint. The semi-active endpoint follows and lags behind causing deformation. The amount of deformation is distributed according to following formulation:

$$def_A = \frac{r_B}{r_A + r_B} def$$

$$def_B = \frac{r_A}{r_A + r_B} def \tag{5.8}$$

where def_A and def_B ($def_A + def_B = def$) represent the required deformations at the two endpoints of the spring, r_A and r_B give the distances traveled by each endpoint of the spring.

5.2.4.3.2 The Direction of the Deformation

The direction of deformation can be chosen to be in the spring direction. Any deformation (i.e. reduction or increase in spring length) occurs in the spring direction. A deformed triangle is shown in figure 5.11, where points P_1 and P_2 are adjusted based on deformations on the active springs s_1 and s_2. The deformation based on the semi-active spring, s_3, is shown in the same figure. The new position for the points are P_1^n and P_2^n. Arrows indicate the direction for the deformations of the spring s_3.

Choosing the spring vector as the deformation direction has, however, a downside. The lengths of active springs are affected by this adjustment. In some situations this may yield constraint violations for active springs. The deformation direction can be chosen in a way that the length of active springs remains the same after deforming the

semi-active springs. The deformation algorithm handles this problem in the fine-tuning phase examined next.

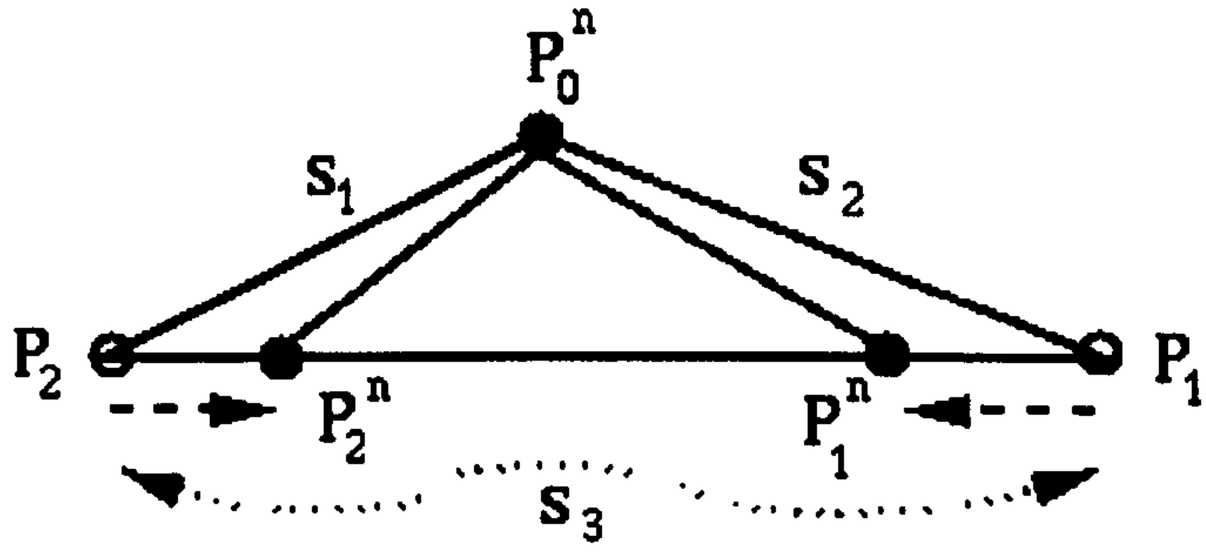


Figure 5.11 Deforming semi-active springs.

5.2.5 Fine Tuning

Active and semi-active springs are deformed and the positions of their vertices are updated using the procedure discussed above. Some of the springs, however, may still violate the length criteria set in section 5.2.3.2. In the following section we provide a method to deform springs in order to ensure that these constraints are satisfied. Before this phase of the algorithm starts, the new orientation, the deformation and therefore the new deformed positions have already been determined. This part of the algorithm revisits all the springs (active or semi-active) processed at previous states. Their lengths are found and their pre-set length criteria are checked. If any of these springs is in violation of their length constraints, their lengths are adjusted. In this sense the fine-tuning phase is very similar to the elastic relaxation phase used by the Chainmail algorithm. The adjustment part is however different. Here we use the same method as that for finding the deformation, as discussed in section 5. The direction of the adjustments is chosen to be in the spring direction. This phase of the algorithm further enforces the spring length criteria already checked by the main body of the algorithm, as discussed in section 5.2. Thus the spring elongation problem is addressed for a second time.

This phase of the algorithm also represents the end of each wave deformation pattern. After fine-tuning is performed, the deformation algorithm moves to the next step, i.e. a new deformation wave. This part of the algorithm may be optionally included into the deformation algorithm. If it is, the algorithm becomes a two step approach and slows down.

5.2.6 Ending the Deformation Propagation

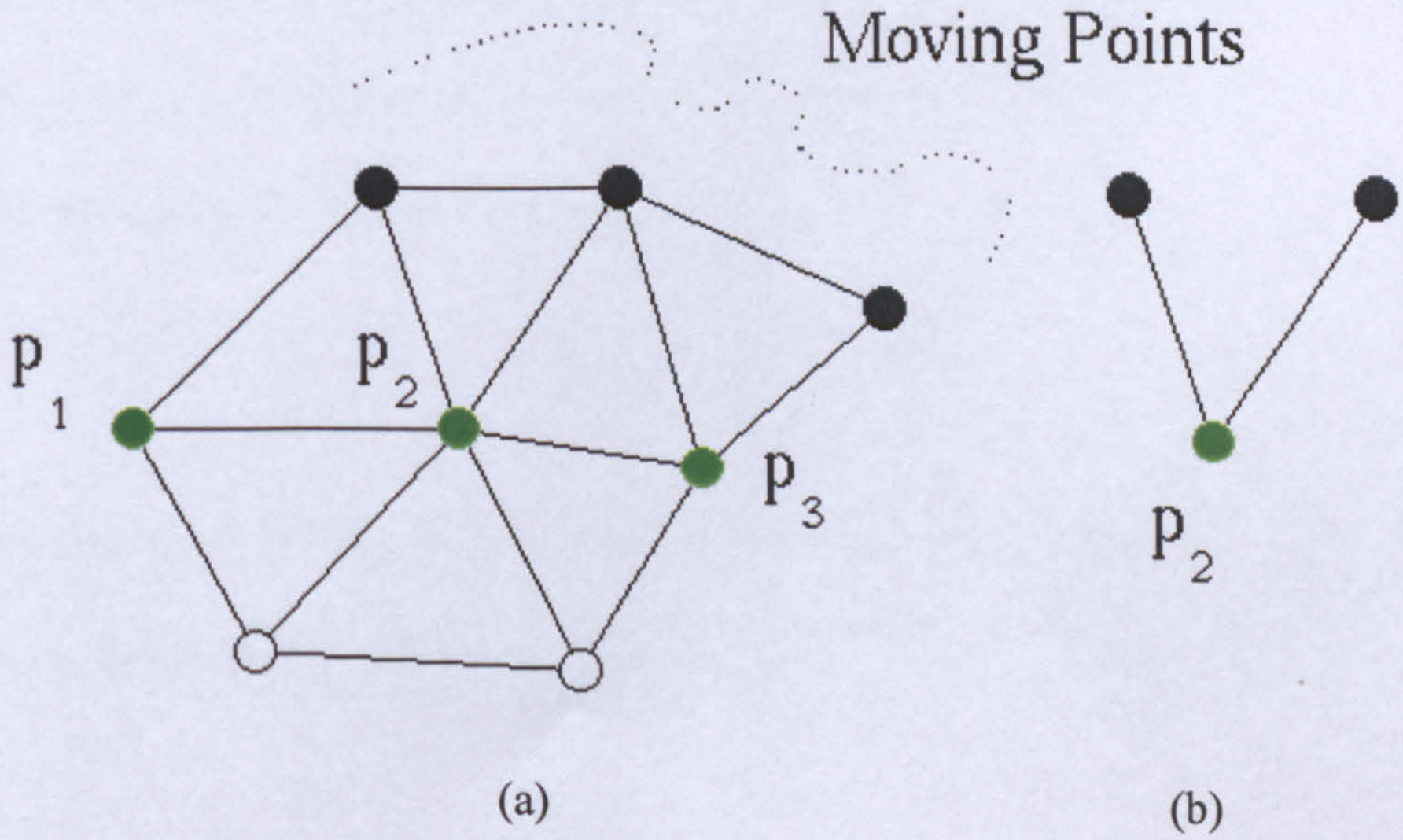
The deformation can be ended when all the points and springs in the mesh are processed. Alternatively, cut-off constraints can be defined. We may, for example, set a condition such that if the movement is less than a certain amount, then propagation in that direction is stopped. This property of the algorithm provides extra control not just on the deformation but also on the deformation time. The deformation can have different propagation speeds in different directions and therefore can be terminated at different times in different direction. The connection to other organs or objects can be simulated using this technique.

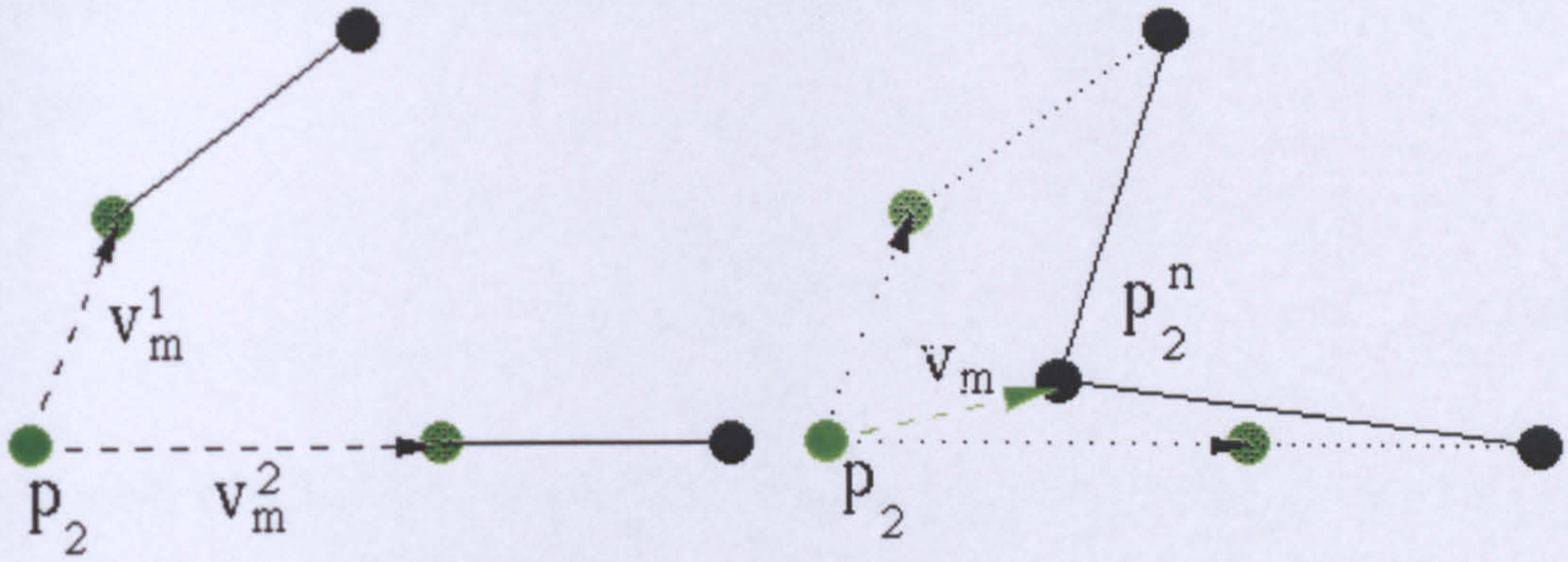
5.2.7 Special Cases

The deformation algorithm has to be able to handle the situations where there may be more than one spring affecting one point. From the nature of the modeling, each spring can be analyzed (deformed) individually, but points are generally under the influence of more than one spring. Therefore one single point may be subject to change of position and deformation by several springs. The other special case may be described as cell conversion or shape alteration. In this case the building blocks, triangles or tetrahedral elements, are forced to change their original shapes. Consequently the deformation starts from changed (altered) shapes and yields unrealistic results. We examine these situations in detail in the following sections.

5.2.7.1 Multiple Movement

A simple 2D mesh is shown in figure 5.12, where points given in black indicate moving points. Consequently the points represented by P_1, P_2, P_3 (in green) are semi-active points and are subject to deformation. Point P_1 is under the influence of one spring but the other two points are pulled by two moving springs. Therefore for the point P_2, for example, the combined effect must be found. In figure 5.12 (a), the initial configuration (before movement) is given. Figure 5.12 (b) shows the point P_2 and the connected moving springs separately. After the movement, the new positions of the springs are shown in figure 5.12 (c) where point P_2 is forced to move in two different directions, terminating in two different locations. These locations are indicated by the vectors $\mathbf{v}_m^1$ and $\mathbf{v}_m^2$. These vectors show the movement paths for this particular point. Point P_2, however, will not move in any of these directions but will follow another vector and terminate at a center location, which represent the combine effect of both pulling springs. This center location is found by combining the path vectors $\mathbf{v}_m^1$ and $\mathbf{v}_m^2$. This center location and the movement are shown in figure 5.12 (d), where the movement vector is given by $\mathbf{V}_m$.





(c) (d)

Figure 5.12 Multiple movement.

There could be a number of ways to find the center location and the movement vector, $\mathbf{V}_m$. The most obvious solution is to compute an average vector. This can be formulated as:

$$\mathbf{V}_m = \frac{\mathbf{v}_m^1 + \mathbf{v}_m^2}{2}. \tag{5.9}$$

This solution may not produce very realistic results simply because the effect of the larger of the two movements is not taken into account. One would expect that P_2 will move closer to the direction represented by vector $\mathbf{v}_m^2$. This is because P_2 is pulled and forced to travel a grater distance in this particular direction rather than in the other direction. A weighting method can be employed to take into account the vector magnitude as given in the following formulation:

$$\mathbf{V}_m = \frac{r_m^1 \mathbf{v}_m^1 + r_m^2 \mathbf{v}_m^2}{r_m^1 + r_m^2} \tag{5.10}$$

where r_m^i represents vector magnitude (i.e. the distance traveled). Naturally, vectors with larger magnitudes will have grater influence and the movement and center

locations will appear closer to them. Equation 5.10 can be given in more general form as follows:

$$\mathbf{V}_m = \frac{\sum r_m^i \times \mathbf{v}_m^i}{\sum r_m^i} \quad . \tag{5.11}$$

This method is used in finding the orientation, as well as, in finding the deformation of semi-active points. The orientation vector is found using equation 5.3. This vector moves an attached semi-active point. There may be several active springs connected to a single semi-active point, giving several orientation vectors. Each of these vectors will try to move the semi-active point in a different direction as shown in figure 5.12 (c). The above method is employed to find only one path to a center point. Each spring now has new and different orientations but a connected point only moves into one location as shown in figure 5.12 (d).

In the deformation process, a single point may be forced to deform by different amount in different directions, because of deformations from various connected springs. A center point corresponding to a combination of these deformations is also found using above method.

5.2.7.2 Shape Alteration (Cell conversion)

The mass-spring systems algorithm works by measuring the current spring length and comparing it with the original spring length. The difference is then used, as discussed in chapter 2, in order to determine the deformation. The downside of this technique is that the deformation law does not consider any shape violation because it is not able to detect shape alterations. Therefore shape conversion (cell conversion) is likely to happen in some situations where a very large force is applied or the shape is altered by external movements (Ganovelli et al 1999).

Shape alteration occurs, in this study, by external movement of vertices. A new shape emerges when one of the vertices of the triangle passes through one of its edges. Figure 5.13 shows a shape conversion. The triangle on the left shows the initial shape before the movement. Point P_3 is externally moved to a new location given by P_3^n. This movement and the new triangle formed are shown on the right hand diagram of figure 5.13. If the deformation starts from the newly shaped triangle, inaccurate results will be produced. This is because, as in the mass-spring systems algorithm, this deformation algorithm measures edges of the new triangle, compares them with the original setting and finally updates the positions of the vertices. This process clearly does not take into account the movement of point P_3.

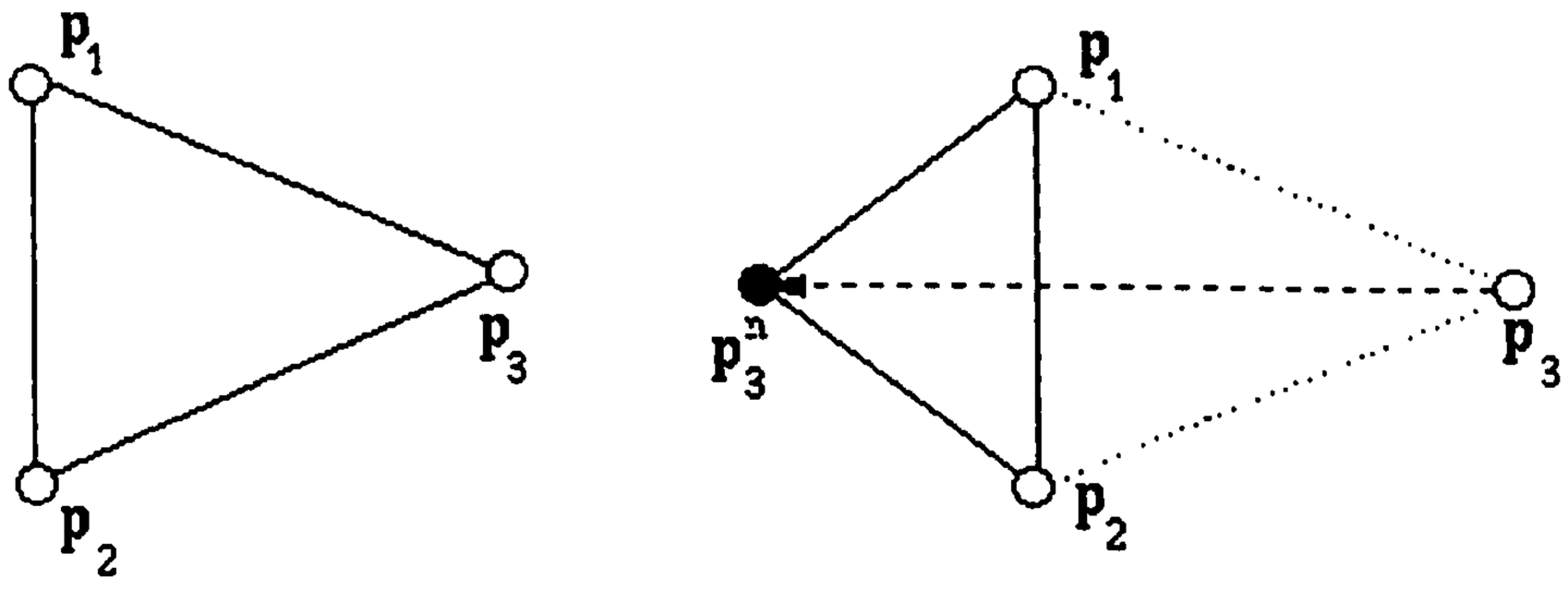
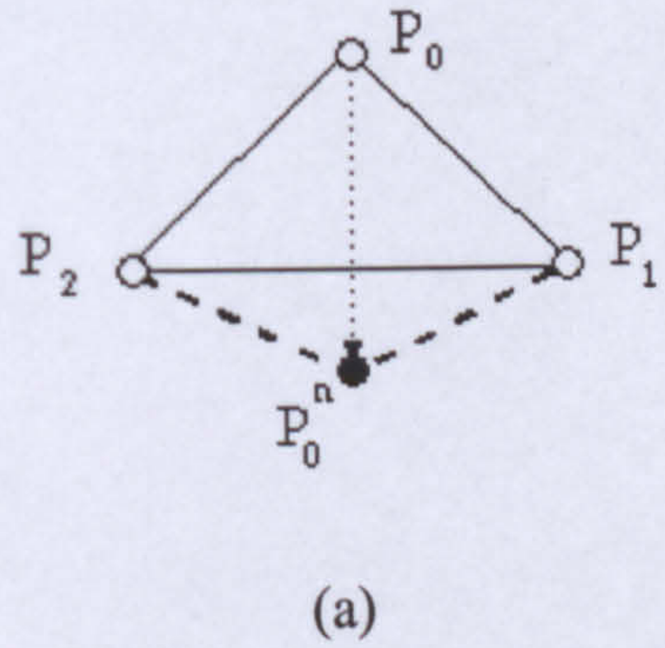


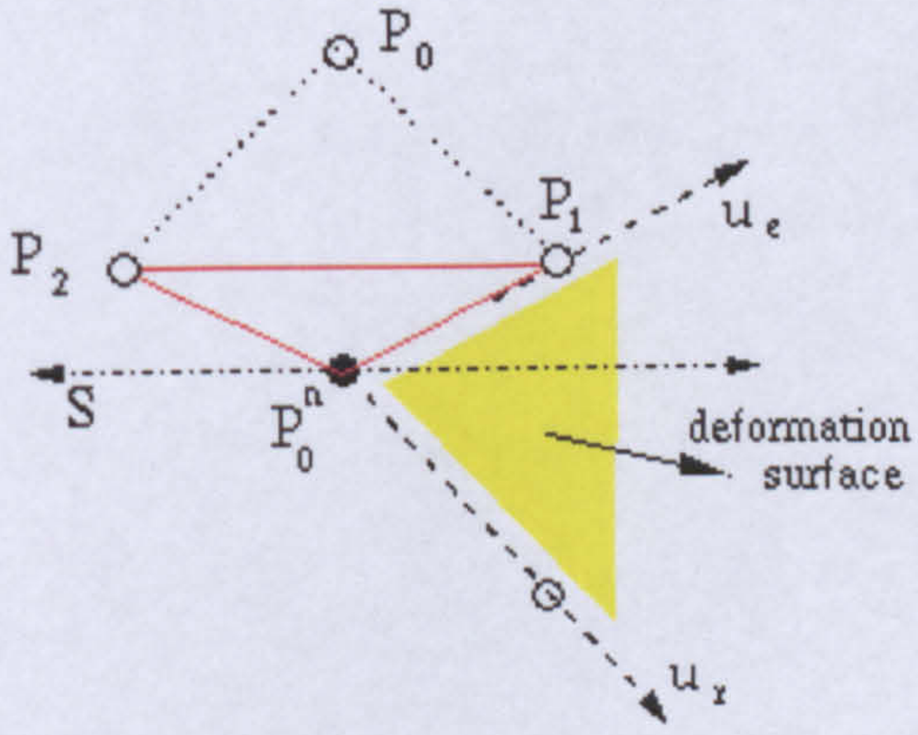
Figure 5.13 Example of shape alteration.

In theory if one of the vertices moves to a new location it is expected that the connected springs (forming the triangle) should move and deform accordingly. Figure 5.14 (a) shows a triangle ($P_0P_2P_1$) with a vertex that is moved to a new location given by P_0^n. As can clearly seen from the figure 5.14 (a) this vertex passes through one of the edges of this triangle and creates a new triangle given by $P_0^nP_2P_1$. The mass-

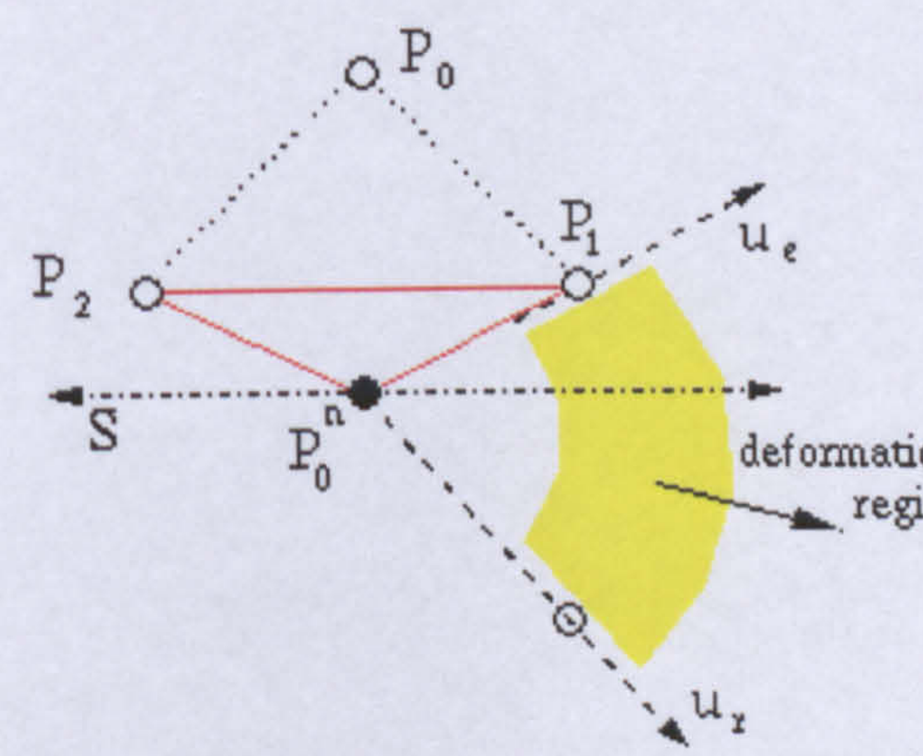
spring systems algorithm performs the deformation based on this newly generated triangle. This will of course cause problems in terms of accuracy and topology changes.



(a)



(b)

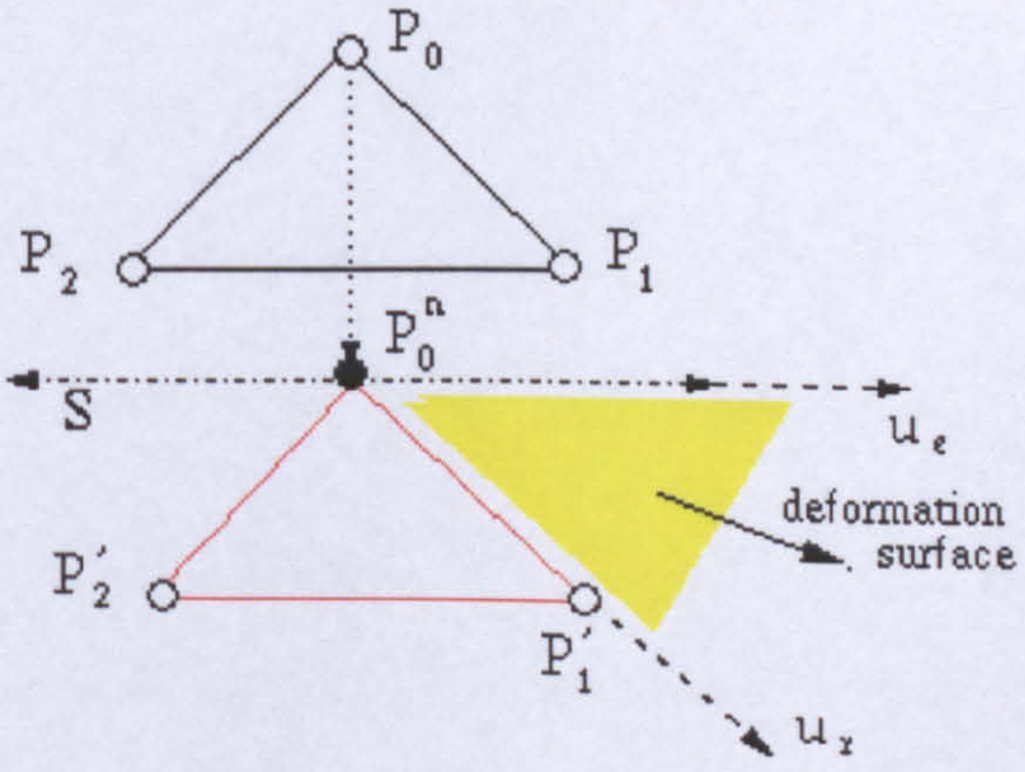


(c)

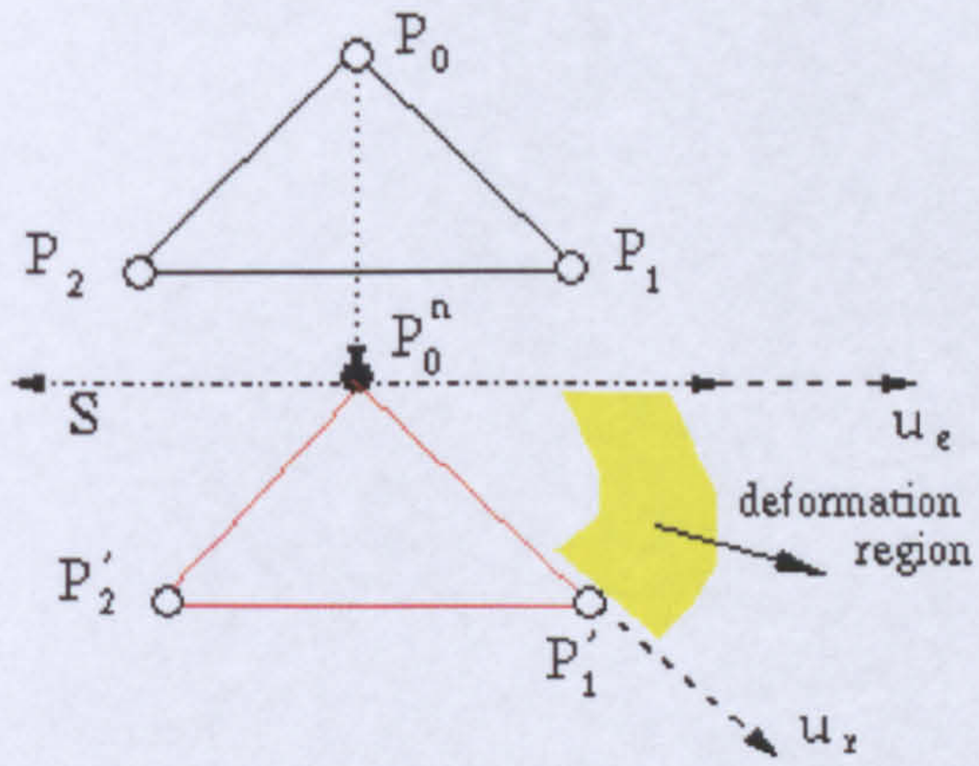
Figure 5.14 Shape alteration (a) the deformation surface (b) and deformation region (c).

Figure 5.14 (b) represents the deformation surface while 5.14 (c) shows the deformation region for the developed algorithm. If the shape alteration was ignored, the deformation would have been searched in this region. The shape of the triangle at the beginning of the deformation process is completely different to the original shape. The deformation results will be completely different than of those a simulation that starts from original shape. In addition, the result may not even be a triangle.

A simple modification, moving the original triangle to a new location, will solve this problem. It is our assumption that moving point P_0 means moving the triangle below to its new location. Let us assume that the triangle is not subject to deformation. In this case the triangle moves (transition) to a new location preserving its original shape. This is illustrated by the triangle $P_0^n P_1^{'} P_2^{'}$ in figure 5.15. The deformed triangle then must lie between the new non-deformed triangle and the surface defined by S where the active point P_0^n lies. The deformation surface and region defined between them are also shown figure 5.15 (a) and (b), respectively.



(a)



(b)

Figure 5.15. The new deformation surface (a) and deformation region (b).

The rigid movement vector $\mathbf{u}_r$ is derived as defined in section 5.2.3.1. The difference in this case is in defining the elastic limit vector. Recall that the elastic limit vector is formed between the moved point and the original point, as given in figure 5.14. In order to be able to handle cell conversion, the elastic limit vector is redefined. Since the new surface (S) is now a minimum limit, we can choose the surface equation as the elastic limit, thus preventing the orientation vector from going behind this limit (surface) as illustrated in figure 5.15.

There are two ways of detecting any shape alteration. Let us define the angle between the elastic limit vector and the new plane (S) vector as θ_1, the angle between the rigid limit vector and the new plane vector as θ_2 and the angle between the limit vectors themselves as θ. The first case is shown in figure 5.16 (a), where crossing occurs inside the triangle. In the first case, as seen from the figure 5.16 (a), the sum of two angles, θ_1 and θ_2 is equal to the angle between the limit vectors and this sum is less than 180 degrees. This implies that shape alteration occurs because the new plane is

located between them and the elastic limit vector is replaced by a vector parallel to the plane as follows:

$$\mathbf{u}_e = \mathbf{S} \tag{5.12}$$

The second case occurs outside the triangle, in figure 5.16 (b) the sum of the angles $(\theta_1+\theta_2+\theta)$ is equal to 360 degrees. In the second case, the direction of the new vector is just opposite to the plane vector. In both figures, the triangle in red depicts the rigid movement without any rotation or deformation.

$$\mathbf{u}_e = -\mathbf{S} \tag{5.13}$$

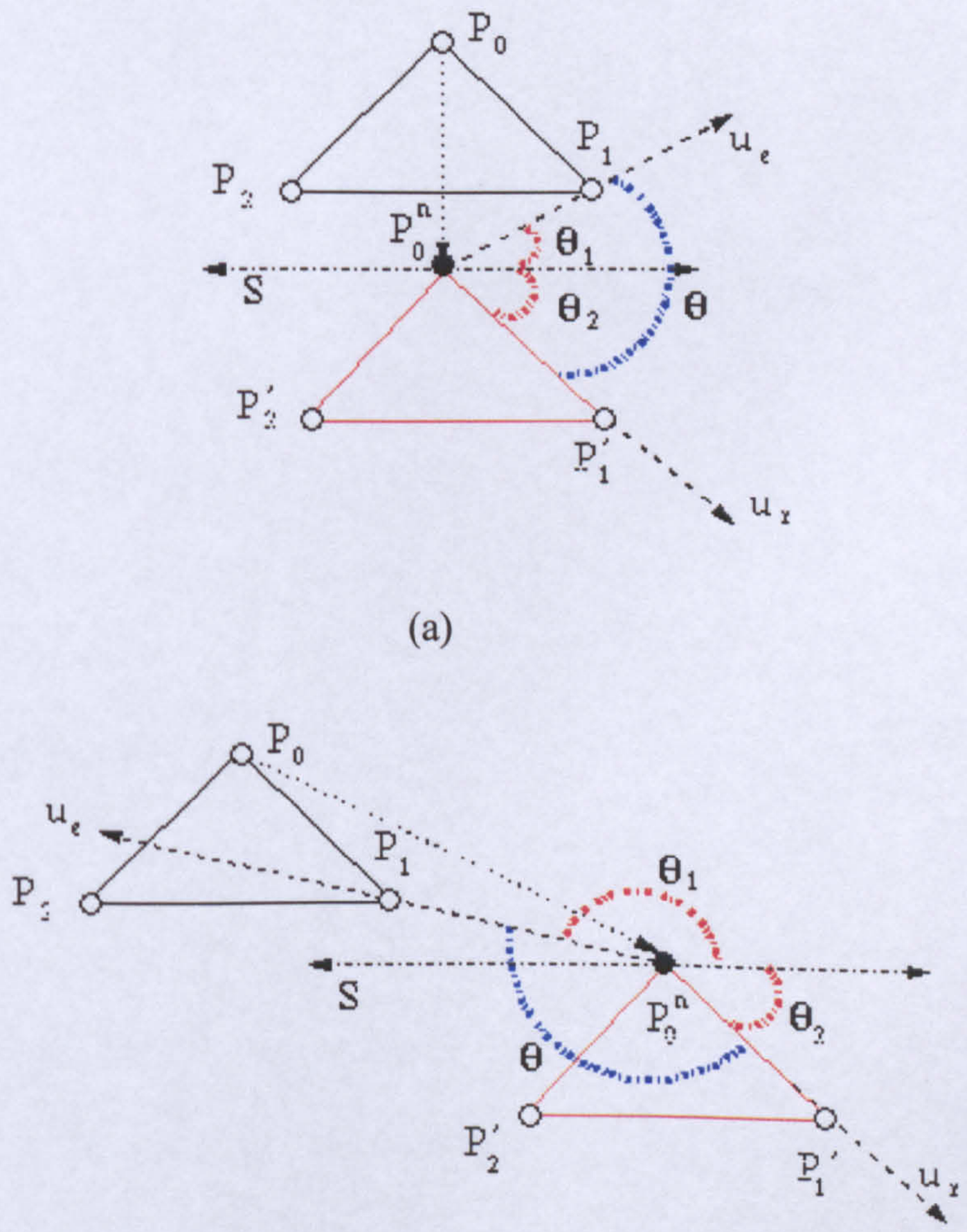


Figure 5.16 Shape alteration and its detection.

This process is illustrated in figure 5.17. A simple structure is simulated using both the mass-spring system's algorithm and the new algorithm. The results of the mass-spring system's algorithm are shown in blue and those of the new algorithm in red. In first two diagrams of this figure both algorithms appear to work, but in the last one where cell alteration occurs, the new algorithm successfully simulates this special case while the MSS algorithm cannot detect the situation and can not simulate it correctly.

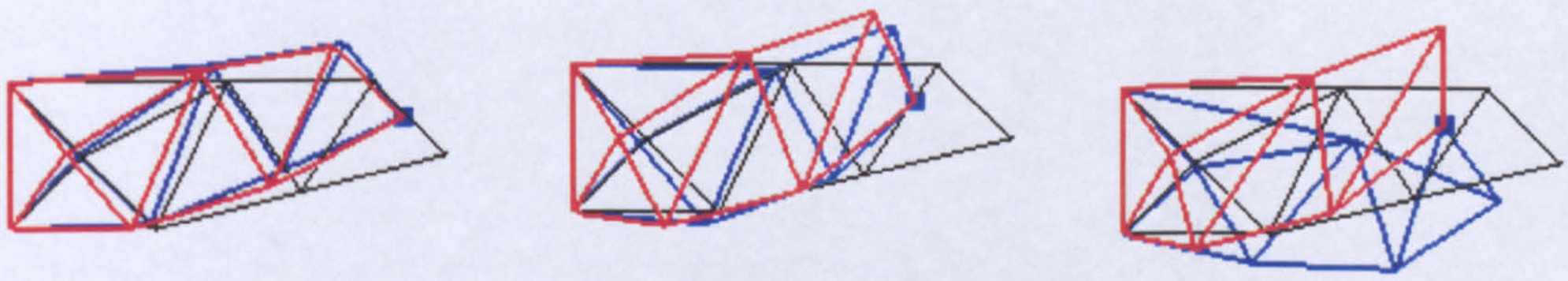


Figure 5.17 A simple simulation to demonstrate shape alteration and handling.

5.3 Summary of the Algorithm

The disadvantages of both the mass-spring systems and the ChainMail algorithms were presented in section 5.1. The new algorithm posses several advantages over them while avoiding their disadvantages:

- It operates on triangular meshes.
- It only requires one or two steps, therefore it is very fast.
- It is very suitable for interactive applications and force feedback.
- It uses a more sophisticated deformation law.
- Material properties can be embedded in deformation law.
- System parameters can be identified from real data.
- It handles both rotation and transition.
- It handles cell conversion (shape alterations).

- The spring elongation problem is resolved.
- It is open for future developments (developing new deformation techniques, including physical properties by means of identifying system parameters).

The new algorithm can be summarized by the following pseudo-code:

```
Initialize the network: Establish connectivity and initial conditions.
While (active points)
        Find: active springs
                For (each active spring)
                  If (spring ON)
                    Find: semi-active point
                      If (!Boundary point)
                              Find limits: u_r , u_e
                              Detect and prevent shape alterations
                              Find orientation
                              Find deformation
                              Turn spring OFF
                For (each semi-active point)
                    Update orientation
                    Update deformation
                    Find Semi-active springs
                For (each semi-active spring)
                    Determine violations
                    Find and distribute deformation
                    Turn spring OFF
        For (each spring)
        Do fine-tuning.
                If (! pre-set condition)
                Active points = semi-active points
Stop
```

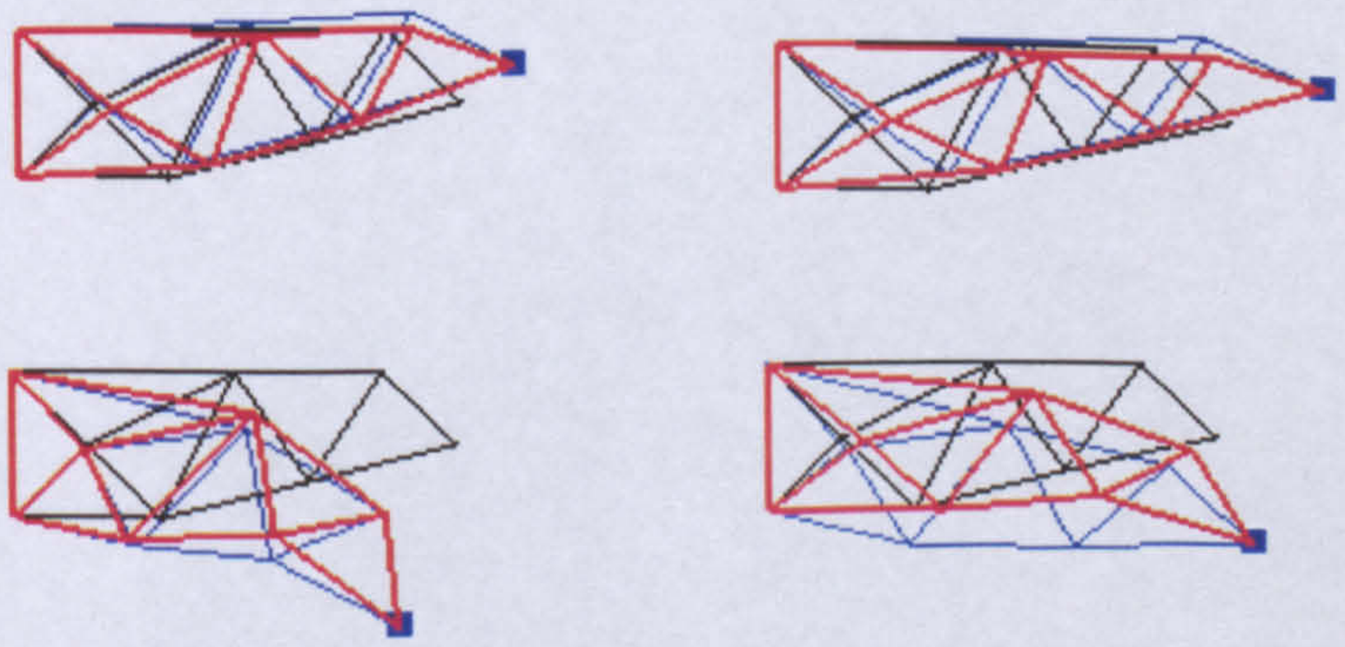
5.4 Applications and Results

The developed algorithm was used in a variety of simulation algorithms. We first give examples from simple 2D simulations in order to show clearly how the new method works in comparison to the mass-spring systems method. Next, we demonstrate the capabilities of the new algorithm in several 3D simulations. We also demonstrate the

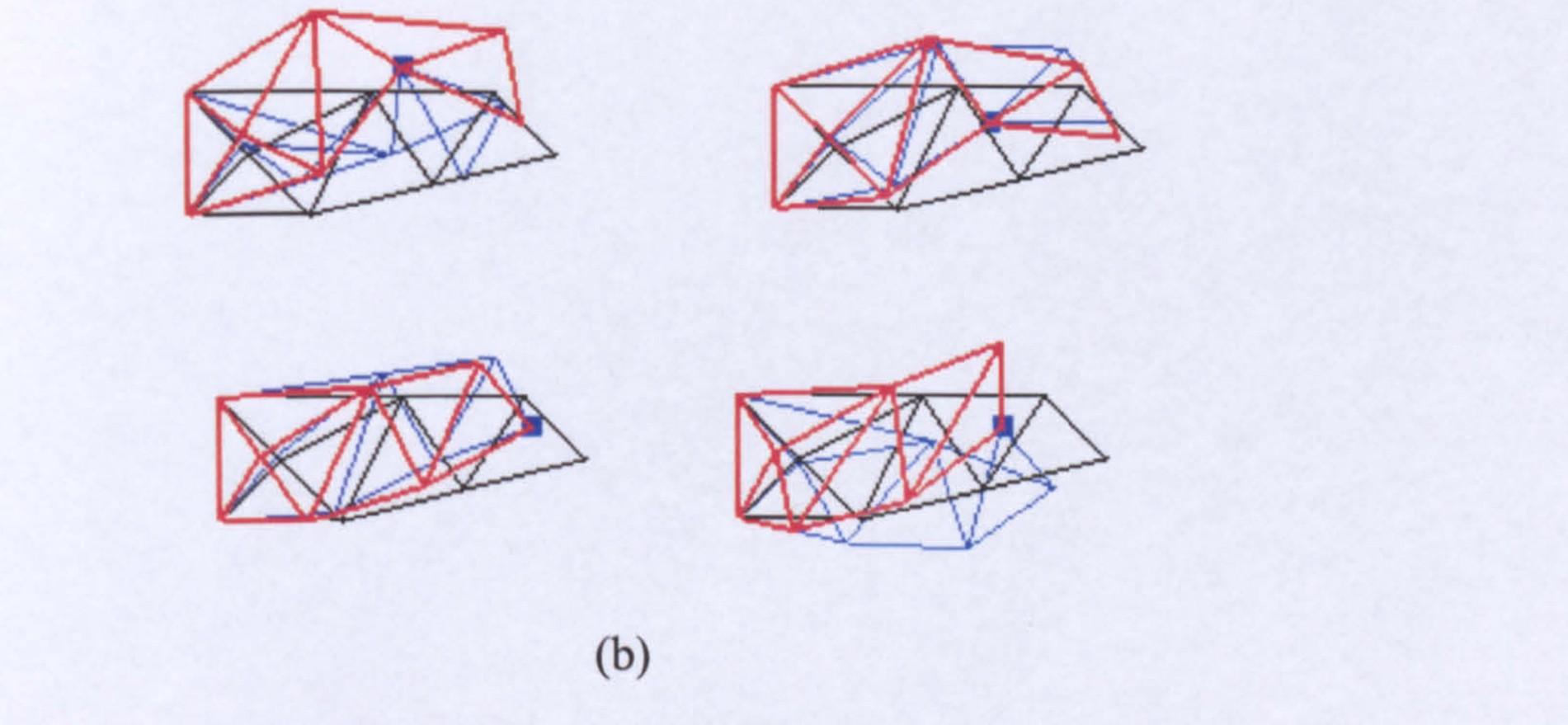
effects of model parameters on simulation outcomes. Finally, a craniofacial surgery simulation as performed in chapter 3 is repeated here with the new algorithm.

5.4.1 2D Applications

The algorithm is applied to the simulation of very simple structure, a 2D model consisting of few triangles. Points are grabbed and moved and the resultant deformation is simulated as shown in figure 5.18 where the leftmost points are constrained. In these figures, the original shape is shown in black, while blue represents the mass-spring systems algorithm output and red represents the output of the new algorithm. As can clearly be seen from these figures, the mass-spring systems and the new algorithms produce quite similar outputs in some cases, see figure 5.18 (a). In other cases, where one point passes through a triangle, the new algorithm simulates the deformation but the mass-spring system's algorithm fails to produce reasonable looking deformations, see figure 5.18 (b). Figure 5.18 (a) shows that both methods produce similar outputs for different deformation cases. In figure 5.18 (b), the first and last figures are examples of situations where the mass-spring system's algorithm fails to detect shape alterations, therefore producing unrealistic results.



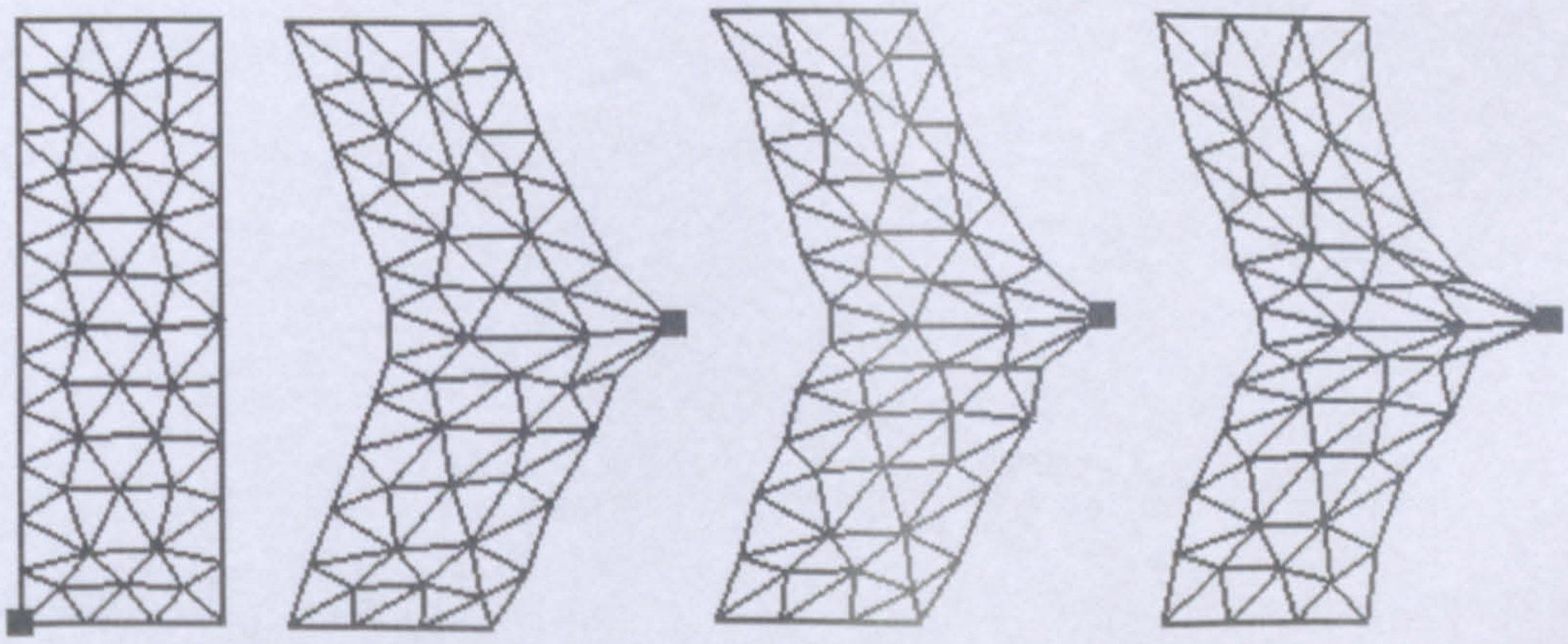
(a)



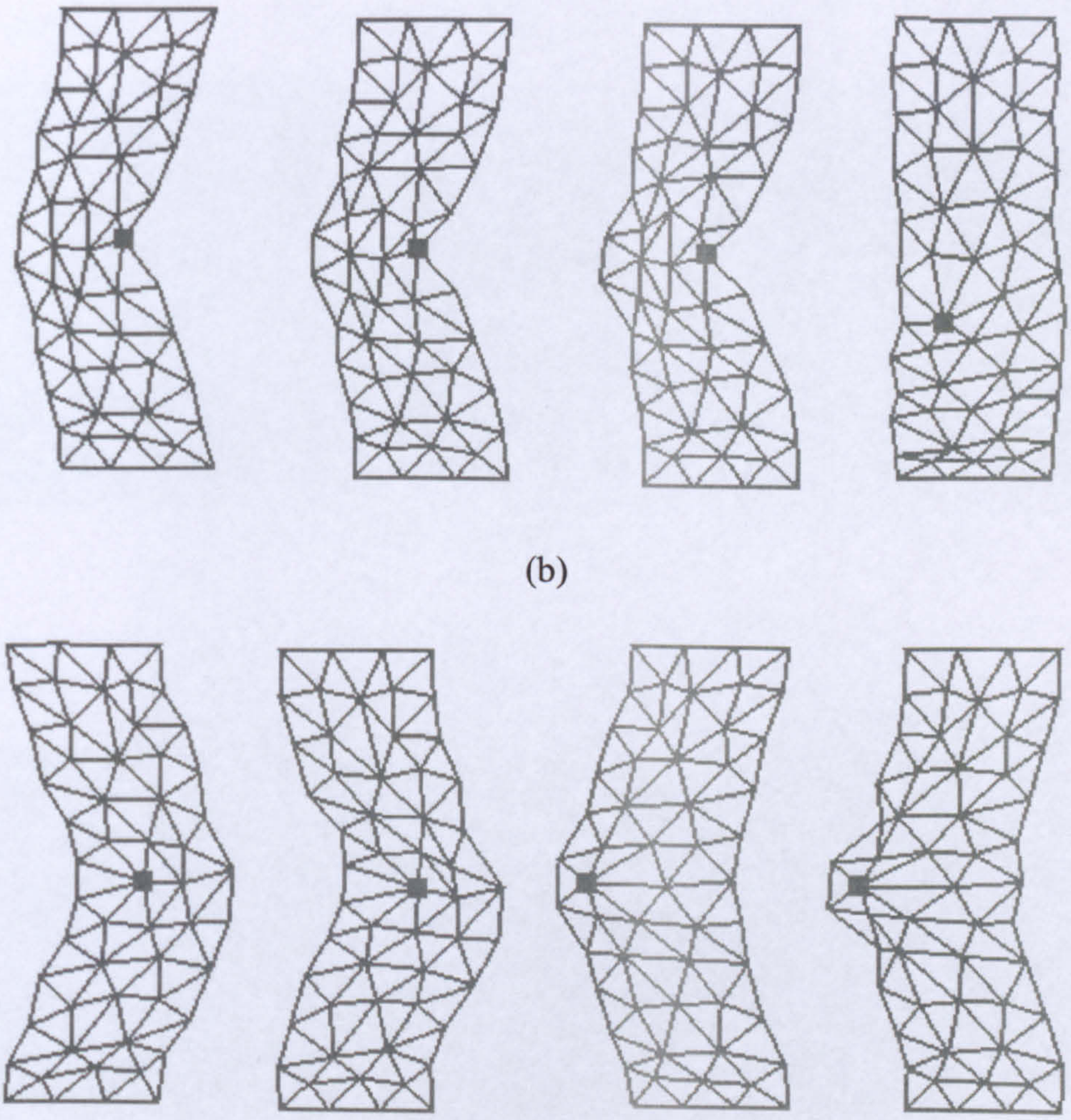
(b)

Figure 5.18 A 2D simulation example carried out using MSS (blue) and MSC (red) algorithms.

The second example is shown in figure 5.19, where a 2D rectangular surface is triangulated and constrained at the bottom and top vertices. The top left figure of row (a) gives the initial configuration. From left to right deformations are produced by changing the value of the parameter α (0.4, 0.6 and 0.8). The second row (b) represents the deformation of the rectangular object produced by moving one of the vertices on the interior of the object. As can clearly be seen, the triangles do not pass through each other or change their structure, which shows that the shape alteration part of the algorithm works very well. Similarly, the last row (c) shows various vertex movements in the interior of the object produced by varying parameter α.



(a)



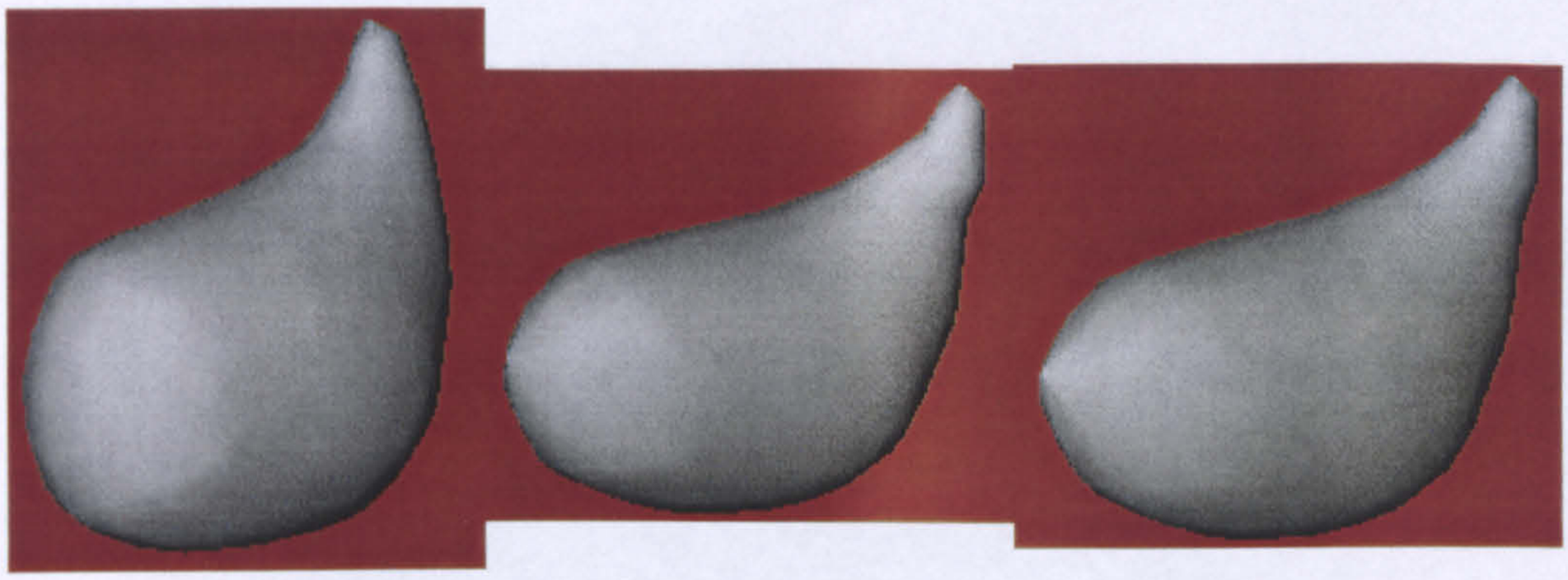
(b)

(c)

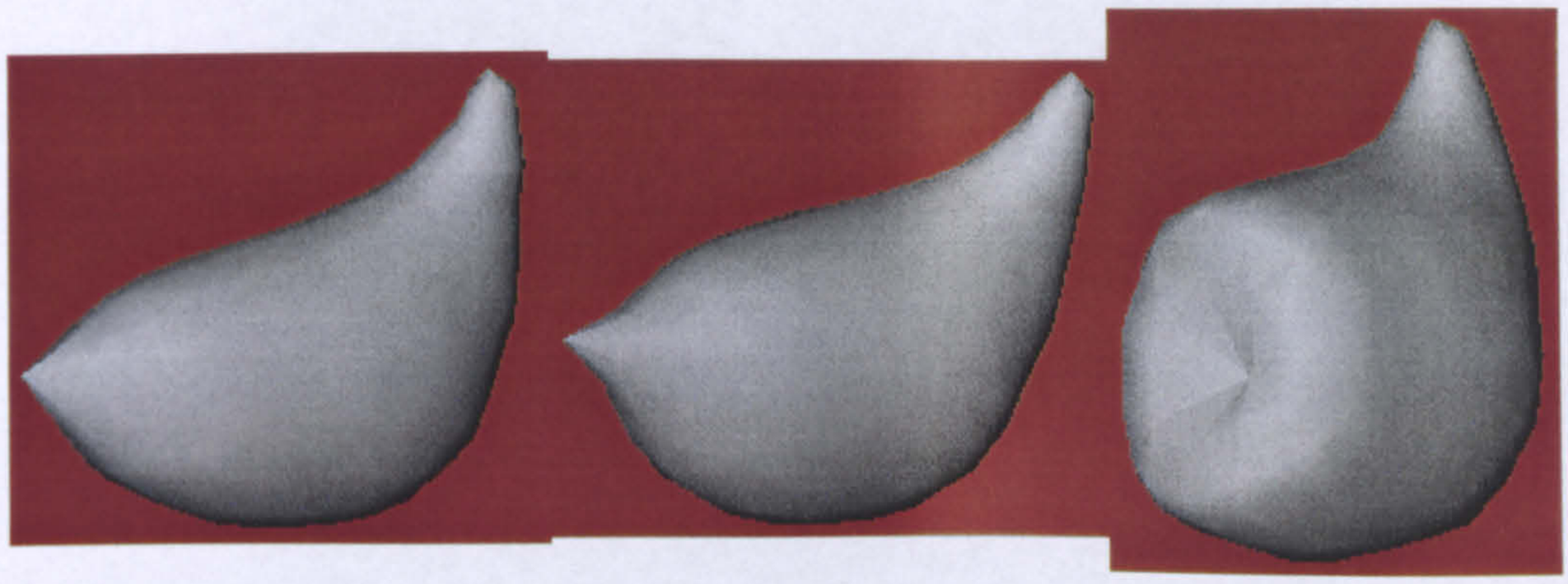
Figure 5.19 An example of a 2D application where the simulation parameters vary.

5.4.2 3D Applications

The second application is a simulation of a simple model *shown in figure 5.20, where 5.20* (a) *represents the original shape. The* model consists of 285 vertices and 566 triangles. *The model is constrained* at its top right tip. *The simulation is carried out* using different values of α (0.2, 0.3, 0.6, and 0.9). *The simulation time for this model* is less than 0.001 seconds. Pulling and pushing a specific vertex deforms the object. As can be seen from the figure different parameters values produce different deformed shapes (b,c,d,e). Figure 5.20 (f) is an example of a deformation where vertex is moved inside the object.



(a) (b) (c)



(d) (e) (f)

Figure 5.20 A simple model is simulated using different values of α .

A similar application is shown in figure 5.21, where a plastic duck is simulated. The results produced by using different simulation parameters are shown. Experiments were carried out by manipulating vertices of the objects (by pulling or pushing them).

These simulations demonstrate that plausible looking deformations can be achieved with the new method and that the outcome of the simulations can be controlled by system parameters. Tuning these parameters (α , β , and $d_{\max}$) will allow us to produce more realistic deformations. The parameter α , whose range is given in [0-1], controls the elasticity of the deformation. If the value of α approaches 0, this implies

that elasticity of the object is increased and similarly more rigid deformations are achieved when the value of α approaches 1.

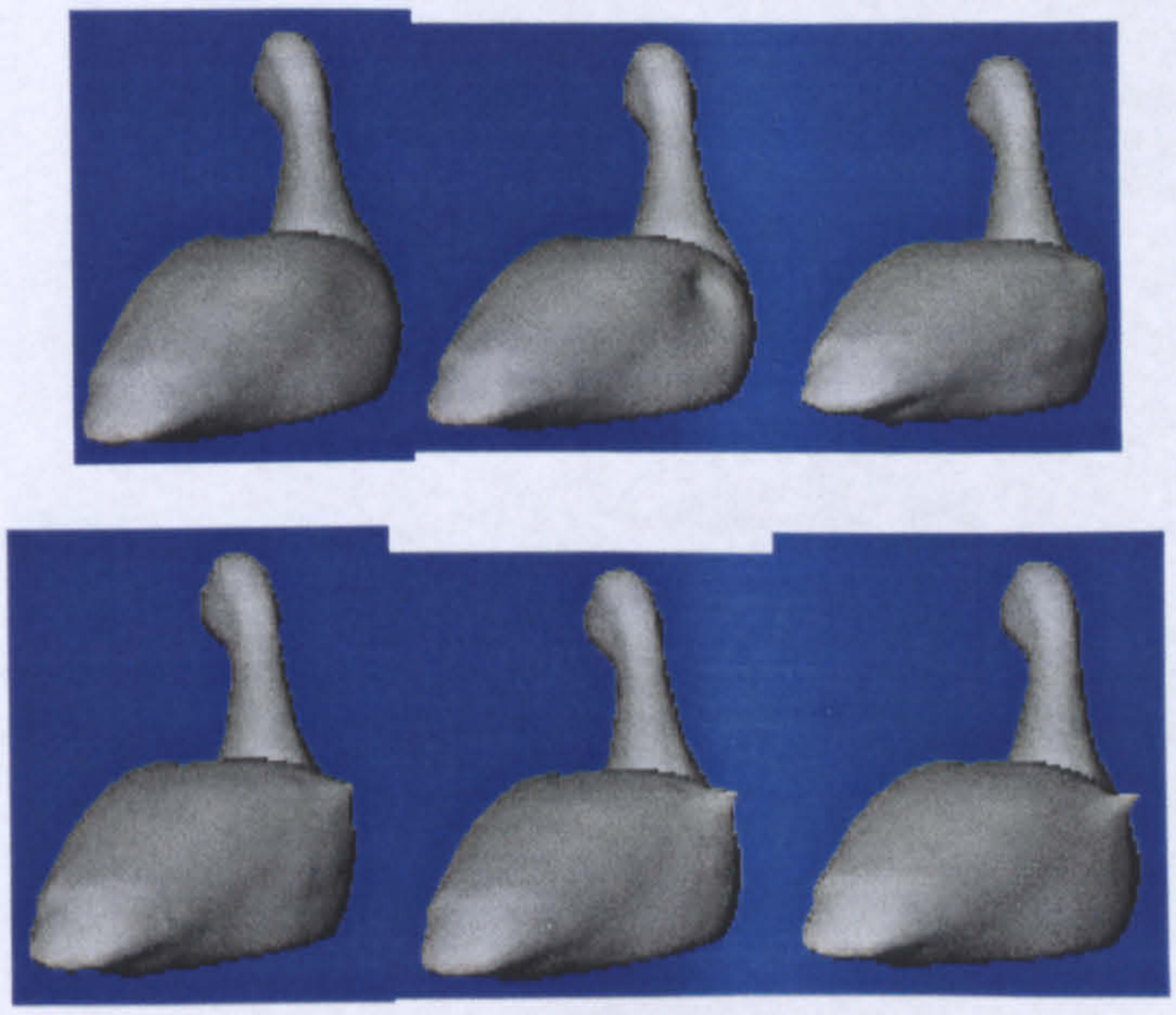
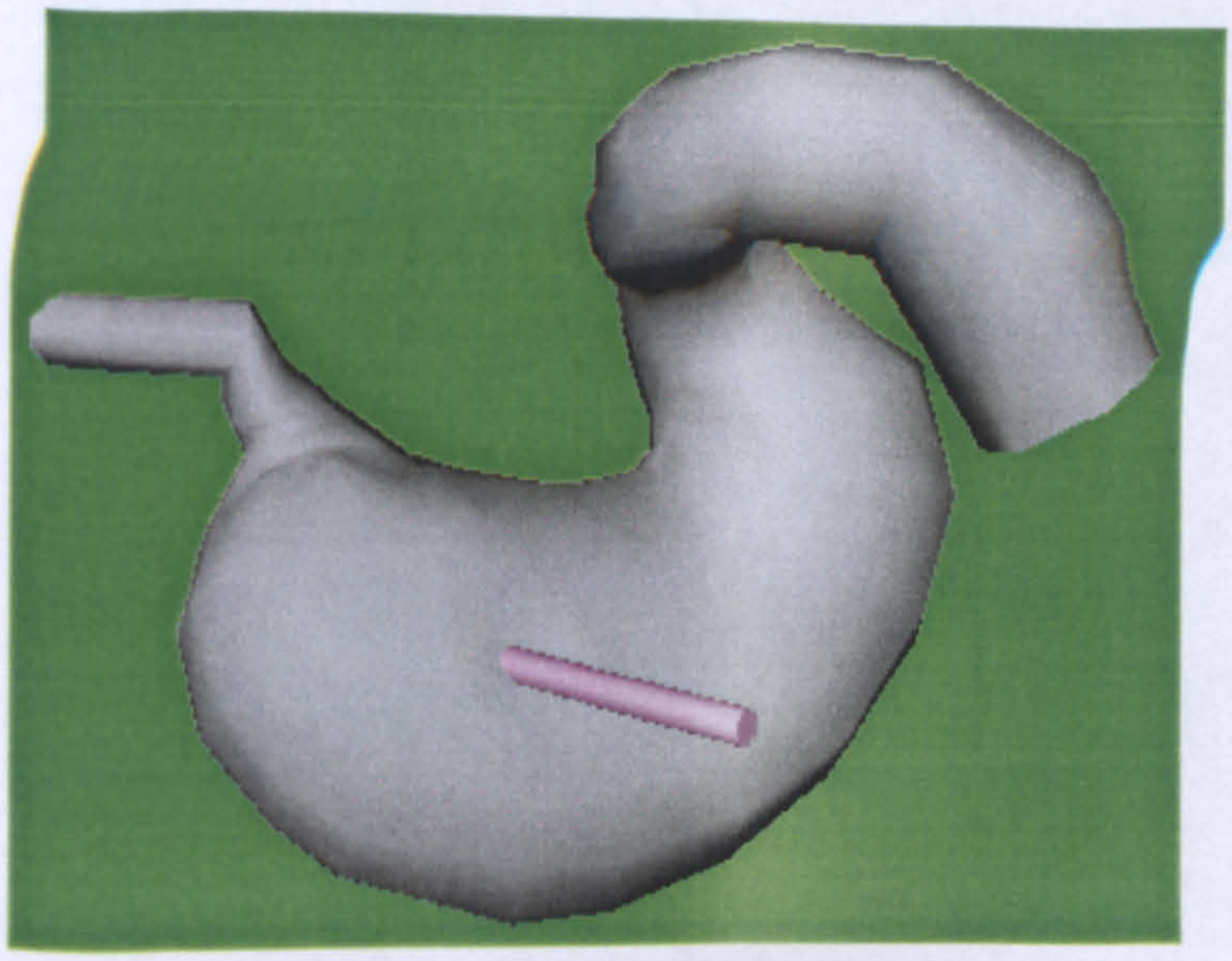


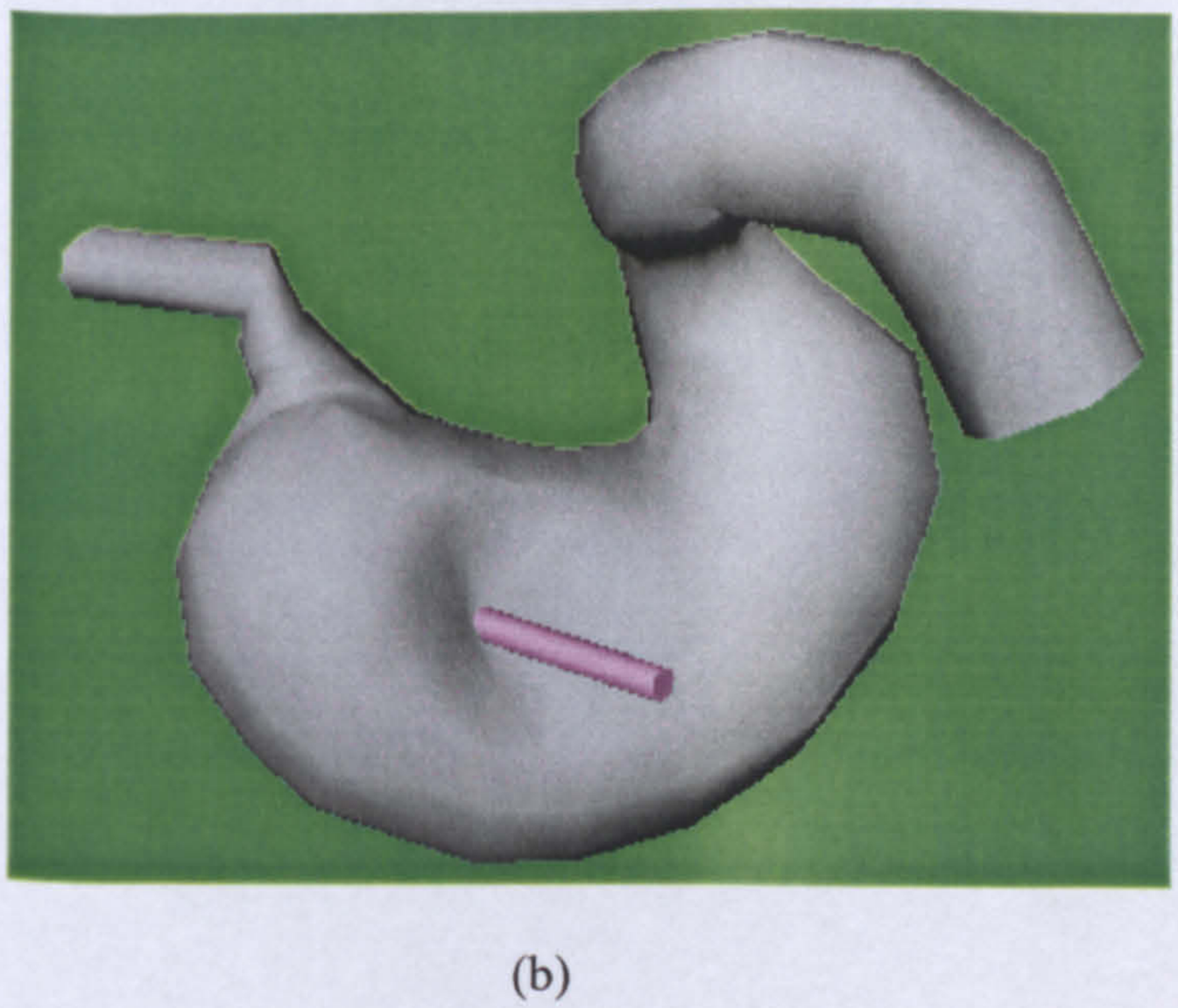
Figure 5. 21 The simulation of a plastic duck using different values of α .

A soft tissue deformation simulation is shown by figure 5.22, where the model of a stomach interacts with a surgical tool. The stomach model (from 3D Café, whose original shape was subdivided to obtain smoother surfaces) consists of 1340 vertices, 4272 triangles and 4808 springs. Figure 5.22 (a) shows the initial shape before the deformation, while figure 5.22 (b) shows the deformation of the stomach model due to the interaction of some of its vertices with a simple tool. This figure clearly shows a uniform deformation, which appears very realistic.

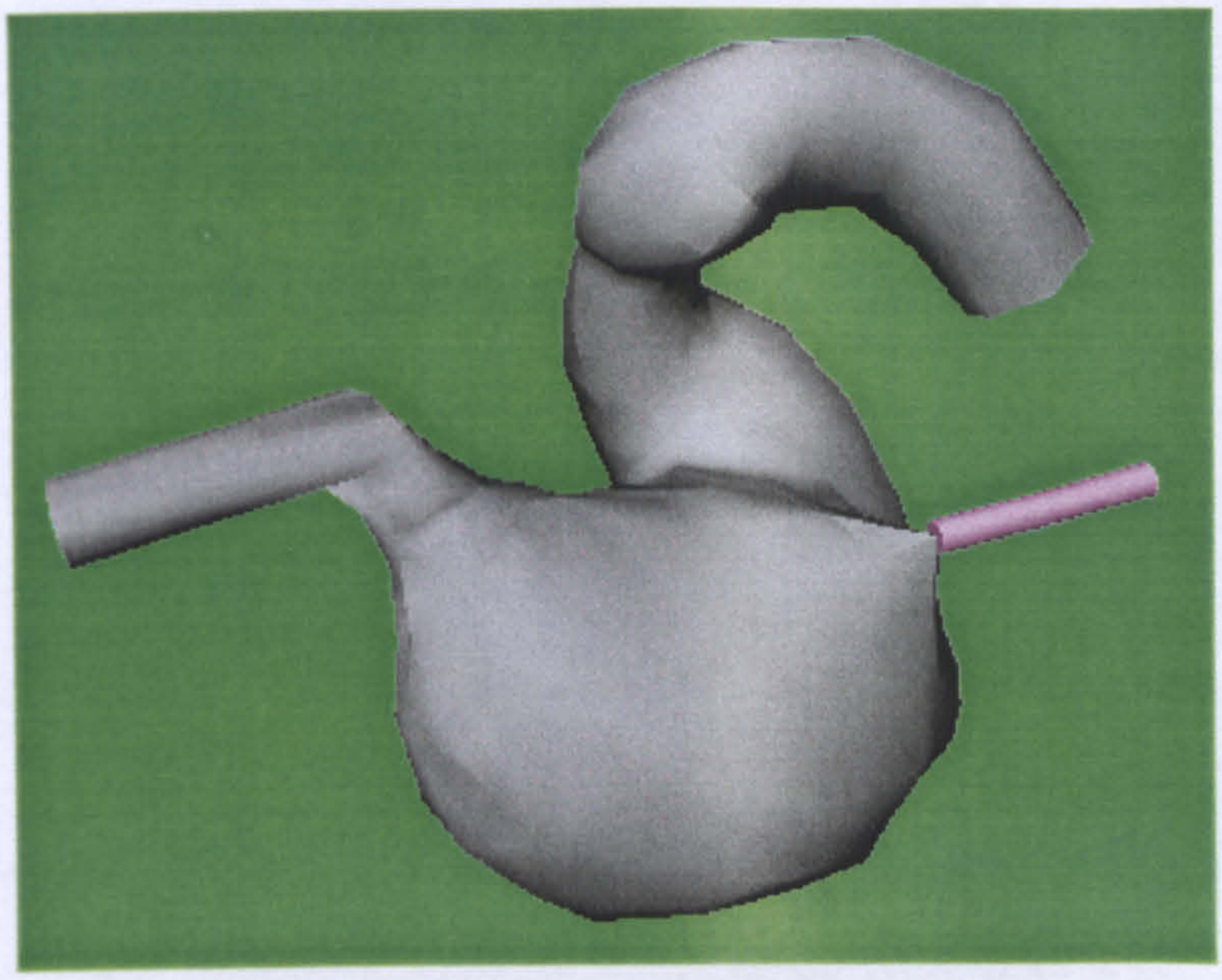
The deformation due to the pulling of some vertices is shown in figure 5.22 (c). In these simulations the stomach model is constrained at both canals at its two ends. The simulation time is still in the range of interactive frame rates (i.e. 0.03 seconds per frame).



(a)



(b)



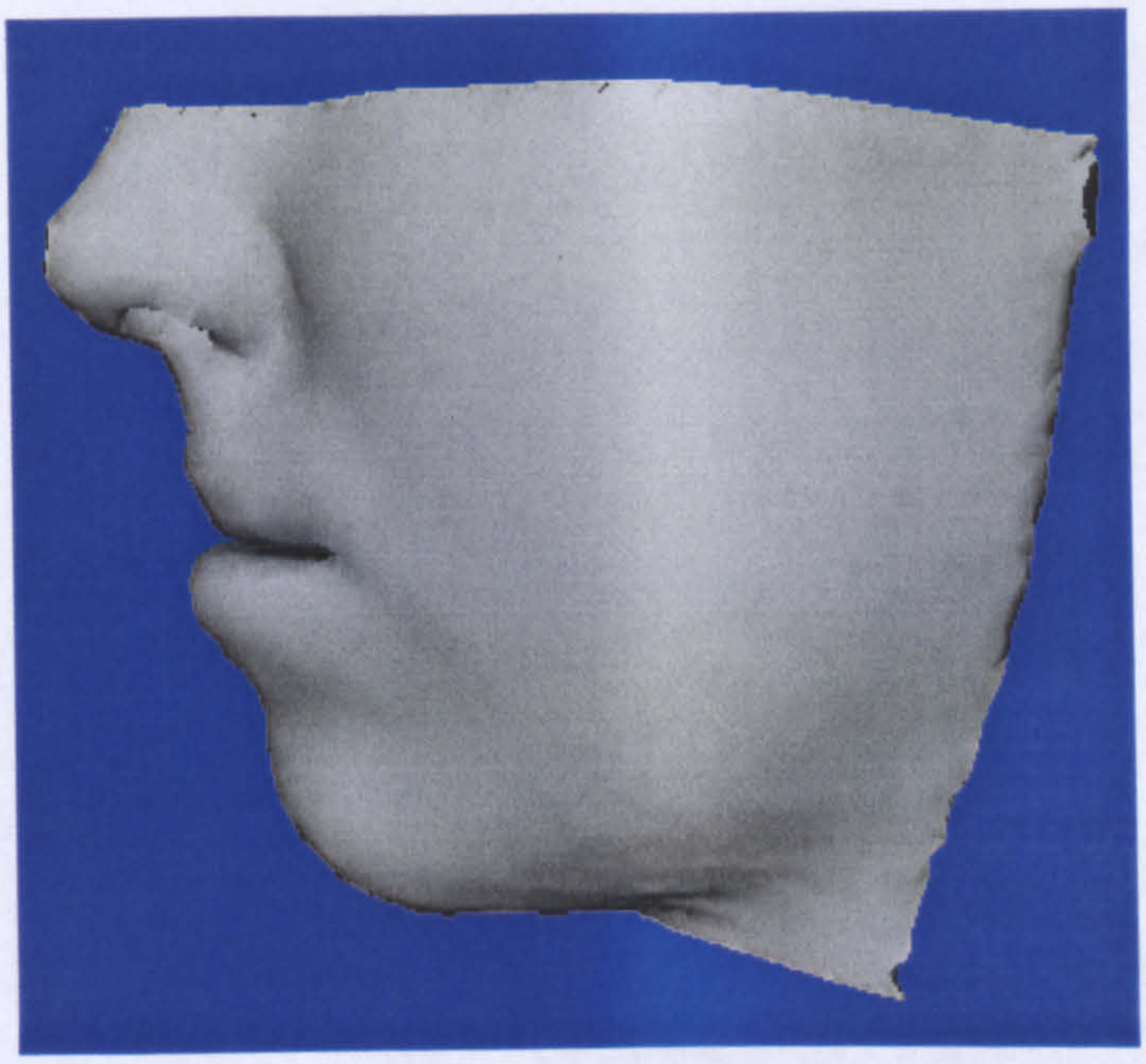
(c)

Figure 5.22 Soft tissue simulation: A stomach model in interaction with a simple tool.

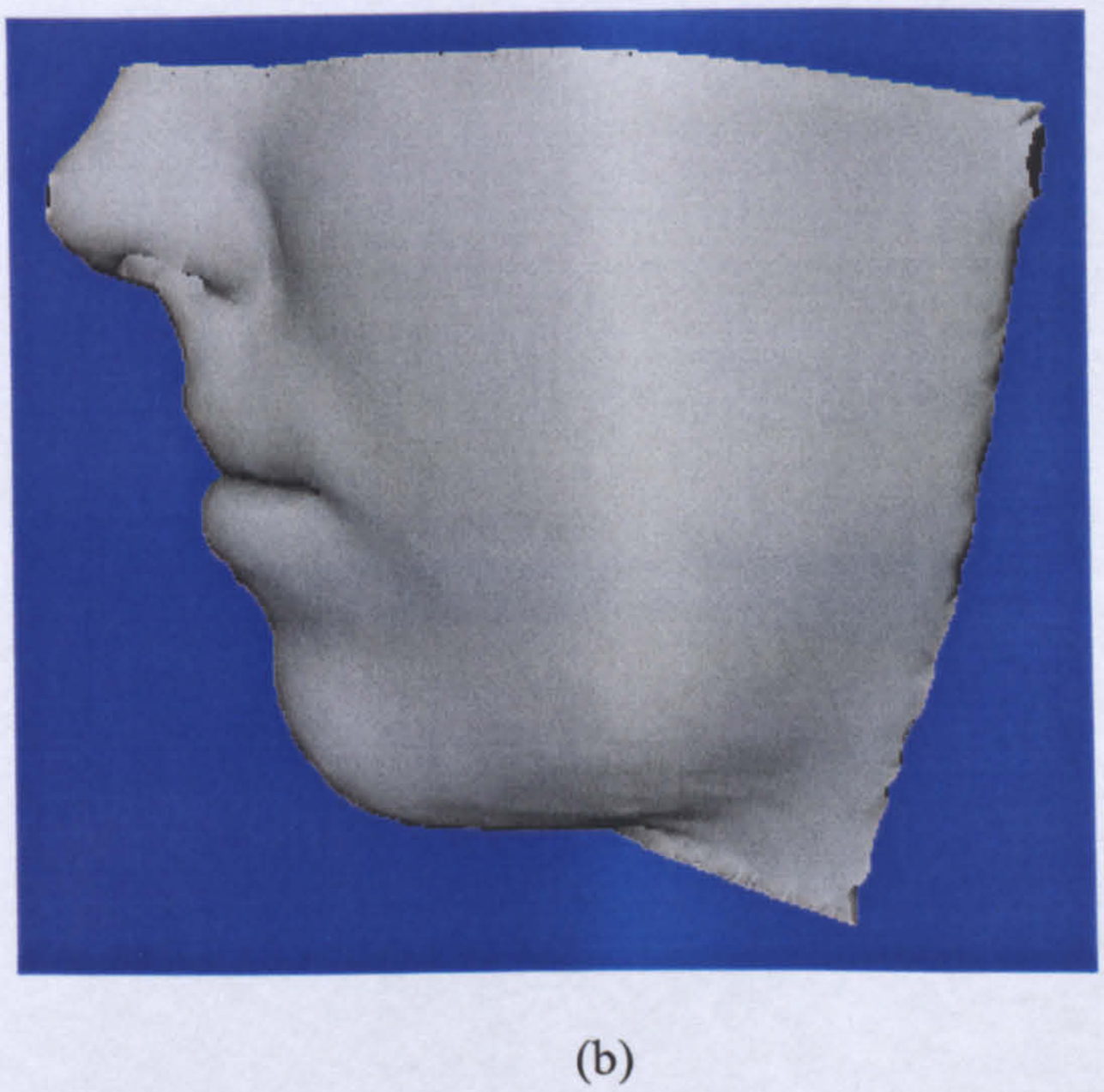
5.4.3 Craniofacial Surgery Simulation

A more sophisticated simulation experiment is realized on the head model developed in chapter 3. Craniofacial surgery (see section 3.4 in chapter 3) is simulated using the new method. Results are very satisfactory and similar to those of the mass-spring systems algorithm. The facial appearance before the surgery is shown in figure 5.23 (a). The bone realignment examined in section 3.4.2 is performed causing facial tissue deformations. Figure 5.23 (b) shows the face after the simulation is performed. Both models are shown superimposed in figure 5.24. These images are compared with the images of figures 3.22 and 3.23 produced by the mass-spring system's algorithm.

We also carried out some facial animation with the new method. The same head model is used in these simulations where the lips are moved to certain locations. These Results are depicted in figure 5.25.



(a)



(b)

Figure 5.23 A craniofacial surgery simulation using mass-spring chain algorithm: (a) initial model and (b) model after the simulation.

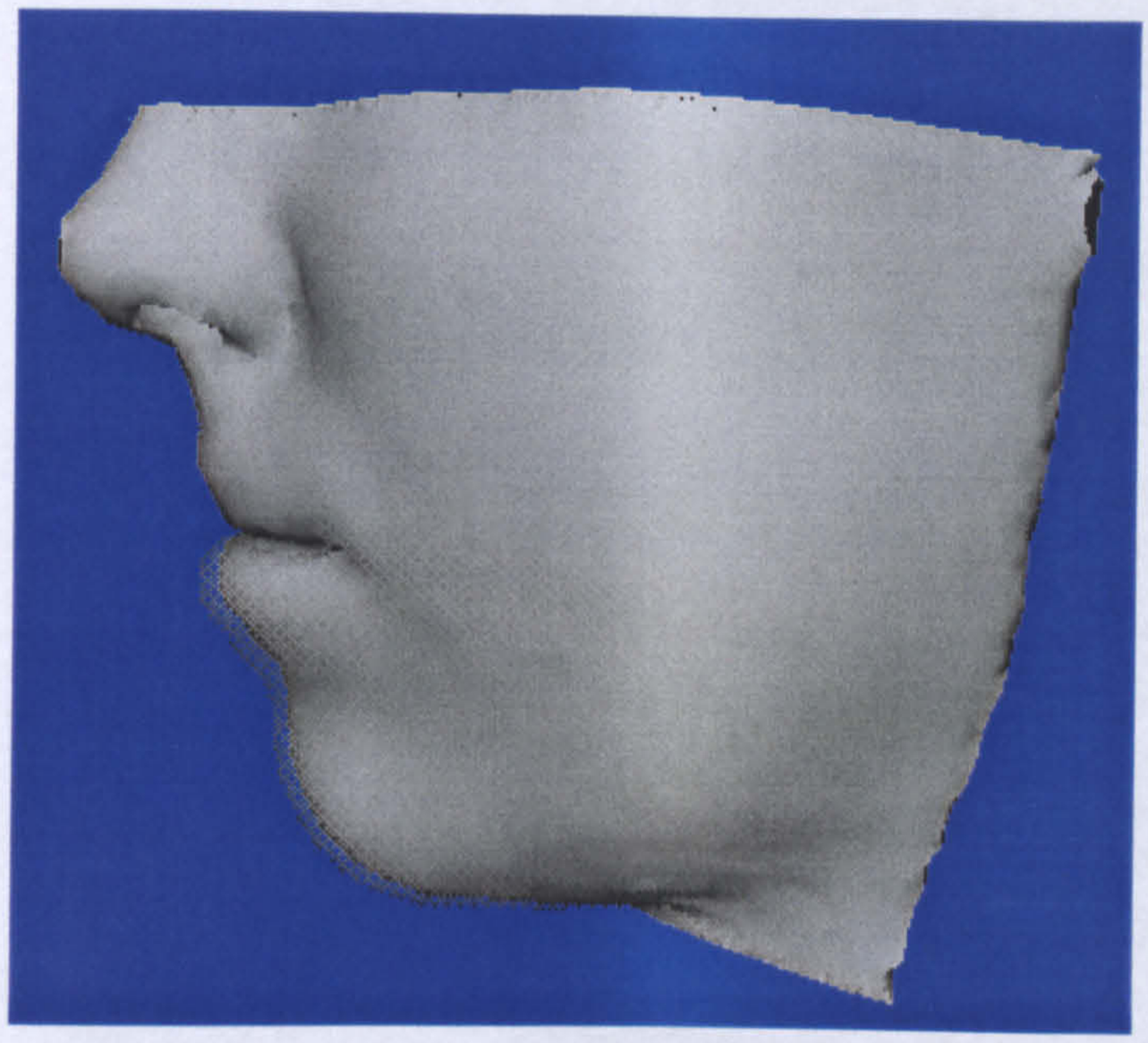
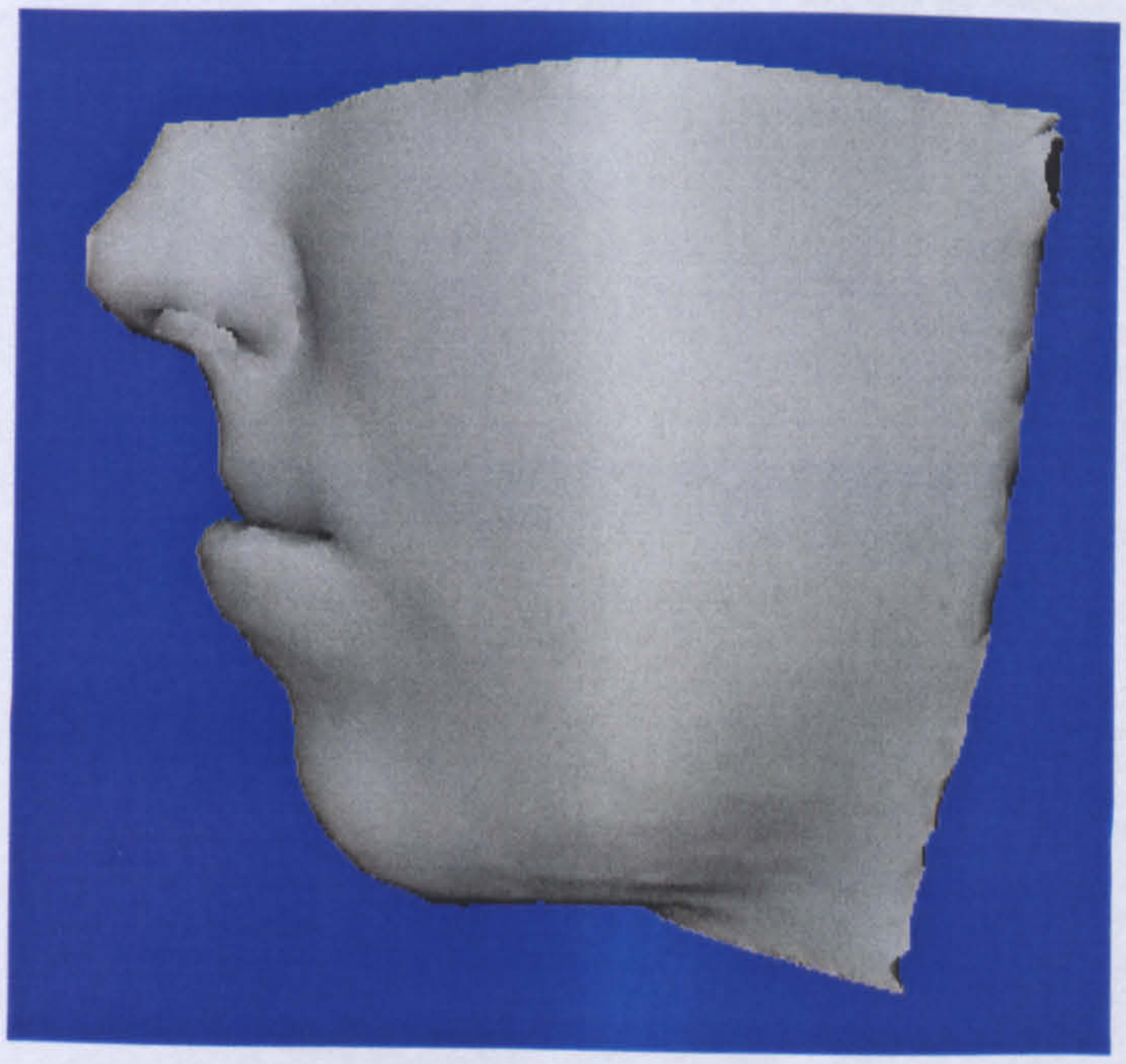
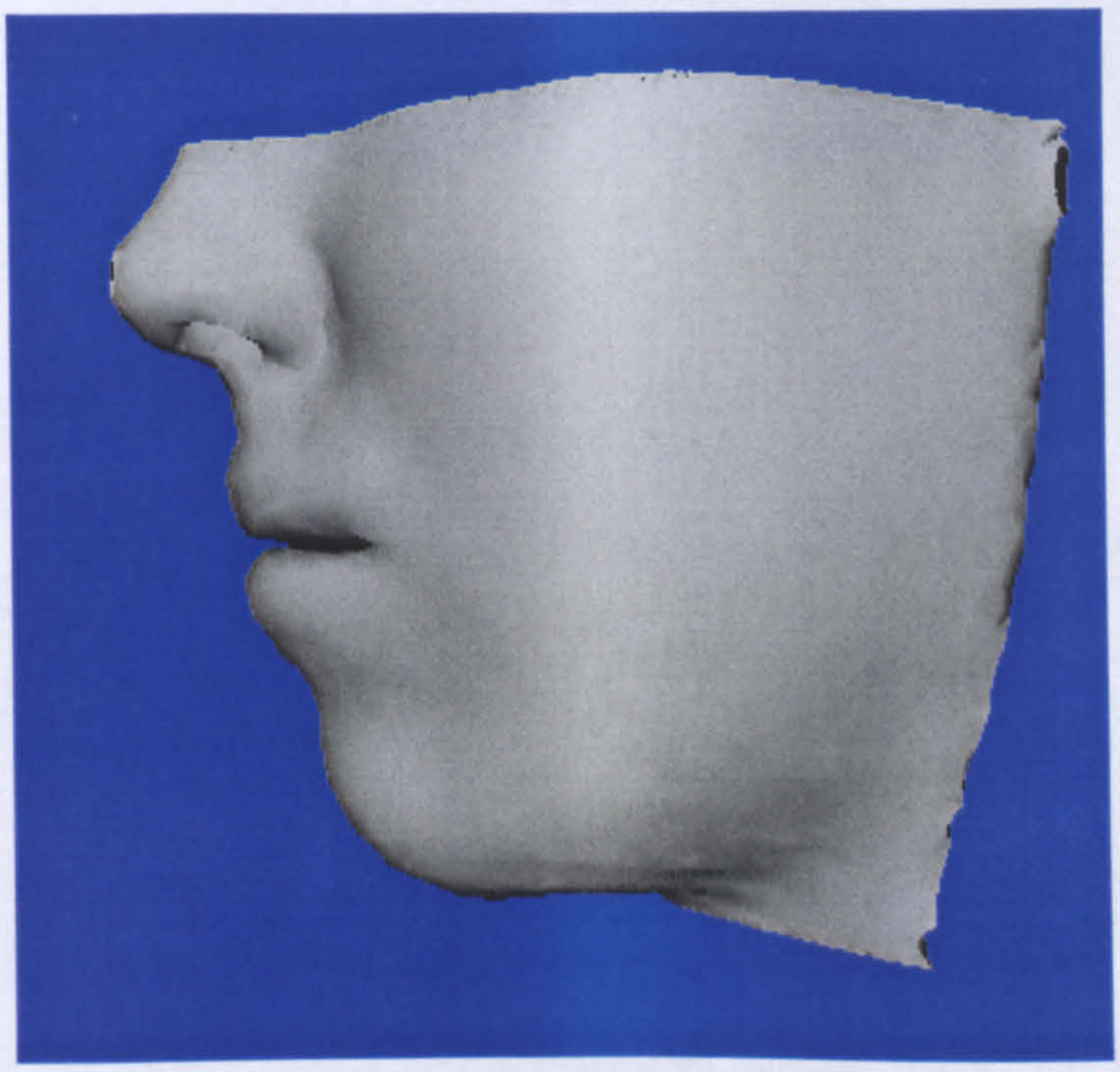


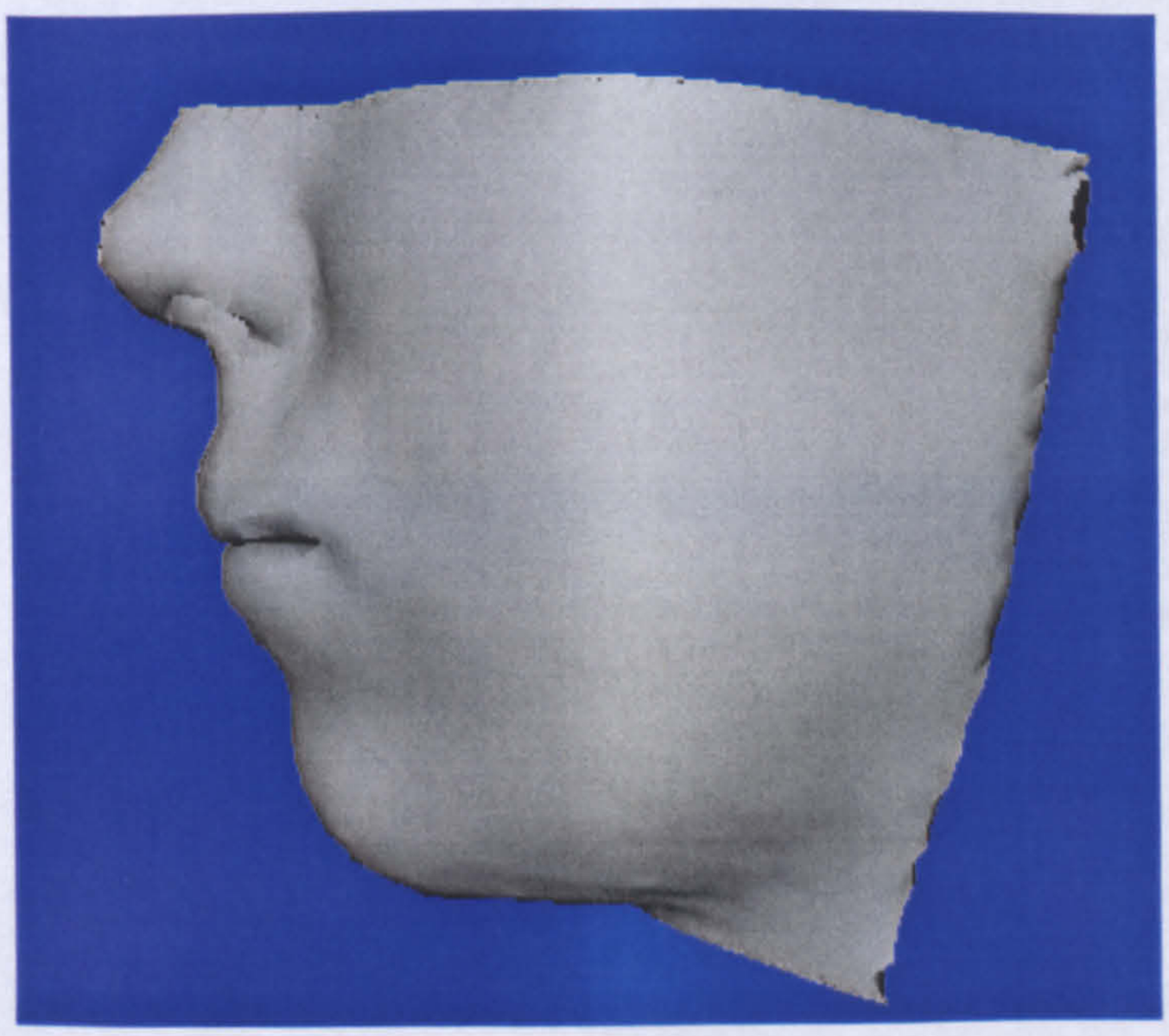
Figure 5.24 Both before and after surgery images.



(a)



(b)



(c)

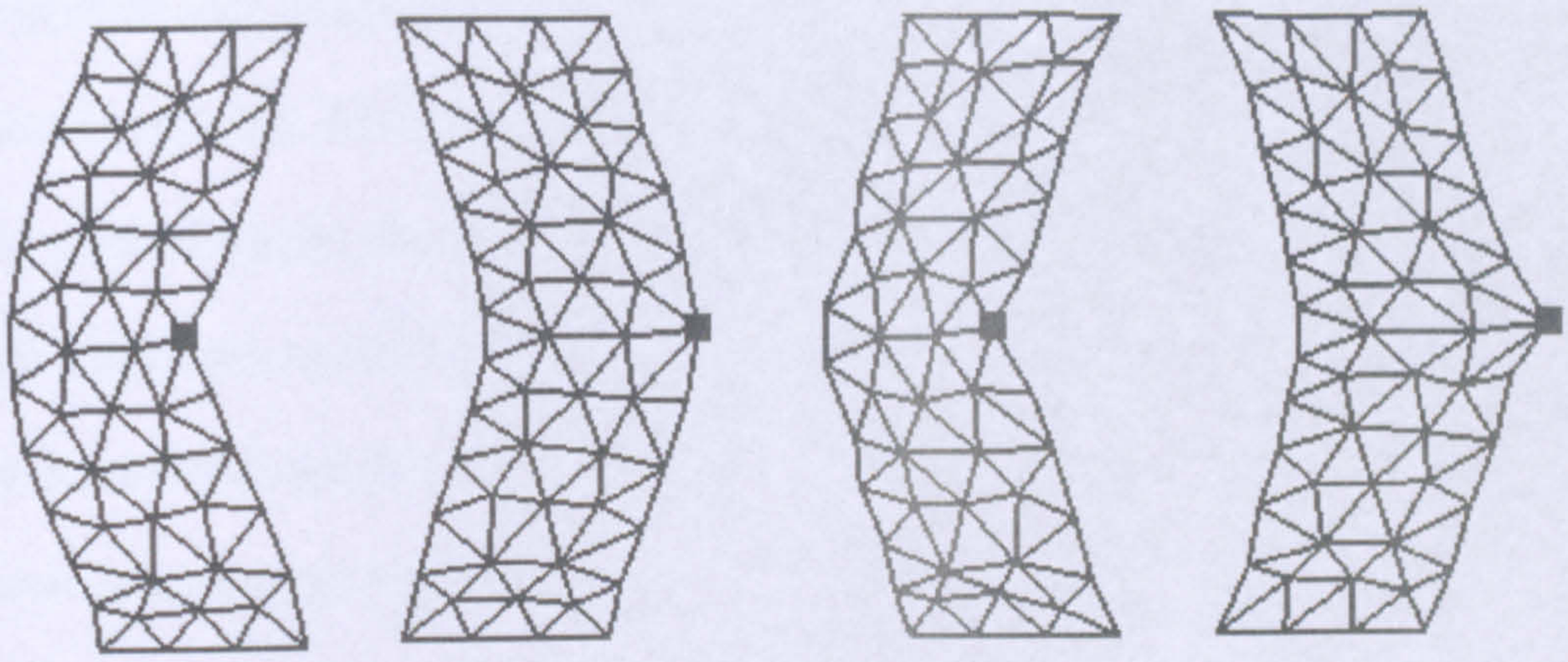
Figure 5.25 Facial animations using the same head model.

5.5 Comparisons

Simulation results from the new method are compared to the results from the mass-spring systems algorithm outlined in chapter 2. Both methods produce acceptable and quite similar results. The results from both methods are shown in figure 5.26, where left-hand side shows the mass-spring system algorithm output, while the right-hand side shows the results of the new method. A virtual force is used in the mass-spring system's simulations except in the craniofacial simulation because the mass-spring system produces better results with external forces instead of vertex movements. If one of the vertices moves inside the figure (passing through other triangles) instability occurs. A simple rectangular 2D model simulation is given in figure 5.26 (a) for the mass-spring system's method and (b) for the new method. The mass-spring method seems to produce smoother results at the outlines. This may be the result of force usage instead of the vertex movement used by the new method.

Experiments show that the results of the new method are greatly effected by the choice of simulation parameters, which can be tuned for better accuracy. When a vertex is pushed inside the object, the new method seems to more accurately simulate the deformation. This is because the area of the object is not effected with the MSS method, while with the new method the object is compressed as expected.

Figures 5.26 (c) and (d) show the deformation simulation of a stomach model. As seen from the figure the results are quite similar. This is also the case in the craniofacial surgery simulation, where both methods work very well as shown in figure 5.26 (e) and (f).



(a) (b)

(c) (d)

(e) (f)

Figure 5.26 Comparison of simulation outcomes between the mass-spring system's algorithm and the new algorithm. The first column shows mass-spring system's results and second column gives mass-spring chain outcome.

A performance comparison (simulation time in seconds) between the two methods is given in table 5.1. For models consisting of around 300 vertices and 600 triangles the new method reaches haptic frame rate, i.e.1000 force samples per second. For models with 1300 vertices and 4500 triangles it works in real time, 30 images per second. For the craniofacial surgery simulation, the new method is about 50 times faster than the mass-spring system's method.

Model	# Vertices	# Triangles	MSS (sec)	New method (sec)
Simple Model	285	566	-	0.001
Bird	2502	5000	-	0.04
Stomach	1340	4272	3.95	0.03
Face	7884	19724 Springs	24.19	0.46

Table 5.1 Simulation times for the mass-spring system versus the new method.

Work	Nodes	Simulation. Time (sec)
Choi, 2002	133	3.136
Sarah, 1999	5000	0.23
Lim, 2004	138	0.016
James, 2001	559	0.0056

Table 5.2 A performance comparison with other published results.

The performance of the new method is also compared against that of other published works. These comparisons are given in table 5.2. As seen from this table our method is one of the fastest algorithms available. Compared to the results presented in Choi et

al. (2002) and Lim et al. (2004), the new method is much faster and compared to the results presented in James and Pai (2001), our method is of similar speed. Sarah's work (1999) using the ChainMail algorithm with approximately 5000 nodes requires 0.23 seconds and our method with approximately 7884 nodes takes about 0.46 seconds.

A craniofacial surgery simulation was done in (Teschner et al. 1999b) where they reached 4.5 second simulation time for a model consisting of 1838 vertices and 11763 springs on a 175MHz workstation. We have achieved 0.46 seconds simulation time for our craniofacial surgery simulation with a model consisting of 7884 vertices and 19724 springs on an 800MHz Pentium 3 PC.

6. Summary

A new deformation algorithm as an alternative to the mass-spring system and ChainMail algorithm was developed based on the working principles of both these methods. The algorithm is fast because it reaches the deformed state in one or two simple steps. It is accurate because the deformation characteristics are controlled by a few parameters, which represent material properties and are easy to tune. In addition, cell conversion leading to unrealistic deformations is prevented. The proposed algorithm is used in the simulation of deformations for a number of different applications, from simple 2D rectangular objects to complex facial tissue simulations. We have compared the performance of the new algorithm to that of the mass-spring system's algorithm. The results were given in table 5.1. Another performance comparison was made in table 5.2, where published results using different methods were presented. The tables show that the new method is much faster than the mass-spring system as well as most other methods. Visual comparison between the results

of the new method and the mass-spring system was also shown in figure 5.26, which reveal that both methods produce quite similar results.

The new method with its structure can be used as a basis for the development of a new deformation algorithm using neural networks. Its geometrical structure and the one or two step approach for the solution permits for such a development. This subject is further discussed in the future work section.

CHAPTER 6

CONCLUSION

Deformable object simulation is very important and one of the most challenging topics in computer graphics. It is challenging because simulation algorithms are required to produce physically accurate results and are required to operate in real time. Both accuracy and performance are in the scope of this thesis. We started out by implementing the mass-spring system's algorithm for the following reasons: As many previous applications indicated that the mass-spring system's algorithm is more suitable for real-time applications and it produces plausible results. This algorithm has been studied in detail covering different integration techniques and collision detection algorithms and has been used to simulate cloth-like objects as well as soft-tissue deformations.

Since our main application area is prediction of soft-tissue deformation, we examined medical data analysis and medical image processing methods. This is necessary for the pre-processing stage involving the acquisition of organ models for the simulation algorithm. At this stage medical data in different formats was read, visualized, interpolated, segmented, filtered and was subjected to some measurements and manipulations, such as cutting and separating. The model obtained through medical image processing tools, however, was not suitable for the simulation algorithm. Our

study therefore also included triangulation and decimation algorithms. A triangulation algorithm produces a polygonal representation of an organ model and the resultant number of triangles are reduced to a reasonable number without significant lost of surface detail for fast rendering.

Our studies along with previous studies showed that mass-spring systems suffer from lack of physical realism mainly because of the choice of simulation parameters. Given the exact parameters representing the deformation behavior of the material mass-spring system can successfully simulate deformations. These parameters are complex and nonlinear, therefore it is difficult to obtain them. In addition, in practice most applications use constant coefficients to represent nonlinear tissue behavior. For these reasons we employed neural networks, which were successfully used in system identification for the unknown system parameters in many engineering applications.

We were able to integrate neural networks into mass-spring systems to improve the accuracy of the simulation algorithm by determining the simulation parameters. The performance of the mass-spring system however was not improved because of the structure of the mass-spring system's algorithm, which relies on an iterative approach. Our search to achieve real-time performance using mass-spring systems and neural networks led to the development of a new method. This method is the product of combining different methods and assumptions. In the following sections we outline our contributions to the field.

6.1 Contributions

Our contributions can be viewed under three different topics. These are: to obtain a head model which can be used in craniofacial surgery simulations, the identification of

simulation parameters using neural networks and finally the development of a new simulation algorithm that provides a suitable structure for neural network applications besides achieving real-time performance.

6.1.1 Model Generation

A head model was generated in order to simulate surgical operations around the head. Some previous works use only surface representations as model, which is clearly not suitable for realistic simulations. This is because internal structures are not taken into account. Other researchers attached surface triangles to the underlying bone structure in order to simulate internal behavior of the face tissues. These types of models do not represent different tissue layers such as fat and muscle tissues. The best model proposed so far traces skin vertices to a center point of the bone structure. Intersections with bone triangles are recorded and an intersection point is assigned by interpolating neighboring intersected points, if tracing fails to find an intersection. This process eventually forms prism elements between the skin surface and the bone surface. The prisms are then divided into different thickness representing different tissue layers.

This model however uses only one center point that fails to represent the human head accurately, which is not a perfect sphere. This method therefore may produce skin vertices attached to the wrong bone parts and may produce inaccurate tissue thickness. In our model, we assigned two center points one for the upper and one for the lower jaws. We also set boundary conditions for skin and bone vertices. This boundary starts form the bottom of the nose and ends at the bottom of the neck. The skin vertices above the lip level are then traced to the upper center point and the remaining skin vertices below the lip level are traced to lower center point. Intersections with bone triangles are then recorded. During this first phase we also determined an average tissue thickness

that is used to assign prism elements that have not found any intersections in the first run. This algorithm ensures the appropriate formation of prism elements between the two surfaces with best approximation of tissue thickness. The model obtained by this technique is then used to simulate soft-tissue deformations caused by surgical operations around the head.

6.1.2 Neural Network System Identification

The biggest challenge in simulations using mass-spring systems is the determination of system parameters. It is crucially important to use the correct coefficients, which dramatically effect the accuracy of the simulation outcome as well as the stability of the simulation. In previous works, some researchers targeted only one parameter and some tried to identify all parameters using pre-set conditions. The best approach proposed so far is to use neural networks, which have already proved to be a powerful tool for system identification. A method used in the literature with six neural networks however did not converge to the original system parameters but instead learnt a combination of internal forces for those specific conditions and applications.

We proposed a new identification method based on the mass-spring structure and specifically for mass-spring systems. This method requires only two neural networks to be trained to learn nonlinear spring stiffness and spring damping. We have also developed a new learning algorithm that involves an adaptive learning rate. In conventional learning algorithms the error signal is used to update the weights of the neural networks. In mass-spring systems the error between the desired output and the neural network output is formed by contributions from several springs. Using the very same error to modify the weights of networks for each spring, which has a different amount of contribution will clearly result in inappropriate learning. Based on spring

theory, we developed an adaptive learning rate, which makes sure that weights are updated according to the contributions from each spring. This learning algorithm also prevents identifying negative values for coefficients by constraining the learning algorithm. Negative learning occurs at the early stages of the learning process and in situations where not enough data is presented. Our algorithm is shown to be very fast to converge and very accurate in learning nonlinear system parameters. Given a set of training data, which may be derived, either from a mathematical model or from experimentation, our model is able to extract system parameters accurately, leading to better simulation realism. Our method was tested using the data generated by the mass-spring systems simulation. Unknown nonlinear parameters were successfully identified.

6.1.3 The New Deformation Algorithm

The most significant contribution of this thesis is the development of a new deformation algorithm. We have studied various deformation algorithms specifically the mass-spring system and ChainMail algorithms because the former produces plausible results in reasonably fast time and the latter is extremely fast making it suitable for real-time interactive applications. As in other deformation techniques, these two algorithms define the relationships between vertices of the building blocks of the model to be simulated. The deformation is then a means of updating these vertices (finding their new locations) based on defined relationships. The problem can be summarized as defining the relationships between these vertices and finding the new positions of vertices.

Inspired by the mass-spring system and the ChainMail algorithms, we defined a formula representing spring length changes and set limits forming a search space where the deformed vertex positions lie. These limits narrow down the solution space making the

algorithm work faster. The limits are based on the minimum and maximum allowed vertex movements and spring elongation. These limits are controlled by some parameters, which relate to the properties of material being simulated. Once the search space is reduced, we assign a position vector (orientation vector) whose purpose is to determine the new locations of the vertices and springs. The orientation vector itself is controlled by a parameter representing the deformation characteristics of the object being deformed. In the next step of the algorithm springs are deformed according to a defined formula to obtain their final deformed positions. Deforming the spring length also includes some parameters. These parameters give great flexibility to represent various deformation characteristics and provide user control over the accuracy of the deformation.

This algorithm does not use an iterative approach, it visits each spring once or twice depending on the implementation and is therefore very fast. Since the deformation search space and the deformation itself is controlled by user defined parameters the accuracy of the algorithm is very high. Its physical accuracy can further be improved by tuning these parameters. We have implemented the new algorithm to simulate various models. Our results show that this algorithm can achieve reasonable accuracy and that the simulation time is reduced by about an order of 100 compared to that of the mass-spring systems algorithms.

6.2 Future Work

The work on improving the accuracy and speed of mass-spring systems led us to use neural networks for better physical realism by approximating system parameters. Our research also led to the development of a new deformation algorithm, which is very fast and suitable for real-time interactive applications. Future work will continue to focus on

the above-mentioned subjects; Mass-Spring Systems, MassSpringChain and Neural Networks.

The work specifically will be focused on improving the new deformation algorithm. We will focus on the determination of the exact locations of mass-points within the established limits after the movement. Instead of using a single vector to find the deformed locations, we will use energy minimization or optimization methods. Since the limits are already set to decrease the search space, these methods will still work very fast.

Second, emphasis will be placed on the identification of the parameters of the new deformation law, such as a nonlinear parameter representing the material characteristics of the objects and a nonlinear parameter controlling the stretch and compression characteristics of materials. A neural network identification algorithm will be developed for this specific case and these parameters will be determined. A powerful aspect of the new algorithm is that it allows real data to be used in the identification process. Since mass-spring systems represent an iterative approach it is extremely difficult to obtain real data at each time step of the iteration algorithm. With our method, however, before and after surgery data will be enough for the identification process. This was one of the main targets in developing such a method. A series of before and after surgery data (easy to obtain) can be presented to the neural network model. The neural network model at the initial state is fitted to the "before surgery data". The network is then deformed and its parameters are adjusted until it matches the "after surgery data".

Ultimately we will modify the new method as a neural network structure and only use neural networks to simulate soft-tissue deformations. This is likely to produce very

realistic simulations since neural networks will learn from real data and will also work in real-time interactively since the algorithm's structure is designed to be very fast.

REFERENCES

1. Albus, J. S., 1972. Theoretical and experimental aspects of a cerebellar model, *Ph.D.* dissertation, University of Maryland.

2. Albus, J. S., 1975a. A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC), *Trans. ASME J. Dynamic System. Meas. Contr,* Volume 97, No. 3, pp 220-227.

3. Albus, J. S., 1975b. Data Storage in the Cerebellar Model Articulation Controller. (CMAC). *Trans. ASME J. Dynamic System. Measurement. Control,* Volume 97, no. 3, pp 228-233.

4. Albus, J. S., 1979. Mechanism of Planning and Problem Solving in the Brain, *Math. Biosci,* Volume 45, pp 247-293.

5. Algorithms 2004: http://www.gamespp.com/algorithms/collisiondetection.

6. Almeida, J. S., Voit, E. O., 2003. Neural-Network-Based Parameter Estimation in S-System Models of Biological Networks, *Genome Informatic*, 14: pp. 114-123.

7. Amenta, N., Bern, M., Kamvysselis, M., 1998. A new Voronoi-Based Surface Reconstruction Algorithm, *International Conference on Computer Graphics and Interactive Techniques,* pp. 415-521.

8. Amira. http://amira.zib.de/papers/vertical/medical.html

9. Analyze. http://www.mayo.edu/bir/Software/Analyze/Analyze1.html

10. Ananthraman, S., Gargk, D.P., 1993. Training, Backpropagation and CMAC Neural Networks for Control of a SCARA Robot *Artif. Intell.*, Vol. 6, No. 2, pp. 105-115.

11. Arun, K. S., Huang, T. S., Blostein, S. T., 1987. Least-squares Fitting of two 3-D Point Sets, *IEEE Trans Pattern Anal. Machine Intell.*, 9(5), pp. 698-700.

12. Atkins, K. S., Mackiewich, B. T., 1998. Fully Automatic segmentation of the brain in MRI, *IEEE Trans. Med. Imag.* 17 (1), pp. 98-107.

13. Badouel, D., 1990. An Efficient Ray-Polygon Intersection. *Graphics Gems*.

14. Baraff, D., Witkin, A., 1998. Large Steps in Cloth Simulation. *SIGGRAPH,* pp 43-54.

15. Baraff, D., Witkin, A., 1999. Physically based modeling course notes, *Course 36, SIGGRAPH'99.*

16. Baraff, D., Witkin, A., 2001. Physically based modeling course notes http://www.pixar.com/companyinfo/research/pbm2001.

17. Bar-Cohen, Y., Breazeal C. L., 2003. Biologically-Inspired Intelligent Robots, *SPIE Press Mongraph*, Vol. 122.

18. Bhat, S.K., Twigg, C.D., Hodgins J.K., Khosla P.K., Popovic, Z., Seitz, S.M., 2003. Estimation Clot Simulation Parameters from Video. *Eurographics/SIGGRAPH Symposium on Computer Animation,* pp. 37-51.

19. Boissonnat, J. D., 1984. Geometric Structures for Three-Dimensional Shape Representation, *ACM Trans. Graph.*, 3(4), pp. 266-286.

20. Bourguigron, D., Cani, M., 2000. Controlling Anisotropy in mass-spring systems. *Computer animation and simulation '00. pp. 113-123.*

21. Bouzas, M., Arnold, D., 1998. Experiments in animation control by neural networks. *Information Visualization, Proceedings IEEE Conference on,* pp. 252-260.

22. Brown, J., Sorkin, S., Bruyns, C., Latombe, J., Montgomery, K., Stephanides, M., 2001. Real-time simulations of deformable objects; tools and application. *Computer Animation, Stanford University.*

23. Brouwer, I., Ustin, J., Bentley, L., Sherman, A., Dhruv, N., Tendick, F., 2001. Measuring in Viva animal soft tissue properties for haptic modeling in surgical simulations. *Medicine Meets Virtual Reality,* pp. 69-74.

24. Bro-Nielsen, M., Cotin., 1996. Real-Time Volumetric Deformable Models for Surgery Simulation Using Finite Element and Condensation, *Computer Graphics Forum,* 15(3): pp. 57-66.

25. Bro-Nielsen, M., 1998. Finite Element Modeling in Surgery Simulation. *Proceedings of the IEEE,* Volume 86, No. 3, pp. 490-503.

26. Bro-Nielsen, M., Helfrick, D., Glass, B., Zeng X., Connacher, H., 1998. VR Simulation of Abdominal Trauma Surgery, *Medicine Meets Virtual Reality 6 (MMVR-6)*, pp. 117-123.

27. Cakmak, H. K., Kuhnophel U., 2000. Animation and Simulation Technique for VR-training Systems in Endoscopic Surgery, *Europraphics Workshop on Animation and Simulation,* pp. 173-185.

28. CGL: http://graphics.ethz.ch/.

29. Chen, Y., Zhu, Q., Kaufman, A., 1998. Physically-Based Animation of Volumetric Objects. *Computer Animation, Proceedings,* pp. 154-160.

30. Chen, F. C., Chang, C. H., 1994. Practical Stability Issues in CMAC Neural Network Control Systems, *Proceedings of the Control Conference Baltimore, Maryland*, pp. 29-45.

31. Choi, Y. J., Hong, M., Choi, M. H., Kim, M. H., 2002. Adaptive Surface Deformation Model with Shape-preserving Spring. *Proceedings of International Conference on Virtual systems and Multimedia,* pp. 1-39.

32. Christiansen H. N., Sederberg, T. W., 1978. Conversion of Complex Contour Line Definitions into Polygonal Elements Mosaics, *International Conference on Computer Graphics and Interactive Techniques,* pp. 187-192.

33. Ciampalini, A., Cignoni, P., Montani, C., Scopigno, R., 1997. Multiresolution Decimation based on Global Error. *The Visual Computer*, 13(5), pp. 228-246.

34. Cohen, I., Cohen, L. D., Ayache, N., 1992. Using Deformable Surfaces to Segment 3-D Images and Infer Differential Structures, *Proceedings of the Second European Conference on Computer Vision,* pp. 648-653.

35. Collins, D. L., Holmes, C. J., Peters, T. M., Evans, A. C., 1996. Automatic 3-D Model Based Neuroanatomical Segmentation, *Human Brain Mapping*, 4, pp. 190-208.

36. Cotin, S., Delingette, H., Ayeche, N., 1999. Real-time Elastic Deformations of Soft Tissue for Surgery Simulation. *IEEE Transactions on Visualization and Computer Graphics,* Volume 5, No 1, pp. 62-73.

37. Cotin, S., Delingette, H., Ayache, N., 2000. A Hybrid Elastic Model Allowing Rea-Time Cutting, Deformations and Force-Feedback for Surgery Training and Simulation. *The Visual Computer*, 16(8): pp. 437-452.

38. d'Aulignac, D., Laugier, C., Cavusoglu, M.C., 1999. Towards a realistic echographic simulator with force feedback. *Proceedings of the 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems,* Vol.2, pp. 727-732.

39. Debunne, G., Desbrun, M., Cani M. P., Barr A. H., 2000. Adaptive Simulation of Soft Bodies in Real Time in Computer Animation, *Proceedings of the 2000 Conference on Computer Animation*, pp.15.

40. Debunne, D., Desbrun, M., Barr, A. H., 2001a. Interactive Animation of Cloth-Like Objects in Virtual Reality. *Journal of Visualization and Computer Animation,* 12(1): pp. 1-12.

41. Debunne, G., Desbrun, M., Cani M. P., Barr A. H., 2001b. Dynamic Real Time Deformations Using Space & Time Adaptive Sampling. *Computer Graphics Proceedings, SIGGRAPH*, pp. 31-36.

42. Delingette, H., Cotin, S., Ayeche, N., 1999. A Hybrid Elastic Model Allowing Real-time Cutting, Deformations and Force-feedback for Surgery Simulation. *Computer Animation, Proceedings,* pp. 70-81.

43. DICOM: http://www.dclunie.com.

44. DiMaio, S. M., 2003. Modeling, Simulation and planning of Needle Motion in Soft tissues, Ph.D. Thesis, *University of British Colombia*, September 2003.

45. Desbrun, M., Schroder, P., Barr, A., 1999. Interactive animation of structured deformable objects. *In Proceedings of Graphics Interface (GI 1999),* pp. 1-8.

46. Desbrun M, Meyer M., Barr, A.H., 2000. Interactive animation of Cloth-Like Objects for Virtual Reality, *Cloth Modeling and Animation*, pp. 219-239.

47. Duchille ix F., Qin, H., Kaufman, A., El-sana, J., 1999. Haptic Sculpting of Dynamic Surfaces. *Symposium on Interactive 3D Graphics,* pp. 103-110.

48. Duysak, A., 1997. System identification and model reference control using CMAC neural networks. *The Pennsylvania State University, Master Thesis.*

49. Duysak A, Zhang J. J., Ilankovan V., 2003, Efficient Modeling and Simulation of Soft Tissue Deformation Using Mass-Spring Systems, *The 17th International Congress and Exhibition on Computer Assisted Radiology and Surgery (CARS 2003).*

50. Duysak A, Zhang J. J., 2003, Identification of Simulation Parameters Using Neural Networks, *The 6th International Conference on Computer Graphics and Artificial Intelligence (3IA 2003),* pp. 337-342.

51. Duysak A, Zhang J. J., 2004, Fast Simulation of Deformable Objects, *International symposium on Computer Animation, The 8th International Conference on Information Visualization, IEEE Computer Society, (IV 2004),* pp.422-427

52. Edelsbrunner, H., Mucke, E. P., 1994. Three-Dimensional Alpha Shapes, *ACM Trans. on Graphics,* 13(1), pp. 43-72.

53. Eischen, J., Bigliani, R., 2000. Continuim Versus Particle Representations. *Cloth Modeling and Animation,* pp. 79- 122.

54. Ganovelli, F., Gignoni P., Scopigno, R., 1999. Introducing Multiresolution Representation in Deformable Object Modeling, *SCCG, Conference Proceedings*, pp. 149-158.

55. Giacomo, T. D., Magnenat-Thalmann N., 2003. Bi-Layered Mass-Spring Model for Fast Deformations of Flexible Linear Bodies. *16TH International Conference of Computer Animation and Social Agents (CASA),* pp. 48-54.

56. Gibson, S.F.F., 1997 (a). 3D Chainmail: A Fast Algorithm for Deforming Volumetric Objects. *Proc. Symp. Interactive 3D Graphics,* pp. 149-154.

57. Gibson, S. F. F., Mirtich, B., 1997 (b). A Survey of Deformable Modeling in Computer Graphics. MERL Technical Report TR97-19, http://www.merl.com.

58. Gibson, S. F. F., 1999. Using Linked Volumes to Model Object Collisions, Deformation, Cutting, Carving, and Joining. *IEEE Transactions on Visualization and Computer Graphics,* Volume 5, No 4, pp. 333-348.

59. Grzeszczuk, R., Terzopoulos, D., Hinton, G., 1998. NeuroAnimator: fast neural network emulation and control of physic-based models. *SIGGRAPH'98, Conference Proceeding,* pp. 9-20.

60. Haptica: http://haptica.com.

61. Hault, M., Strasser, W., 2004. Corotational Simulation of Deformable Solids, *In Proc. WSCG,* pp. 133-145.

62. Hoppe, H., 1996. Progressive Meshes. *Proc. SIGGRAPH,* pp. 99-108.

63. Hoppe, H., DeRose, T., Duchamp T., McDonald J., Stuetzle, W., 1993. Mesh Optimization, *Computer Graphics Proceedings, SIGGRAPH,* pp. 19-26.

64. Horvath, G., Dunay, R., Pataki, B., 1996. Recurrent CMAC: a Powerful Neural Network for System Identification, *IEEE Instrumentation and Measurement Technology Conference,* Vol. 2, pp. 992-997.

65. Howlett, P., Hewitt, W.T., 1998. Mass-spring Simulation Using Adaptive Non-Active Points. *EUROGRAPHICS,* Volume 17, No 3, pp. C346-353.

66. Hutchinson, D., Preston, M., Hewitt, T., 1996. Adaptive Refinement for Mass/Spring Simulations, *Eurographics Workshop on Computer Animation and Simulation,* pp. 31-45.

67. IMAGE: http://www.comp.leeds.ac.uk/comir/resources/links_c.html.

68. INRIA: http://www.inria.fr.

69. Ishikawa, T., Sera, H., Marishima, S., Terzopoulos, D., 1998. Facial image reconstruction by estimated muscle parameters. *Automatic Face and Gesture Recognition, Proc. Third IEEE International Conference on*, pp. 342-347.

70. Jain, A. K., Mao, J., 1996. Artificial Neural networks: A Tutorial, *IEEE on Neural Networks,* pp. 31-44.

71. James, D. L., Pai, D. K., 2001. A Unified Treatment of Elastostatic Contact Simulation for Real Time Haptics. *Haptics-e* Vol. 2, Number 1, pp. 1-13.

72. Joukhadar, A., Laugier, G.C.H., 1997. Constraint-based identification of a dynamic model. *I International Conference on Robots and Systems, IROS'97,* pp. 3373-42.

73. Kang, Y.M., Choi, J.H., Cho, H.G., 2000 (a). Fast and stable animation of cloth with an approximated implicit method. *Proceedings of the Computer Graphics International (CGI'00),* pp. 247-255.

74. Kang, Y.M., Choi, J.H., Cho, H.G., Lee D. H., Park C. J., 2000 (b). Real Time animation technique for Flexible and Thin Objects, *In proceedings of the Winter School of Computer Graphics (WSCG 2000),* pp. 322-329.

75. Kang, Y.M., Choi, J.H., Cho, H.G., Lee D. H., 2001. An Efficient Animation of Wrinkled Cloth with Approximated Implicit Integration, *The Visual Computer, 17(3),* pp. 147-157.

76. Kawamato, I., Liquni, Y., Adachi, N., 1995. Design of a Cerebellar Model Arithmetic Computer with Adaptive Resolution and its Application to Nonlinear Signal Processing, *Electronics and Communications in Japan*, part 3, vol. 78, no. 9, pp. 31-38.

77. Keeve, K., Girod, S., Pfeifle, P., Girod, B., 1996 a. Anatomy-based Facial Tissue Modeling Using the Finite Element Method. *Visualization, Proceedings,* pp. 21-28.

78. Keeve, E., Girod, S., Girod, B., 1996 b. Computer-aided craniofacial surgery. *Proc. Of Computer Assisted Radiology CAR'96,* pp. 757-763.

79. Keeve, E., Girod, S., Kikins, R., Girod, B., 1998. Deformable modeling of facial tissue for craniofacial surgery simulation. *Invited Paper, Computer aided surgery,* pp. 1-10.

80. Keeve, E., Kikinis, R., 1999. Deformable Modeling of Facial Tissue. *Proceedings of the First Joint BMES/EMBS Conference,* Vol. 1, pp. 502.

81. Kerdok, A.E., Cotin, S.M., Ottensmeyer, M.P., Galea, A., Howe, R. D., Dawson, S.L., 2003. Truth Cube: Establishing Physical Standards for Real Time Soft Tissue Simulation. *Medical Image Analysis*, pp. 283-291.

82. Kim, H., Lin, C., 1992. Use of Adaptive Resolution for Better CMAC Learning, *IJCNN92*, pp. 517-522.

83. KISMET 3D simulation software: http://iregt1.iai.fzk.de.

84. Koch, R. M., Roth, S.H.M., Grass, M.H., Zimmermann, A. P., Soiler, H. F., 2002. A framework for facial surgery simulation.. *Proceedings of ACM SCCG* http://graphics.ethz.ch/main.php?Menu=5&Submenu=1.

85. Koch R. M., Gross, M. H. Bosshard A. A., 1998. Emotion Editing Using Finite Elements, *Proceedings of the Eurographics, Computer Graphics Forum*, Vol. 17, N0. 3, C295-C302.

86. Koch, R.M., Gross, M.H., Buren, D.F., Fankhauser, G., Parish, Y.I.H., Carls, F.R., 1996. Simulating facial surgery using finite element models. *ACM Computer Graphics SIGGRAPH,* pp.421-428.

87. Kraft, L.G., Campagna, D.P., 1990. A Comparison Between CMAC Neural Network Control and Two Traditional Adaptive Control Systems, *IEEE Control Systems Magazine*, Vol. 10, pp. 36-43.

88. Kraft, L.G., Ho, S., 1991. Stability Properties of CMAC Neural Networks, *IEEE Control Magazine*, pp. 31-33.

89. Kuhnopfel, U., Cakmak, H.K., Maab H., 2000. Endoscapic surgery training using virtual reality and deformable tissue simulation. *Computers&Graphics 24 (2000),* pp. 621-632.

90. Kuhnopfel, U.,Cakmak, H. K., Maab, H., 1999. 3D Modeling for Endoscopic Surgery, *Proc. IEEE Symposium on Simulation*, pp. 22-32.

91. Lafluer, B., Magnenat-thalmann, N., Thalmann, D., 1991. Cloth Animation with self-collision Detection. *Modeling in Computer Graphics,* pp. 179-187.

92. Lee, Y., Terzopoulos, D., Waters, K., 1995. Realistic modeling for facial animation. *ACM Computer Graphics,* Vol. 29, pp. 55-62, Aug. 6-11.

93. Lim, K. M., Wang, F., Poston, T., Zhang, L., Teo, C. L., Burdet, E., 2004. Multi-Scale Simulation for Microsurgery Trainer. *IEEE International Conference on Robotics and Automation.*

94. Lin, Y., Song, S., 1992. A CMAC Neural-Network-Based Algorithm for the Kinematic Control of a Walking Machine. *Engng Applic. Artif. Intell.*, vol. 5, no. 6, pp. 539-551.

95. Lorensen, W. E., Cline, H.E., 1987. A High Resolution 3D Surface Construction Algorithm. *ACM Computer Graphics,* Volume 21, No 24, pp. 163-169.

96. Louchet, L., Provot, X., Crochemore, D., 1995. Evolutionary identification of cloth animation models. *In proceedings of the 6th Eurographics Workshop on Animation and Simulation,* pp. 44-54.

97. Lu, W., Keyhani, A., Fardoun, A., 2003. Neural Network Based Modeling and Parameter Identification of Switched Reluctance Motors, *IEEE Transactions on Energy Conversion,* Vol. 18, No. 2, pp. 284-290.

98. Maes, F., Collignon, A., Vandermeulen, D., Marchal, G., Suetens, P., 1997. Multimodality Image Registration by Maximization of Mutual Information, *IEEE Trans. Med. Imag.*, 16(2), pp. 187-198.

99. Maintz, J. B. A., Viergever, M. A., 1998. A Survey of Medical Image Registration. *Med. Image Anal.*, 2(1), pp. 1-36.

100. MarchingCubes: http://www.essi.fr/~lingrand/MarchingCubes/accueil.html.

101. Maurer, C. R., Aboutanos, B., Dawant, B. M., Maciunas, R. J., Fitzparick, J. M., 1996. Registration of 3-D Images using Weighted Geometrical Features, *IEEE Trans. Med. Imag.*, 15(6), pp. 836-849.

102. Miller, W. T., 1987. Sensor-Based Control of Robotic manipulators Using a General Learning Algorithm *IEEE Journal of Robotics and Automation.*, Vol. ra-3, no. 2, pp. 157-165.

103. Miller, W. T., 1988. An Overview of the CMAC Neural Network, *IEEE Control System Magazine,* pp. 20-27.

104. Miller, W. T., 1989. Real-Time Application of Neural networks for Sensor-Based Control of Robotics with Vision, *IEEE Transactions on Systems, Man, and Cybernetics*}, vol. 19, no. 4, pp. 825-831.

105. Miller, W. T., Glantz, F.H., Kraft, L.G., 1990. An Associative Neural Network Alternative to Backpropagation, *Proceedings of the IEEE.*, Vol. 78, no. 10, pp. 1561-1567.

106. MIRALab: http://www.miralabwww.unige.ch.

107. Mooler, T., Trumbore, B., 1997. Fast, Minimum Storage Ray-Triangle Intersection, *J. Graphics Tools, 2(1): pp.* 21-28.

108. Moore, M., Wilhelm, J., 1988. Collision Detection and Response for Computer Animation, *Computer Graphics*, 22(4), pp. 289-298.

109. Montgomery, K., Bruyns, C., Brown, J., Thonier, G., Tellier, A., Latombe, J. C., 2002. Spring: A General Framework for Collaborative, Real-Time Surgical Simulation, *Medicine Meets Virtual Reality (MMVR02).*

110. Muller, M., Dorsey, J., McMillan, L., Jagnow, R., Cutler, B., 2002. Stable Real-Time Deformations, *Eurographics Symposium on Computer Animation, ACM SIGGRAPH,* pp. 49-54.

111. Narendra, K. S., Parthasarathy, K., 1990. Identification and Control of Dynamical Systems Using Neural Networks, *IEEE Trans. Neural Networks*, vol. 1, pp. 4-27.

112. Narendra, K. S., Mukhopadhyay, S., 1992. Handbook of Intelligent control:Neural, Adaptive, and Fuzzy Approaches, pp. 141-183, Van Nostrand.

113. Narendra, K. S., 1996. Neural Networks for Control: Theory and Practice, *Proceedings of the IEEE*, vol. 84 no. 10, pp. 1385-1406.

114. Nedel, L. P., Thalmann, D., 1998. Real Time Muscle Deformations Using Mass-spring Systems. *Computer Graphics International, Proceedings,* pp. 156-165.

115. Nelson, J., Kraft, L. G., 1994. Real Time Control of an Inverted Pendulum System Using Complementary Neural Networks and Optimal Techniques, *Proceedings of the American Control Conference*, pp. 2553-2554.

116. Nielsen, F. A. Polygon Generation Program. ftp://eivind.imm.dtu.dk

117. Nurnberger, A., Radetzky, A., Kruse, R., 1998. A problem specific recurrent neural network for the description and simulation of dynamic spring models. *Proceedings of the International Joint Conference on Neural Networks,* pp. 486-473.

118. Nurnberger, A., Radetzky, A., Kruse, R., 1999. Determination of elastodynamic model parameters using a recurrent neuro-fuzzy system. *Proceedings of the 7th European Congress on Intelligent Techniques and Soft Computing (EUFIT'99*), pp. 1-8.

119. Nurnberger, A., Radetzky, A., Kruse, R., 2001. Using recurrent neuro-fuzzy techniques for the identification and simulation of dynamic systems. *Neurocomputing 36,* pp. 123-147.

120. Opendx: http://www.opendx.org/.

121. O'Rourke, J., 1998. Segment-Triangle Intersection, *Computational geometry in C (2nd Edition).*

122. Park, J., Kim, S.Y., Son, S.W., Kwon D.S., 2002. Shape retaining Chain Linked Model for Real-Time Volume Haptic Rendering, *Proceedings of the 2002 IEEE Symposium on Volume Visualization and Graphics,* pp. 65-72.

123. Parks, P. C., Militzer, J., 1992. A Comparison of Five Algorithms for the Training of CMAC Memories for Learning Control Systems, *Automatica*, Vol. 28, no. 5, pp. 1027-1035.

124. Pearson, R. K., Pottmann, M., 2000. Grey-Box Identification of Block-Oriented Nonlinear Models, *Journal of Process Control*, 10: pp. 301-315.

125. PHANToM: http://www.sensable.com/index.asp.

126. PooleHospital: http://www.poolehos.org.

127. Provot, X., 1995. Deformation constraints in a mass-spring model to describe rigid cloth behavior. *Proc. Of Graphics Interface'95,* pp. 147-154.

128. Provot, X., 1997. Collision and Self Collision Handling in Cloth Model Dedicated to Design Garments. *Proceeding of Graphics Interface'97,* pp. 177-189.

129. Ropovic, J., Hoppe, H., 1997. Progressive Simplicial Complexes. *Proc. Siggraph* pp. 217-224.

130. Roth, S. H. M., Gross, M. H., Turello, S., Carls, F.R., 1998. A Bernstein-Bezier Approach to Soft Tissue Simulation. *EUROGRAPHICS,* Volume 17, No 3, pp. C285-294.

131. Sayil, S., Lee, K. Y., 2002. An Hybrid Neighborhood Training and Maximum Error Algorithm for CMAC, *World Congress on Computational Intelligent,* 5,31,2002.

132. Schill M.A., Gibson S.F.F., Bender, H.J., Manner, R., 1998. Biomedical Simulation of the Vitreous Humor in the Eye using an Enhanced ChainMail Algorithm, *MICCAI*, pp. 679-687.

133. Schroeder, W.J., Zarge, J. A., Lorensen, W.E., 1992. Decimation of Triangle Meshes. *Computer Graphics(SIGGRAPH'92 Proc.)* 26(2):pp. 65-70.

134. Schumaker, L. L., 1990. Reconstruction of 3D Objects using Splines, *SPIE, Vol. Curves and Surfaces in Computer Vision and Graphics,* pp. 130-140.

135. Shiraishi, H., Ipri, S. L., Cho, D. D., 1995. CMAC Neural Network Controller for Fuel-Injection Systems, *IEEE Trans. on Control Systems Technology*, vol. 3, No. 1, pp. 32-36.

136. Simulab: http://www.simulab.com.

137. Sinha, N. K., 2000. Identification of Continious-Time Systems from Samples of Input-Output Data: An Introduction, *Sadhana*, Vo. 25, Part. 2, pp. 75-83.

138. Sjoberg, J., Zhang, O., Ljung, L., Benveniste, A., Delyon, B., Glorennec, P., Hjalmasson, H., Juditsky, A., 1995. Nonlinear Black-Box Modeling in System Identification. A Unified Overview, *Automatica,* 31(12): pp. 1691-1724.

139. Stanford: http://biocomp.stanford.edu/.

140. Sunday, D., 2001. Intersections of Rays and Segments with Triangles in 3D, May 2001 Algorithms. http://softsurfer.com/algorithm_archive.htm.

141. Szekely, G., Kelemen, A., Brechbuhler, C., Gerig, G., 1996. Segmentation of 2-D and 3-D objects from MRI Volume Data using Constrained Elastic Deformations of Flexible Fourier Contour and Surface Models, *Med. Image Anal.* 1(1), pp. 19-34.

142. Terzopoulos, D., Waters, K., 1991. Techniques for realistic facial modeling and animating. *Proc. Of Computer Animation'91,* pp 59-73.

143. Teschner, M., Girod, S., Girod, B., 1999a. Optimization approaches for soft-tissue prediction in craniofocial surgery simulation. *Second Int. Conf. On Medical Image Computing and Computer-Assisted Intervention MICCAI'99*, pp. 1183-1190.

144. Teschner, M., Girod, S., Girod, B., 2000. Direct computation of nonlinear soft-tissue deformation. *Vision, Modeling, and Visualization VMV'00,* pp. 383-390.

145. Teschner, M., Girod, S., Girod, B., 1999b. Interactive Osteotomy Simulation and Soft-Tissue Prediction, *Proc. Vision Modeling Visualization*, pp. 405-412.

146. Thompson, D. E., Kwon, S., 1995. Neighborhood Sequential and Random Training Techniques for CMAC, *IEEE Trans. on Neural Networks*, vol. 6, no. 1, pp. 196-202.

147. Uhrig, R. E., 1995. Introduction to Artificial Neural networks, *Proceedings of the 1995 IEEE IECON 21st International Conference,* Vol. 1, pp. 33-37.

148. Van den Elsen, P. A., Maintz, J. B. A., Pol, E. J. D., Viergever, M. A., 1995. Automatic Registration of CT and MRI Brain Images Using Correlation of Geometrical Features, *IEEE Trans. Med. Imag.*, 14(2), pp. 384-396.

149. Veltkamp R. C., 1995. Boundaries Through Scattered Points of Unknown Density, *Graphical Models and Image Processing,* 57(6), pp. 441-452.

150. Volino, P., Magnenat-thalmann, N., 1994. Efficient Self-collision Detection on Smoothly discretized Surface Animations using Geometric Shape Regularity. *Proceedings of Eurographics'94,* 13(3): pp. 155-166.

151. Volino, P., Magnenat-Thalmann, N., 1995. Collision and Self Collision Detection: Efficient and Robust Solutions for Highly Deformable Surfaces. *EUROGRAPHICS workshop, Maastricht, The Netherlands*, pp. 55-65.

152. Yang, Y., Magnenat-thalmann, N., 1993. An Improved Algorithm for Collision Detection in Cloth Animation with Human Body. *In Proceedings of Pacific Graphics,* pp. 237-251.

153. Wagner, C., Schill, M. A., Manner, R., 2002. Collision Detection and Tissue Modeling in a VR Simulator for Eye Surgery, *Proceedings of the Workshop on Virtual Environment*, pp. 27-36.

154. Wang, Z., Schiano, J. L., Ginsberg, M. D., 1996. Hash-coding in CMAC Neural Networks, *In Proceedings of International Conference on Neural Networks*, pp. 1698-1703.

155. Webster, R. W., Zimmerman, D. I., Mobler, B. J., Melkonian, M. G., Haluck R. S., 2001. A Prototype Haptic Suturing Simulator, *Medicine Meets Virtual Reality*, pp. 567-569.

156. Wen, R. C., Ker, J. S., Kuo, Y.H., 1996. A CMAC Neural Network Chip for Color Correction *IEEE Conference on Neural Networks*, no. 3, pp. 1943-1948.

157. Witkin, A., Baraff, D., Kass, M., 1997. An introduction to Physically Based Modeling: An Introduction to Continuum Dynamics for Computer Graphics. *SIGGRAPH'97,* http://www-2.cs.cmu.edu/afs/cs/user/baraff/www/pbm/pbm.html.

158. Wong, Y. F., Sideris, A., 1992. Learning Convergence in the Cerebellar Model Articulation Controller, *IEEE Transactions on Neural Networks*, vol. 3, no. 1, pp. 115-121.

159. Woods, R. P., Mazziotta, J. C., Cherry, S. R., 1993. MRI-PET Registration with Automated Algorithm, *Journal of Computer Assisted Topography*, 17(4), pp. 536-546.

160. Wu, X., Downes, M.S., Goktekin, T., Tendick, F., 2001. Adaptive nonlinear finite elements for deformable body simulation using dynamic progressive meshes. *EUROGRAPHICS 2001,* Volume 20, pp.349-358.

161. 3DVIEWNIX: http://www.mipg.upenn.edu/~Vnews/.

162. 3D Doctor: http://www.ablesw.com/3d-doctor/3ddoctor.html.

163. 3D Slicer: http://www.slicer.org/.

164. 3D Café: http://www.3dcafe.com/asp/freestuff.asp.

165. Xu, L., Jiang, J. P., Zhu, J., 1994. Supervised Learning Control of a Nonlinear Polymerization Reactor Using the CMAC Neural Network for Knowledge Storage, *IEE Proc. Control Theory Appl.*, Vol. 141, No. 1, pp. 33-38.