REAL-TIME DEFORMABLE OBJECTS IN COLLABORATIVE VIRTUAL ENVIRONMENTS

by

SELÇUK SÜMENGEN

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of
Master of Science

Sabancı University

August 2006

REAL-TIME DEFORMABLE OBJECTS

IN COLLABORATIVE VIRTUAL ENVIRONMENTS

APPROVED BY:

Asst. Prof. Selim Balcısoy …….

(Thesis Advisor)

Asst. Prof. Serhat Yeşilyurt …….

(Thesis Co-Advisor)

Asst. Prof. Erkay Savaş …….

Asst. Prof. Albert Levi …….

Asst. Prof. Ayhan Bozkurt …….

DATE OF APPROVAL: …….

REAL-TIME DEFORMABLE OBJECTS
IN COLLABORATIVE VIRTUAL ENVIRONMENTS

Selçuk Sümengen

EECS, M.Sc. Thesis, 2006

Thesis Advisor: Asst. Prof. Selim Balcısoy
Thesis Co-Advisor: Asst. Prof. Serhat Yeşilyurt

Keywords: Distributed and Network Virtual Environments, Collaborative Virtual Environments, Physically Based Modeling, Deformable Objects, Real-time Simulation, Computational Geometry and Object Modeling.

Abstract

This thesis presents a method for deformations on closed surfaces in 3D over a network, which is suitable for simulation of tissue and organs for training purposes, as well as cloth simulation in collaborative virtual environments (CVE). CVE's are extensively used for training, design and gaming for several years. To demonstrate a deformable object on a CVE, we employ a real-time physical simulation of a uniform-tension-membrane, based on linear finite-element-discretization of the surface yielding a sparse linear system of equations, which is solved using the Runge-Kutta Fehlberg method. The proposed method introduces an architecture that distributes the computational load of physical simulation between clients. As our approach requires a uniform-mesh representation of the simulated structure, we also designed and implemented an algorithm that converts irregularly triangulated genus zero surfaces into a uniform triangular mesh with regular connectivity. This algorithm uses spring-embedders for stretch optimization of the spherical parameterization step. The strength of our approach comes from the subdivision methodology that enables to use multi-resolution surfaces for graphical representation, physical simulation, and network transmission, without compromising simulation accuracy and visual quality.

İŞBİRLİKÇİ SANAL ORTAMLARDA
GERÇEK ZAMANLI DEFORME OLABİLEN NESNELER

Selçuk Sümengen

EECS, Yüksek Lisans Tezi, 2006

Tez Danışmanı: Yar. Doç. Selim Balcısoy
Yardımcı Tez Danışmanı: Yar. Doç. Serhat Yeşilyurt

Anahtar Kelimeler: Dağıtık ve Ağ Sanal Ortamları, İşbirliği Yapılabilen Sanal Ortamlar, Fiziksel Tabanlı Modelleme, Deforme Olabilen Nesneler, Gerçek Zamanlı Benzetim, Hesaplanabilir Geometri ve Nesne Modelleme.

Özet

Bu tez, network üzerinden çalışan, eğitim amaçlı doku ve organ simülasyonları veya işbirliği yapılabilen sanal ortamlarda kumaş simulasyonları için kullanılabilecek, 3 boyutlu kapalı yüzeylerin deformasyonu için uygun bir metod sunuyor. İşbirliği yapılabilen sanal ortamlar (İSO) uzun yıllardır çok yaygın olarak eğitim, dizayn ve oyun amaçlı kullanılmaktadır. İSO da, deforme olabilen bir nesneyi canlandırabilmek için, doğrusal sonlu eleman bölünmesine uğramış bir yüzeye dayanan eşit gerginlikte bir zarın gerçek zamanlı fiziksel simulasyonu, yüzeyden çıkarılan seyrek denklem sistemi Runge-Kutta Fehlberg methodu ile çözülerek yapılmıştır. Sunulan metod fiziksel simulasyonun hesap yükünü kullanıcılar arasında bölen bir mimari ortaya koyuyor. Yaklaşımımız benzetimi yapılan muntazam bir ağ yapısı gerektirdiği için aynı zamanda düzensiz üçgenlenmiş sıfırıncı takımdan yüzeyleri, düzenli bağlantıları olan muntazam işlenmiş ağ yapılarına çeviren bir algoritma dizaynı yapıldı ve tamamlandı. Algoritma küresel parametrizasyon adımında esnetme optimizasyonu için yay düzenekleri kullanmaktadır. Yaklaşımımız gücünü grafik gösterim, fiziksel simulasyon ve network iletşimi sırasında kullanılan, simulasyon doğruluğu ve grafik gösterminden ödün vermeyen, farklı çözünürlüklü alt bölümlere ayırma metodolojisinden almaktadır.

Acknowledgements

I would like to express my deepest gratitude and appreciation to my advisor, Selim Balcısoy for his continuous support and excellent guidance. I am also thankful for his trust in my abilities and for his efforts in motivating me to pursue graduate study.

I would like to thank my co-advisor Serhat Yeşilyurt, for helping me with my thesis and assisted with the challenging research that lies behind it.

I have been honored to have Albert Levi, Erkay Savaş, and Ayhan Bozkurt as members of my thesis committee. I am grateful for their valuable review and comments on the thesis.

The members of Computer Graphics Laboratory have contributed immensely to my personal and professional life. I add my sincere thanks to Can Özmen, Başak Alper, Ceren Kayalar, and Ekrem Serin. I am indebted to all laboratory members for their contribution, especially to Mustafa Tolga Eren, who helped me with the implementation and experiments.

Thanks to all my friends and colleagues for their support. I am very grateful for the time spent with the friends and memories.

Finally, I would like to thank my parents for their love and support.

This work was financially supported by TUBITAK (The Scientific and Technological Research Council of Turkey) through a research fellowship.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CVE	:	Collaborative Virtual Environment
P2P	:	Peer-to-peer
UK	:	United Kingdom
DVE	:	Distributed Virtual Environment
DIVE	:	The Distributed Interactive Virtual Environment
3D	:	Three Dimensional
NPSNET	:	Naval Post Graduate School Network
MASSIVE	:	Model, Architecture and System for Spatial Interaction in Virtual Environments
KHz	:	Kilohertz
DOF	:	Degree of Freedom
BEM	:	Boundary Element Method
FEM	:	Finite Element Method
2D	:	Two Dimensional
LAPACK	:	Linear Algebra Package
BLAS	:	Basic Linear Algebra Subprograms
UDP	:	User Datagram Protocol
FPS	:	Frames per Second
CPU	:	Central Processing Unit
GPU	:	Graphics Processing Unit

LIST OF SYMBOLS

$O(\ )$	:	Big O Notation
χ	:	Euler Characteristics
φ	:	Mapping Function
$\rightarrow$	:	Transformation
Θ	:	Longitude
Φ	:	Colatitude
r	:	Radius
C^1	:	Continuous First Derivative

1. INTRODUCTION

1.1. Motivation

Collaborative Virtual Environments (CVE) are being extensively used for training, design and gaming for several years. They enable participants to get immersed into a Virtual Environment where they can perform a task or experience a story together. In most use cases such as gaming and education, current CVE's are sufficient to address user expectations related to visual realism, animations and networking. However, CVE's also involve substantial amount of interaction between the users and the objects in the synthetic worlds, which should be visually appealing and physically realistic as well. Current CVE's are mostly limited to avatar-avatar interaction or the object interactions are animated using offline techniques and they are commonly hard-coded into the application. Real-time physical simulation of deformable bodies in CVE's, enables accurate replication of the real world objects like cloth.

On the other hand, medical and some engineering applications definitely require real-time accurate simulations. When users want to train on a surgical operation, a CVE should support accurate simulation and visualization of an organ's deformation for each participant in real-time. Also, haptic devices are used in such simulations to enable force feedback and train the operator with the aid of visual display as well the sense of touch. Haptic rendering necessitate real-time and accurate physical simulation, since it requires stable values of simulated environment to generate force feedback.

1.2. Outline of the thesis

This thesis proposes a deformable body simulation and visualization framework for collaborative virtual environments and distributed haptic platforms.

First, we summarize the parallel research fields with our approach, and give brief information about the related works in Chapter 2. Key concepts are also introduced in Chapter 2, and the models associated with the methods that we applied, are gathered together.

In Chapter 3, we provide an agenda for the numerical methods that we used to perform physical simulation as a reference for the implementation details.

Our approach is explored in detail and the methods that we adopted are examined in Chapter 4. We introduced a generic mesh representation for our deformable models and proposed a method for the conversion of irregular genus zero models into a regular mesh with a tetrahedral domain. We present a subdivision scheme for our mesh representation that allows fast refinement using convolution kernels, and enables high resolution surfaces for an enhanced visual display while maintaining the network and simulation models at optimal resolutions. Physical simulation characteristics and theoretical aspects of deformations are presented in the same chapter. Proposed linear finite element model is explained and theoretical background is reviewed. In the Chapter 4, we gave details of network architecture, and shared environment model that allows peer-to-peer (P2P) collaboration with distribution of simulation load among peers. The synchronization of graphics, network and physical simulation models is an important concept, and is clarified in the last section of this chapter.

In Chapter 5, we present the results of our approach with the rendered images, and the numerical outputs of the physical simulation. We also tested the stability of the physical simulation and effects of the parameters. Network requirements and the performance are explored as well.

2. RELATED WORK

2.1. Collaborative and Distributed Network Virtual Environments

2.1.1. Introduction

Collaborative Virtual Environments (CVE) became an active research area, especially after the rapid development and raising popularity of the Virtual Reality technology that enables users to interact with each other in a computer simulated environment. First international conference on CVE's, CVE'96 was held in Nottingham, UK, and followed by CVE'98 at Manchester, UK. Definition of the collaborative virtual environments as an introduction to the CVE conferences was made by Churchill and Snowdon as follows:

> *"A Collaborative Virtual Environment (CVE) is an application that uses a Virtual Environment to support human-human and human-system communication. Within such virtual environments, multiple users can convene, communicate and collaborate." [1]*

Given this broad definition of CVEs, the field is open for researchers from disciplines such as psychology, sociology, work practice studies, architecture, artificial intelligence and art [2] as well as computer scientists. According to this approach, shared spaces where people can communicate and collaborate, including text based environments can also be classified as CVEs.

Four key features of the CVEs are stated as shared context, awareness of others, negotiation and communication, and flexible viewpoints [1]. *Shared context* can be interpreted as the shared knowledge of current and past activities, shared artifacts and environments that facilitates a common understanding between collaborators.

Awareness is an understanding of the activities of others, which provides a context for your own activity [3]. Awareness facilitates collaborative working of the group by preserving the relevance of individual contributions. Collaborative work requires *negotiation* on roles and activities as well as conversations between collaborators. *Flexible and multiple viewpoints* provide different representations for individuals having different tasks.

Although the debate on conceptual definition of CVEs among scientists and researchers continues, such a collaborative virtual environment should be synchronized and distributed to all participating sites in order to give the users the illusion of being located in the same place at the same time [4]. Distributed Virtual Environments, also referred in the literature as Network Virtual Environments, provide a framework for CVEs. The terms, CVEs and DVEs are often mistakenly taken identical, but they are complementary, where the CVEs give emphasize to the interaction among individuals and joint activities in a shared space, DVEs mostly deals with the maintenance of a virtual worlds synchronized and distributed over participants.

One of the reasons that motivate the research on DVEs, is the need for a virtual environment that looks accurate, and it is accurate in details, thus individual computers are incapable of representation of such an environment, and it requires multiple computer systems [5]. Approaching to the reality, the virtual environment should approximate the complexity of the real world. Human-computer interactions using the representations of real world objects should be consistent with real world physical constraints.

Stytz [5], states the display of a virtual environment must accurately express the real world objects that it portrays, and introduces notion of fidelity in the DVE. *Sensory fidelity* requires the accurate replication of real world, using visual, auditory and tactile information. Accurate depiction of the gravity, motion, energy consumption and conservation is the *physical fidelity*. Objects having the correct relative scales and velocities among each other within each a computer and the distributed environments is the *modeling fidelity*. Assuring the limits of the latency and lag between one action and a notification of it by other participants is the *time fidelity*. All of these

classifications of fidelity and the later ones that are not mentioned here address the desire of making virtual environments “real” (Figure 2.1).

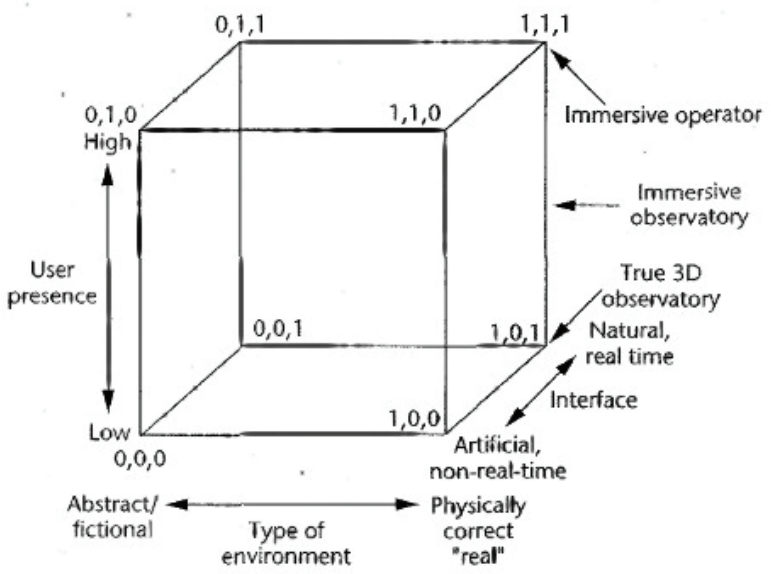

Figure 2.1: Dimensions of the Virtual Environment Technology. [5]

2.1.2. Historical Timeline of Collaborative and Distributive Virtual Environments

DIVE [6] is one of the first Distributed Virtual Environment that allows participants to collaborate in a 3D virtual world which facilitates audio, video and text transmission for communication. DIVE also allows interaction with virtual objects. On the other hand, NPSNET [7] is designed for military training and simulation for networked environments and each participant machine acts as a military vehicle or a dismounted person and is able to interact in the virtual world. NPSNET also introduced virtual humans which area actually human like avatars. MASSIVE is especially used for public participation and performance [8].

There are only a few approaches that in particularly deal with the significance of physical simulation in collaborative virtual environments. A recent work by Jorissen [9], gives a detailed survey on state of the art of dynamic interactions and physical simulations in CVE’s. Jorissen et al. introduces a collaborative virtual environment, where the object-object interaction is allowed in

addition to avatar-object and avatar-avatar interactions using a non-commercial physics engine.

There are few attempts to introduce deformable objects into CVE's: Dequidt et al. [10] propose a system based on Ghost objects to handle network latency. Ghost objects are associated to objects manipulated over the network and introduced into the client side to perform physical simulations asynchronously at each user.

Collaborative Haptics Environments are also introduced to handle surgical training and simulations with the use of special architectures [11]. Haptic rendering must be performed at simulation rates higher than 1 KHz and most approaches require particular hardware or a computer architecture running real-time operating systems [12]. Distributed and collaborative haptic visualization of a 1 DOF crank model is also achieved using client-server architecture building a haptic communication library allowing real-time communication needed for haptic rendering only on Intranets [13].

2.2. Deformable Objects

Simulation of deformable objects is a significant research area for over two decades since it enables the cloth animations, tissue modeling, virtual surgery, and many more applications in the field of computer graphics. Early approaches on the visualization of deformable models used non-physical and purely geometric techniques, most of which is classified as Free-Form-Deformations [14]. Physics based approaches appeared afterwards and gained a popular attention by enabling cloth animations [15]. This approach, which is a linear model based on energy minimization and continuing approaches using explicit integration schemes, are suffering from stability issues for large body deformations. Baraff and Witkin [16], introduced an implicit integration scheme for stable simulations using large time steps. On the other hand, real-time simulation of deformable models is an other challenge, and linear mass-spring models introduced at first [17]. As an alternative, Boundary Element Method (BEM) is introduced, which is inspired by Finite Element Method (FEM), however, considers only the surface of the model [18]. Non-linear

FEM's models are not suitable for real-time simulations since they are computationally intensive, so deformable objects simulfations for cloth simulations in virtual environment continued to use improved mass-spring models [19, 20]. Also, pre-computed models for real-time dynamic deformations are considered [21]. Since medical applications require real-time and accurate simulations some approaches used FEM to parameterize the mass-spring model to improve accuracy [22].

3. NUMERICAL METHODS

3.1. Introduction

In the Numerical Methods Chapter, we introduce the key concepts defining the mathematical procedure and computations handled for a typical physical simulation. Further reference can be found in Siggraph Course notes on Physically Based Modeling [23] by Baraff and Witkin, Numerical Recipes text book [24] as well as LAPACK and BLAS software libraries.

3.2. Initial Value Problems

When the behavior of a system is described with an ordinary differential equation of the form:

$$\dot{x} = f(x,t), \tag{3.1}$$

where, f is a known function, x is the state of the system, $\dot{x}$ is the time derivative. x has a starting value given $x(t_0) = x_0$, and it is desired to find $\dot{x}_i$ at some final point x_f or at some discrete list of points.

3.2.1. Euler Integration

3.2.2. Explicit Integration (Forward Euler)

Forward Euler Integration is a basic numerical integration scheme, approximating the true integral by fallowing the trajectory as a polygonal path.

$$x(t+\Delta t) = x(t) + \Delta t f(x,t) \tag{3.2}$$

Euler's integration is not accurate and error introduced by Euler integration can be found by the Taylor series expansion of $x(t+\Delta t)$.

$$x(t_0+\Delta t) = x(t_0) + \Delta t\dot{x}(t_0) + \frac{1}{2}\Delta t^2\ddot{x}(t_0) + O(\Delta t^3) \tag{3.3}$$

and the difference between the Euler integration is the error introduced,

$$\frac{1}{2}\Delta t^2\ddot{x}(t_0) + O(\Delta t^3) \tag{3.4}$$

In the expression Δt^2 is the domination term and the error introduced by the integration is of order Δt^2.

$$Error = O(\Delta t^2) \tag{3.5}$$

3.2.3. Mid-point Method

Error bound in the equitation 2.4 can be easily reduced to Δt^3 by slight modification of the Forward Euler Integration,

$$x(t+\Delta t) = x(t) + f\left(f\left(\frac{x+\Delta t}{2}, \frac{t+\Delta t}{2}\right), t\right) \tag{3.6}$$

This method is called mid-point method since uses a midpoint evaluation of f. Mid-Point Method is a second order solution method and has an error bound of Δt^2.

3.2.4. Runge-Kutta-Fehlberg Method

Runga-Kutta methods propagate a solution over an interval by evaluating f for several steps. Fehlberg has developed a fifth order Runge-Kutta method that has an error bound of Δt^5.

$$q_1 = f(x,t) \tag{3.7.a}$$

$$q_2 = f\left(t + \frac{\Delta t}{4}, x + \frac{\Delta t q_1}{4}\right) \tag{3.7.b}$$

$$q_3 = f\left(t + \frac{3\Delta t}{8}, x + \Delta t\left(\frac{3q_1}{32} + \frac{9q_2}{32}\right)\right) \tag{3.7.c}$$

$$q_4 = f\left(t + \frac{12\Delta t}{13}, x + \Delta t\left(\frac{1932q_1}{2197} - \frac{7200q_2}{2197} + \frac{7296q_3}{2197}\right)\right) \tag{3.7.d}$$

$$q_5 = f\left(t + \Delta t, x + \Delta t\left(\frac{439q_1}{216} - 8q_2 + \frac{3680q_3}{513} + \frac{845q_4}{4104}\right)\right) \tag{3.7.e}$$

$$q_6 = f\left(t + \frac{\Delta t}{2}, x + \Delta t\left(\frac{-8q_1}{27} + 2q_2 - \frac{3544q_3}{2565} + \frac{1859q_4}{4104} - \frac{11q_5}{40}\right)\right) \tag{3.7.f}$$

$$x(t + \Delta t) = x(t) + \Delta t\left(\frac{16q_1}{135} + \frac{6656q_3}{12825} + \frac{28561q_4}{56430} - \frac{9q_5}{50} + \frac{2q_6}{55}\right) \tag{3.7.g}$$

3.3. Matrix Representation Schemes

Real-time operation of our approach is strictly depended on matrix-vector multiplications. Finite element discretization step results in very large sparse matrices and these matrices are updated regularly. Accessing matrix elements in a shorter time is an important constraint while keeping the matrix in a compact form.

3.3.1. Diagonal Storage Scheme

Finite element of finite state discretization generally produces diagonal matrices that are mostly sparse. Efficient storage of these matrices is also important for an efficient multiplication. This storage scheme is simple and stores the diagonals having non-zero elements. Non-empty diagonals are stored consecutively on an array, and an index array is maintained to identify the diagonal by their distance from the

main diagonal.

$$M = \begin{bmatrix} 1 & 0 & 8 & 0 \\ 0 & -2 & 0 & 9 \\ 0 & 2 & 6 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}, \quad Values = \begin{bmatrix} * & 1 & 8 \\ 0 & -2 & 9 \\ 2 & 6 & * \\ 0 & 3 & * \end{bmatrix}, \quad \text{Distance} = \begin{bmatrix} -1 & 0 & 2 \end{bmatrix} \tag{3.8}$$

For an n × n matrix having d non zero diagonals, when multiplied by a vector, this storage scheme performs O(dn) multiplications. Accessing a matrix element takes O(1) time, however insertion of a non-zero element is O(n) and this scheme also keeps redundant zero elements on the diagonals having any non-zero value.

3.3.2. Coordinate Format Storage Scheme

This scheme is designed for sparse matrices having no regular structure. It keeps the non-zero elements consecutively on a data array and maintains two index arrays to identify row, and column number of the elements on the data array.

$$M = \begin{bmatrix} 1 & 0 & 8 & 0 \\ 0 & -2 & 0 & 9 \\ 0 & 2 & 6 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}, \qquad \begin{aligned} values &= \begin{bmatrix} 1 & 8 & -2 & 9 & 2 & 6 & 3 \end{bmatrix} \\ rows &= \begin{bmatrix} 1 & 1 & 2 & 2 & 3 & 3 & 4 \end{bmatrix} \\ columns &= \begin{bmatrix} 1 & 2 & 2 & 3 & 3 & 4 & 4 \end{bmatrix} \end{aligned} \tag{3.9}$$

For an n x n matrix having k non zero elements, when multiplied by a vector, this storage scheme performs O(kn) multiplications. Accessing a matrix element takes O(log(n)) time, and insertion is O(n) and completely forming the matrix is O(nlog(n)) time.

However, we make a modification to existing scheme, introducing additional pointers exploiting the geometric affinity of stored elements and reducing access time to O(n), also on k consecutive insertions, if k > log(n), we form the matrix again to reduce insertion time (Appendix A).

4. NETWORK DEFORMABLE OBJECTS

4.1. Introduction

Our method applies a collaborative deformation on a linear membrane model over network, which is appropriate for simulation of tissue and organs for training purposes, as well as cloth simulation in the virtual environments. 3D models of these objects with reasonable parameters are necessary for a realistic visualization and simulation. Our approach works with genus zero surfaces, which are suitable for representing such objects (Figure 4.1).

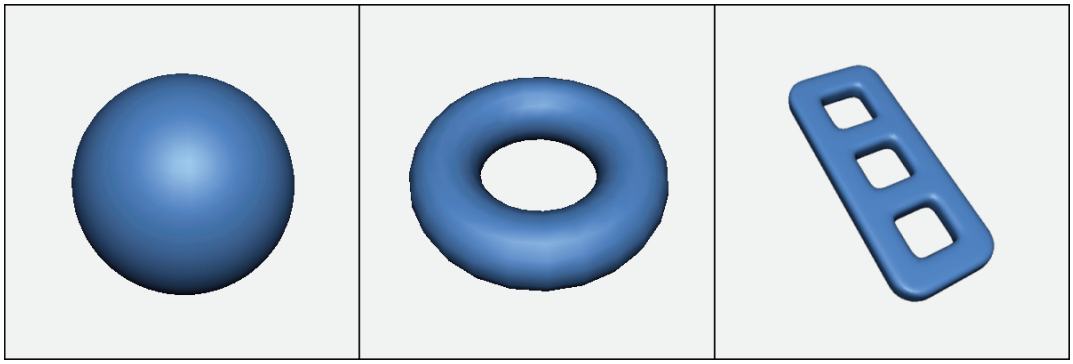

Figure 4.1: Examples of genus zero, genus one, and genus three surfaces.

4.2. Geometric Model

Since the proposed approach requires a uniform representation of the simulated structure, restriction on the genus of the model allows us to construct a regular 2D grid that corresponds to the surface of the model.

The *genus* of a connected, orientable surface is an integer representing the maximum number of cuttings along closed simple curves without rendering the

resultant manifold disconnected [25]. In other words, genus is the number of holes or handles on a closed surface. Sphere has genus zero, and torus has genus one.

Also, the following relationship holds for the genus of a surface,

$$\chi = 2 - 2g, \tag{4.1}$$

where the *Euler characteristic* χ for a polyhedron defined as,

$$\chi = V - E + F, \tag{4.2}$$

and V is the number of vertices, E is the number of edges and F is the number of the faces [26].

The surface of any convex polyhedron is homeomorphic to a sphere and has Euler characteristic of 2. Homeomorphic spaces are identical from the viewpoint of the topology [27], therefore genus zero surfaces preserve their topological properties under spherical parameterization and can be mapped onto a convex regular polyhedron.

4.2.1. Mesh Representation

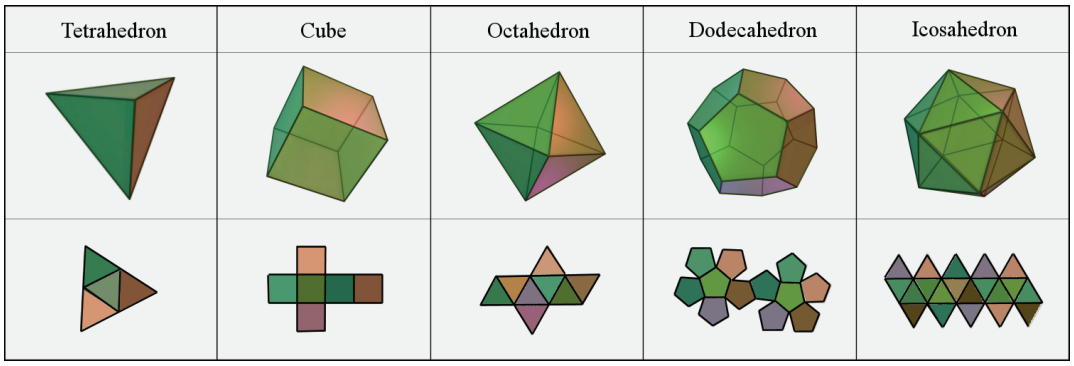

Figure 4.2: Five platonic solids and their flattened view.

There are five convex regular polyhedrons that are also called platonic solids (Figure 4.2). Tetrahedron, cube and octahedron can be unfolded onto a plane easily and they are good candidates to form a domain for regular meshes while dodecahedron and icosahedron have much more complex flattened structure having

twelve and twenty faces respectively.

We have chosen tetrahedron as the domain for our mesh representation, since it has four equilateral triangular faces that can be represented as a 2D grid having (2^n + 1) x (2^{n+1} + 1) nodes where n >= 0 (Figure 4.3).

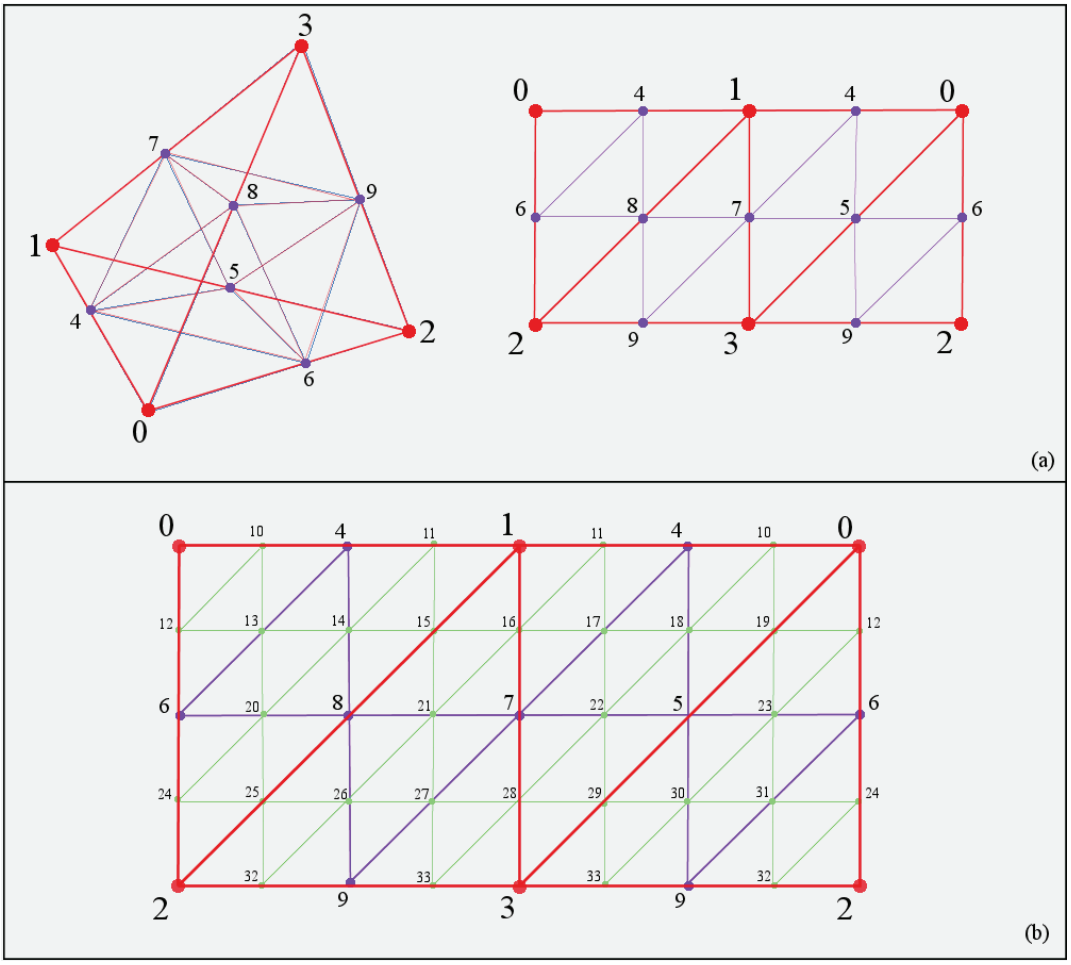

Figure 4.3: 2D Grid representation of tetrahedron, (a) n = 1, (b) n = 2.

4.2.2. Mesh Generation

We propose an algorithm that converts irregularly triangulated genus zero surfaces into a uniform mesh with regular connectivity. Previous approach for constructing regular meshes with fixed and simple topology by Hoppe [28], generates a spherical parameterization of the surface and the domain. Surface, projected on the

sphere, mapped on to the domain, and unfolded to generate the geometry image. We apply a similar procedure, but we introduce different techniques for spherical parameterization and model re-meshing. It allows adjusting the tradeoff between face area uniformity of the generated mesh, and preserving the accuracy with the original mesh.

4.2.2.1. Spherical Parameterization

In our approach, success of the regular mesh generation strictly depends on the spherical parameterization step, and the parameterization of a detailed input mesh is a computationally intensive process that often requires reasonably large amount of time. Our method combines some of well known techniques and introduces several improvements, and taking the advantage of recently available graphics hardware.

Given a triangle mesh M, the problem of spherical parameterization is to form a continuous invertible map $\varphi : S \rightarrow M$ from the unit sphere to the mesh [28]. Spherical parameterization of regular tetrahedral domain D, and irregular input mesh M are necessary to generate Sphere to Mesh ($S \rightarrow M$) and Sphere to Domain ($S \rightarrow D$) mappings that will allow us to perform Mesh to Sphere and Sphere to Domain ($M \rightarrow S \rightarrow D$) transformation.

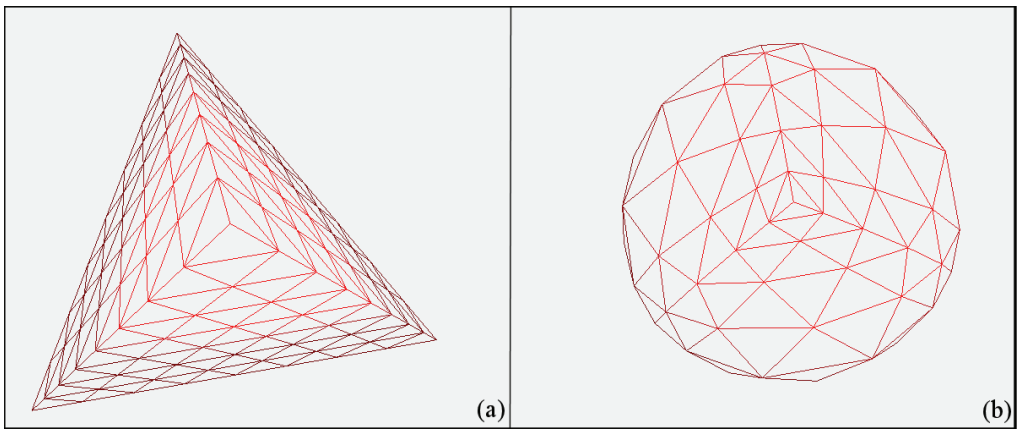

Figure 4.4: Gnomonic Projection of Tetrahedron.

Any convex polyhedron can easily be projected onto a unit sphere switching to spherical coordinate system (Θ, Φ, r) and setting a unit radius for all vertices

(Gnomonic Projection), however translation between each mesh triangle and spherical triangle might introduce a certain amount of distortion (Figure 4.4). Spherical triangle mapping alters the uniformity of domain tessellation and there are certain mapping methods other than "Gnomonic Projection". Some of these methods [28] focus on different uniformity measures such as area or edge length uniformity and some of them apply several stretch optimization techniques. Since the aim of spherical parameterization is to perform mesh conversation from an irregularly triangulated input mesh, it has been shown that the under sampling is directly related to the stretch of a parameterization [29].

Previous approaches define a stretch norm to measure the stretch efficiency and concludes that minimizing the stretch norm is a non-linear optimization problem [28, 29]. We attack this problem by a modification of well known technique used for graph drawing. Graph drawing using force directed placement methods, which are also called spring-embedders, distributes vertices evenly in the frame and minimize edge crossings while favoring uniformity of the edge lengths [30]. Since we implemented a deformable physics engine that can handle mass spring systems efficiently, we introduce a variant of spring-embedders for stretch optimization.

A spring-embedder model is generated from the gnomonic projection of the domain. Every vertex has a constant mass, and springs are introduced in between neighboring vertices. An external force field (Equitation 4.3) is applied from the center of the domain that limits displacements of vertices on the unit sphere.

$$f_{External_i} = \left(1 - \|x_i\|\right) \times \hat{x}_i \qquad \forall i,\ 0 \le i \le nNodes. \tag{4.3}$$

Springs between the vertices tend to preserve initial edge lengths and resists movements that change the topology; however we need to establish a tension on these springs to perform stretch optimization. We scale down the positions of the vertices that are projected onto unit sphere (Equitation 4.4), and external force which is applied continuously expands the vertices onto the unit sphere again while producing a tension on the springs.

$$x_{i_{new}} = C \times x_i, \qquad \forall i,\ 0 \le i \le nNodes,\ 0 < C < 1 \tag{4.4}$$

Tensions on the springs do not affect the vertex placements on the unit sphere unless the stiffness parameters are adjusted. Given that we are seeking a uniform spherical parameterization, stiffness of each spring is re-adjusted proportional to the areas of neighboring faces (Equitation 4.6). Stiffness parameters are updated continuously to achieve area uniform tessellation over the unit sphere (Figure 4.6).

$$area_{mean} = \sum_{i=0}^{nFaces} area_{face_i} / nFaces, \qquad area_{deviation} = \sum_{i=0}^{nFaces} \left(area_{face_i} - area_{average}\right)^2 / nFaces,$$

$$area_{face_i} = \left(area_{face_i} - area_{mean}\right) / area_{deviation}, \qquad \forall i,\ 0 \le i \le nNodes. \tag{4.5}$$

$$stiffness_i = \left(area_{face_{i0}} + area_{face_{i1}}\right) / 2 + 1, \qquad \begin{array}{c} \forall i,\ 0 \le i \le nEdges \\ face_{i0} \ \ and \ \ face_{i1} \ \ are \ \ adjacent \ \ to \ \ edge \ \ i. \end{array} \tag{4.6}$$

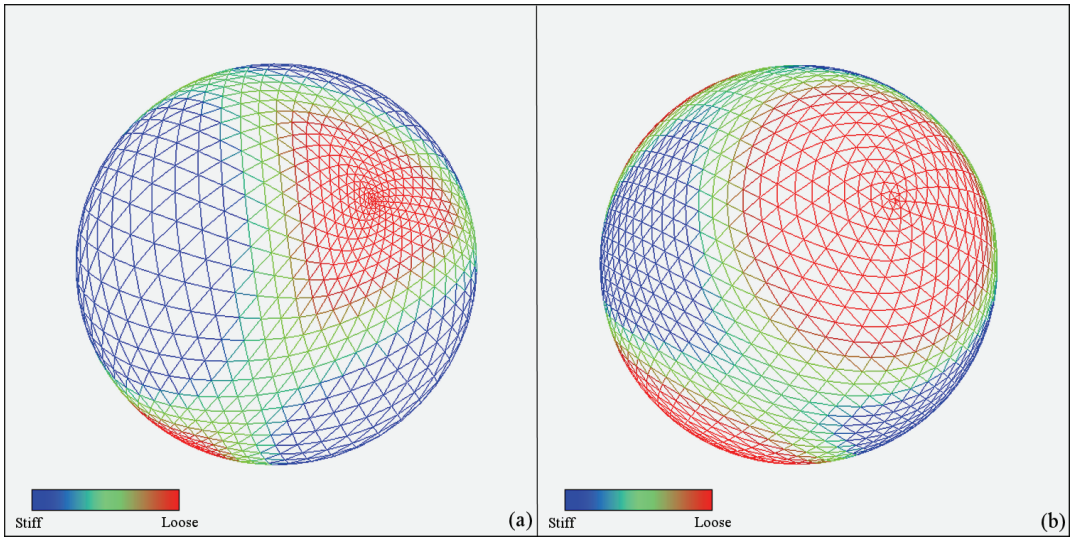

Figure 4.5: (a) Gnomonic Projection of Tetrahedron. (b) Stretched Gnomonic Projection of Tetrahedron.

Our proposed force model is a feasible stretch optimization technique for domain to sphere mapping; however, it is insufficient for mesh to sphere mappings where the projection of non-convex polyhedron into a unit sphere results in edge

crossings and does not preserve initial surface topology. We use a vertex displacement procedure that is similar to the relaxation method (Equation 4.7) of previous spherical parameterization approaches [31] for each time step to overcome this problem (Figure 4.6).

$$x_{i_{new}} = \sum_{j=0}^{nNeighbors_i} x_{ij} / nNeighbors_i, \qquad \begin{array}{c} \forall i, \ 0 \le i \le nNodes, \\ x_{ij} \ \ is \ \ j_{th} \ \ neighbour \ \ of \ \ x_i. \end{array} \tag{4.7}$$

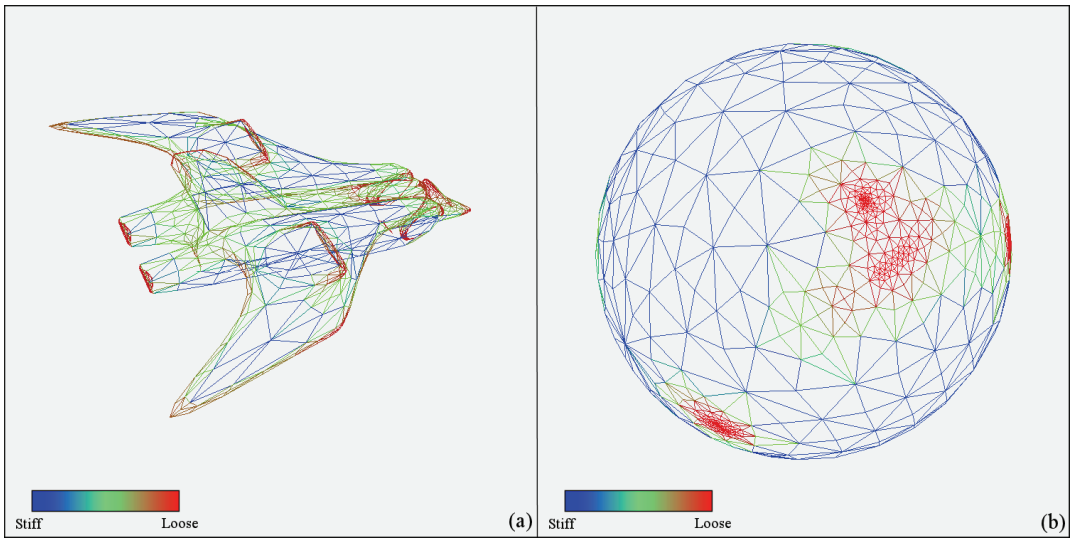

Figure 4.6: (a) Irregular Input Mesh. (b) Stretched Gnomonic Projection of Input Mesh.

4.2.2.2. Model Re-meshing

After spherical parameterization of convex polyhedron domain (D→S) and the input mesh (M→S), it is straightforward to generate inverse function of the domain to sphere (S→D) mapping. Combining the spherical mappings mesh to sphere (M→S) and sphere to domain (S→D) to derive mesh to domain mapping (M→D), requires intersection of the sets on the sphere. However, transformed vertex coordinates of the mesh and domain might not intersect on the sphere, and vertices of the domain might fall inside of a mesh facet. For each vertex of the domain, intersecting face of the parameterized mesh should be found out and 3D coordinates of domain vertex should be computed by interpolating the vertices of the intersecting

face (Figure 4.7).

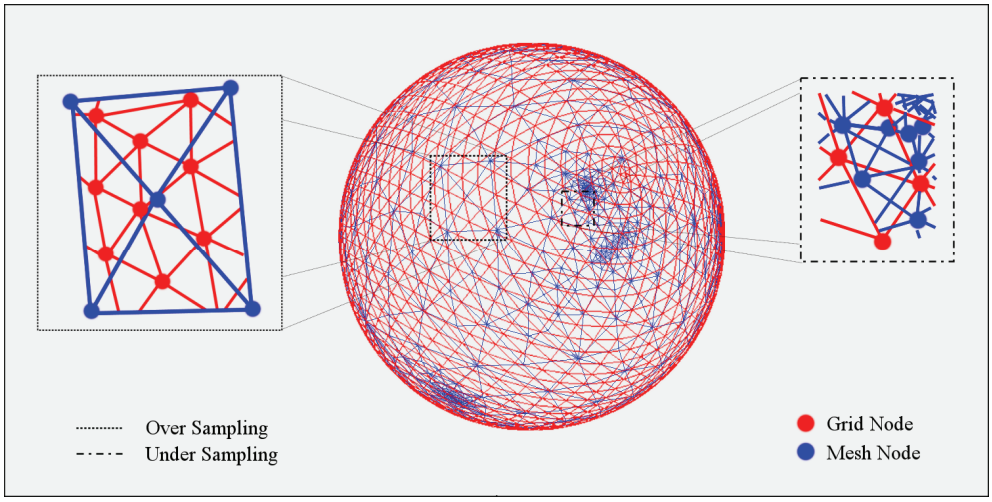

Figure 4.7: Intersecting Spherical Projections of Tetrahedral Domain and Input Mesh.

Since determining the intersecting faces of the parameterized mesh for each vertex on the domain and computing the interpolated coordinates are costly procedure, we introduce a fast method taking the advantage of recent advances in graphics hardware [32].

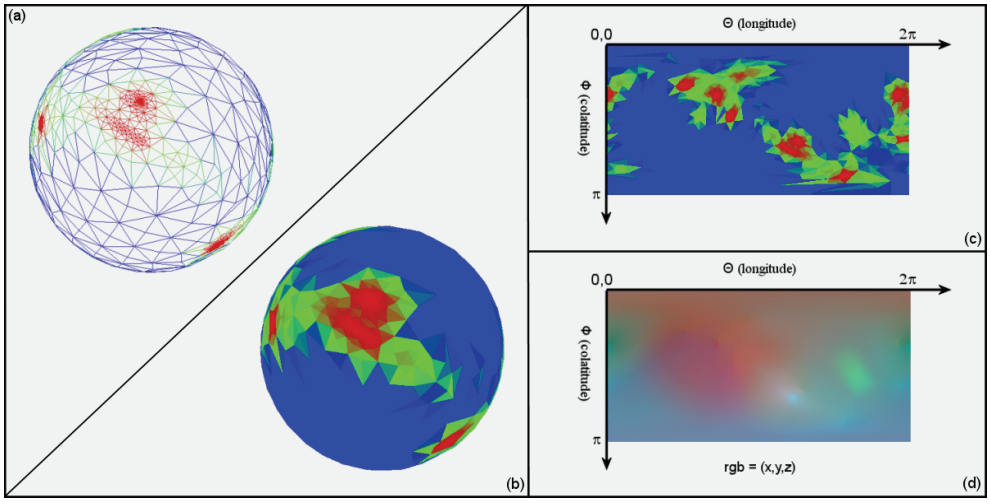

Figure 4.8: Spherical projection of input mesh is, (a) rendered as 3D wireframe, (b) 3D colored surface, (c) 2D colored surface, and (d) 2D colored surface, where the original positions of vertices are used as color components.

Using OpenGL and programmable shaders, we render the faces of the parameterized mesh onto the frame buffer using the two dimensional spherical coordinates (Θ and Φ) of the transformed vertices (Figure 4.8). Initial Cartesian coordinates (x, y, and z) of the parameterized mesh vertices are attached to color attributes (r, g, and b) at the vertex shader, and inside of each face is filled with the interpolated Cartesian coordinates at the fragment level. Rendered image is then fetched from the frame buffer as a 2D texture and used like a lookup table to generate 3D coordinates of the domain vertices (Figure 4.9).

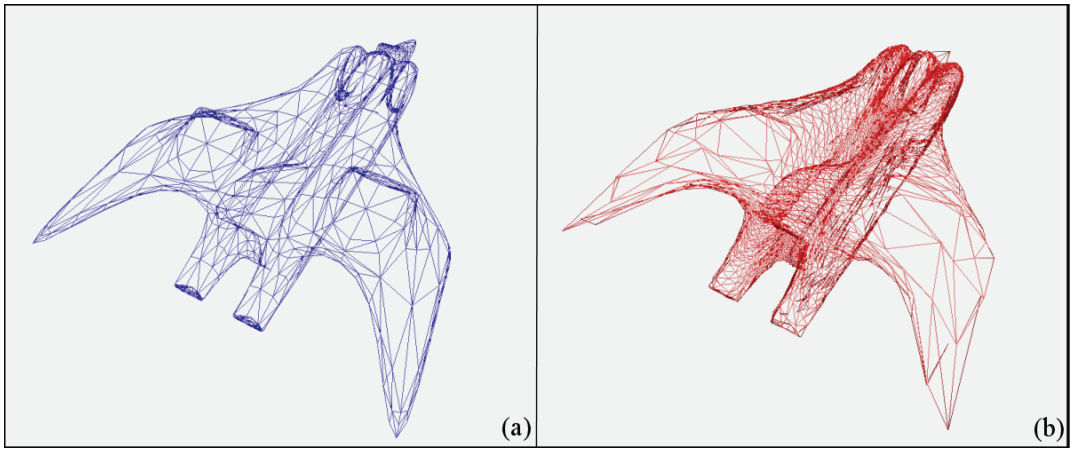

Figure 4.9: Final comparison of (a) the input mesh with 1444 vertices, and (b) the resulting regular mesh with 129*65 vertices.

4.2.3. Subdivision Scheme using Convolution Kernels

Subdivision methodology is appropriate for our approach since it allows multi-resolution representation of a surface and fast switching between detail levels. It also favors numerical stability [33], so it is highly suitable for physical simulation of deformations using finite element and finite difference methods.

We used a variant of butterfly subdivision scheme [34] that generates a C^1 smooth triangular mesh. Modified Butterfly Scheme is an *interpolating subdivision* scheme, where the original vertices (control points) are also the vertices of the refined surface and surface is interpolating to a limit surface. This behavior makes it possible to use surfaces with different resolutions for graphical representation, physical

simulation, and network transmission, without compromising the integrity of simulation accuracy and the rendered image.

There are two different methods of refinement for subdivision schemes. Subdivision schemes that perform face split for each refinement level are defined as *primal schemes*, and the other schemes that perform vertex split are called *dual schemes.* Primal subdivision schemes introduce new vertices at each refinement which are called *odd vertices*, and vertices inherited from the previous level are called *even vertices.* Moreover, this naming convention comes from the node numbering of the one dimensional case. Also vertices are identified as regular and extraordinary vertices depending on their valances. Subdivision schemes, that are defined for triangular meshes create new vertices of valance 6 in the interior of the surface, and 4 on the boundaries. These vertices having valance 4 and valance 6 are defined as *regular vertices* and vertices of other valances are called *extraordinary vertices*.

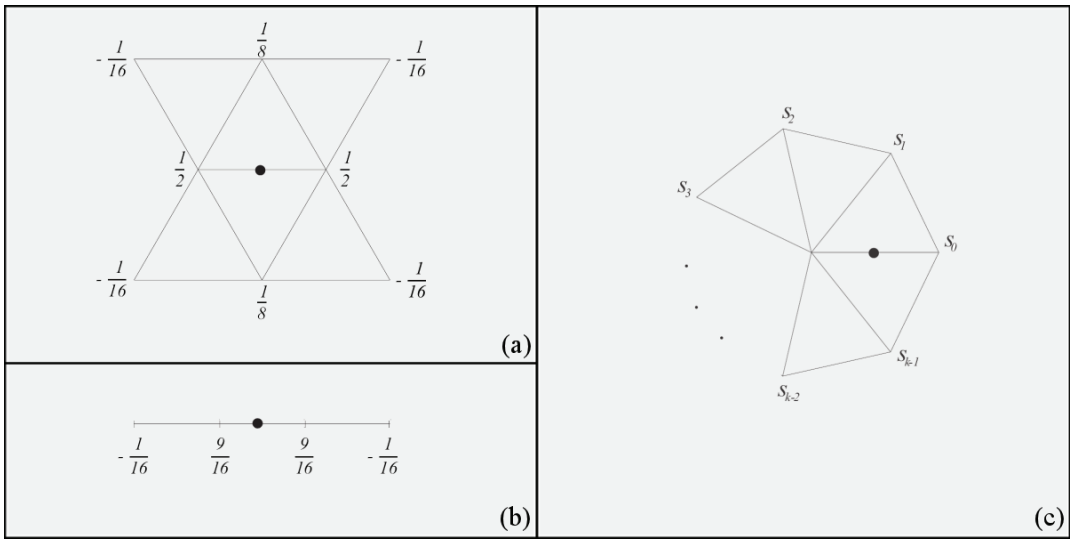

Figure 4.10: (a) Mask for interior odd vertices with regular neighbors, (b) Mask for crease and boundary vertices, (c) mask for odd vertices adjacent to extraordinary vertices. The coefficients s_i are $1/k\,(1/4 + \cos(2i\pi/k) + 1/2\cos(4i\pi/k))$ for $k > 5$. For $k = 3$, $s_0 = 1/12$, $s_{1,2} = -1/12$; for $k = 4$, $s_0 = 3/8$, $s_1 = 1/8$, $s_{1,3} = 0$ [33].

Modified butterfly scheme is a primal subdivision scheme, and *masks* are used to generate odd vertices from the values of even vertices at each refinement

(Figure 4.10). There are two groups of masks for odd vertices. Interior odd vertices that are adjacent to regular vertices and odd vertices that are adjacent to boundary or crease vertices have generated by two masks having constant coefficients. Masks for odd vertices that are adjacent to an extraordinary vertex have changing structure and varying coefficients depending on the valance of extraordinary vertex.

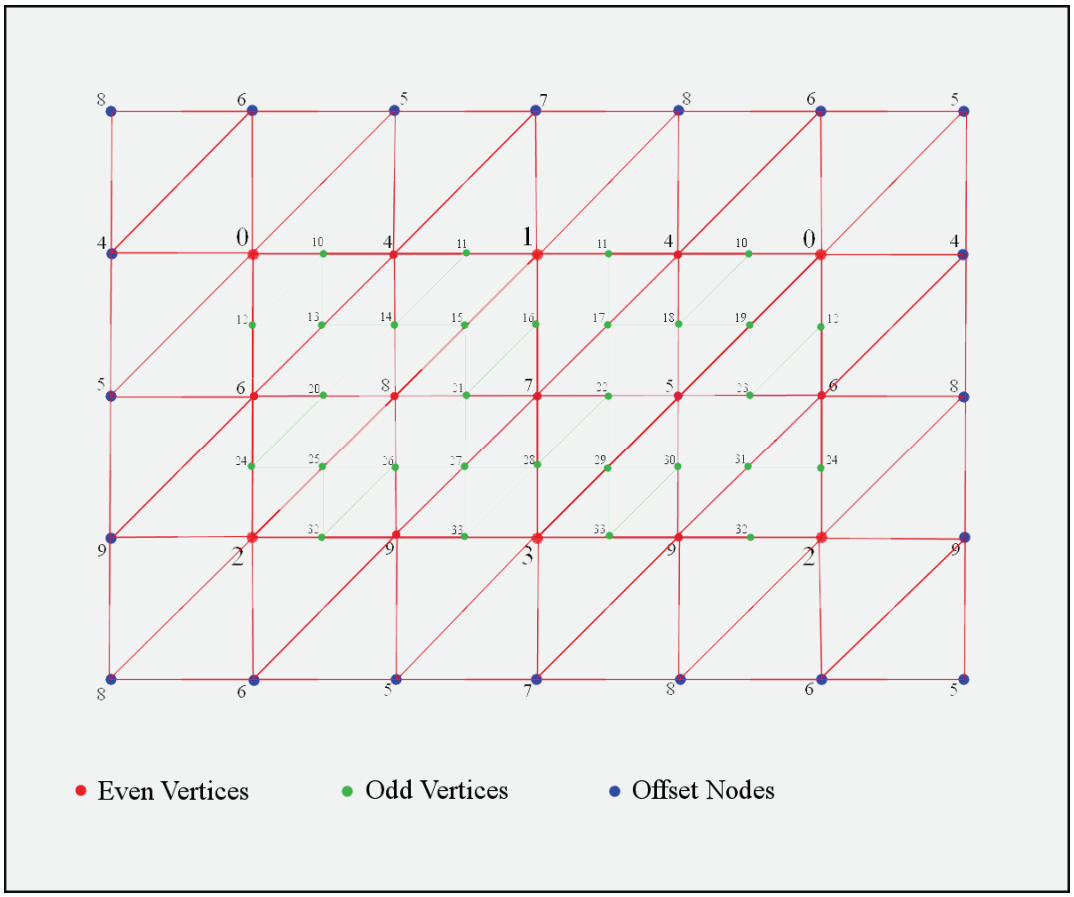

Figure 4.11: Figure 4.10: Modified 2D Grid Structure.

After having a regular mesh representation as a 2D grid structure, we introduce some modifications (Figure 4.11) to apply a fast and robust refinement strategy using modified butterfly scheme. Taking advantage of having a regular domain, we have no boundary or crease vertices, but there are 4 extraordinary vertices of valances 3 on the corners of the tetrahedral domain. However, if we duplicate the edges of these vertices, they can be treated as regular vertices having valances of 3×2. Since the duplicate edges are symmetric to existing edges, resulting odd vertices will have same values. This modification allows us to use the mask for interior odd

vertices with regular neighbors for all the grid nodes. We also introduce offsets to 2D grid representation. Offsets are the copies of grid nodes, assuring existing neighboring properties and they are updated before the convolution process (Figure 4.12).

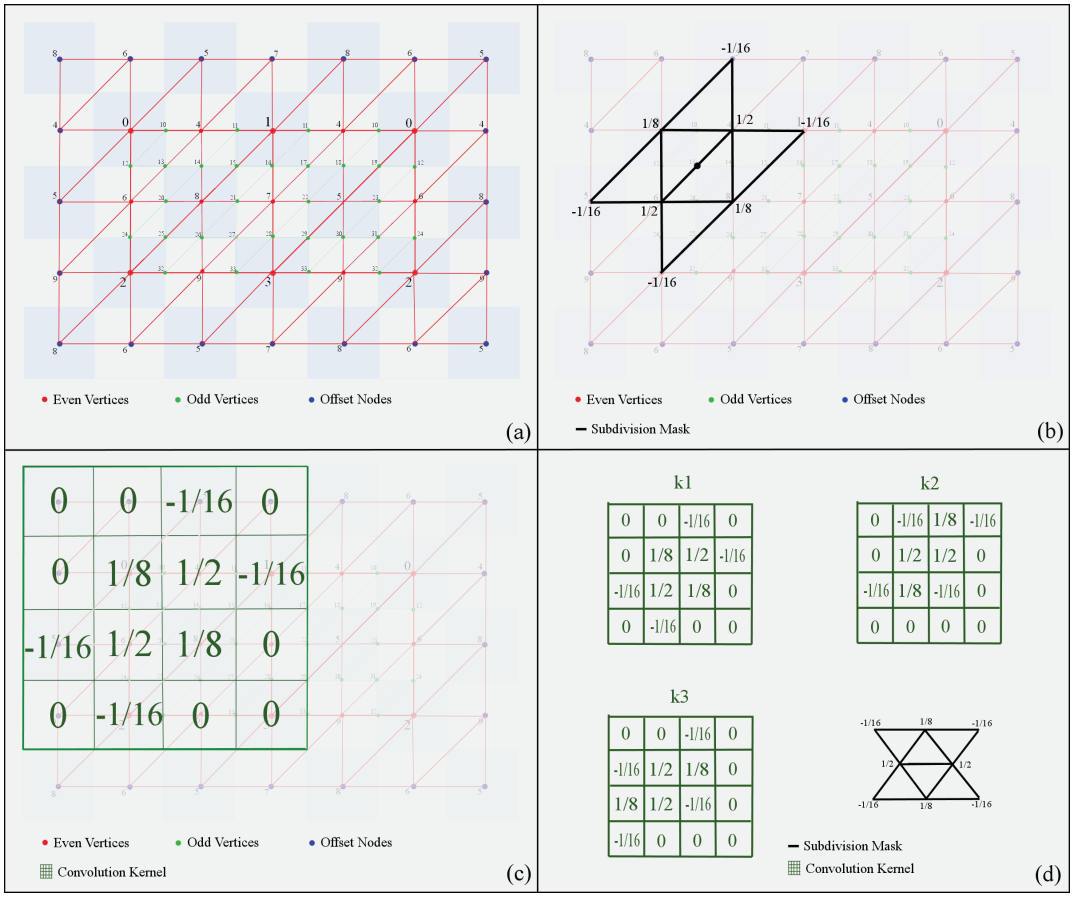

Figure 4.12: (a) Modified 2D Grid Structure. (b) Application of mask for interior odd vertices with regular neighbors. (c) Equivalent convolution kernel. (d) Three convolution kernels generated for three edges.

With a 2D grid representation and a mask with constant coefficients, odd vertices can be generated by consecutive convolutions with three kernels, which are created by rotating the subdivision mask three times (Figure 4.12.d). Necessity for the grid offsets arises from the application of the mask to the grid boundaries, and modified subdivision scheme requires first neighbors of even vertices that are next to generated vertex. Offset width does not change according to the grid dimensions and time required for the update of the offsets is negligible. After three times convolutions of the n^{th} level subdivision surface, resulting 2D grids are merged to generate $n+1^{th}$

level subdivision surface having $(2^{n+1} + 1) \times (2^{n+2} + 1)$ nodes (Figure 4.13).

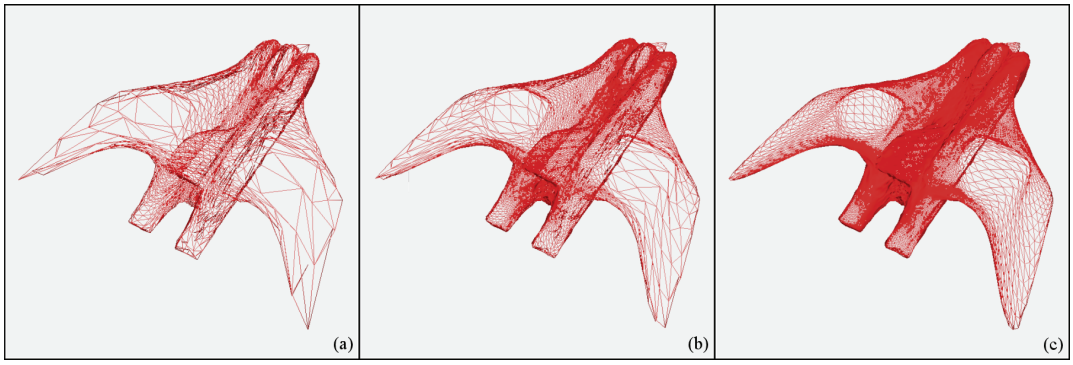

Figure 4.13: Comparison of resulting mesh refined by subdivision and rendered at different level of details, (a) 129x65=8335 vertices (b) 257x129=33153 vertices (c) 512x257=131841 vertices.

4.3. Physical Model

Simulation of deformable objects is an extensive research area, where several methods are present, varying from fast and simple methods favoring speed and scalability, to much more complex methods favoring accuracy and stability. Linear methods such as mass-spring models for dynamic deformations are suitable for use in real-time applications; however, they are not capable of handling large deformations and small time steps are required to guarantee stability. On the other hand, non-linear models incorporating large viscoelastic and plastic deformations are computationally intensive, and despite their physical accuracy, real-time simulation of large deformations is only possible with massively parallel computers.

For the demonstration of the deformable object on a collaborative virtual environment, we use a real-time physical simulation of a uniform-tension-membrane, based on linear finite-elements. We introduced finite element discretization to form the global stiffness matrix, which is updated frequently to handle large deformations with enhanced accuracy and we used Runge-Kutta-Fehlberg method for integration to achieve bigger time steps and improved stability.

4.3.1. Linear Finite-Element Model

Application of the finite-element method for the wave equation [35, 36], describing the time-dependent small deformations of a uniform-tension membrane results in a standard system of equations [37]:

$$M\ddot{x} = -B\dot{x} - Kx + f_{external}, \tag{4.8}$$

where, x is the normal deformation of each node, M is the diagonal mass matrix, $f_{external}$ is the external force vector due to user interactions, B is the diagonal damping matrix, and K is the stiffness matrix. In our implementation, we separate normal deformation and the velocity of each node to improve the stability of the Runge-Kutta method used to solve the linear system. Namely, we have

$$\dot{x} = v, \tag{4.9}$$

and the resulting equation of motion:

$$M\dot{v} = -Bv - Kx + f_{external} \tag{4.10}$$

The finite element method works well with an arbitrary triangulation of a surface as well as a proposed regular grid structure. In our implementation we apply the damping matrix directly on the nodal velocities, so as to model a permeable membrane placed in a fluid. In some standard formulations, the damping is applied to relative nodal velocities. The two yields in similar solutions, however our implementation results in simpler sparse structures and faster simulation times via improved stability of nodal damping.

4.3.2. Stiffness Matrix Generation

Finite Element Method is defined as a powerful numerical procedure, where a body is subdivided into a discrete number of finite elements [38]. According to our mesh representation and subdivision methodology, each group of vertices forming a triangle on the mesh is taken as an element. Global stiffness matrix K is assembled

using the element stiffness matrices K^e, which identifies the relation between nodal displacements and the force exerted on the element nodes maintaining the linear elasticity model [39, 40].

$$K=\sum_e K^e, \tag{4.11}$$

$$K^e=\begin{bmatrix} k_1^e+k_2^e & -k_1^e & -k_2^e \\ -k_2^e & k_2^e+k_3^e & -k_3^e \\ -k_1^e & -k_3^e & k_1^e+k_3^e \end{bmatrix}, \tag{4.12}$$

Members k_i^e of the element stiffness matrix are derived using the material properties of the object (Appendix B).

4.3.3. Handling Boundary Conditions and Domain Decomposition

In the finite element methodology, there are two classes of boundary conditions. *The essential boundary* conditions are also called *geometric boundary conditions* and correspond to prescribed displacements and rotations, while *the natural boundary conditions* or also called *force boundary conditions* correspond to prescribed forces and moments [35]. The natural boundary conditions are introduced to the system as external forces in the equation 4.10; however, handling the essential boundary conditions is not straightforward.

In our approach, essential boundary conditions arise from two circumstances. First, the membrane model should have some of its nodes with fixed displacements in the simulated environment. A membrane model having no geometric boundary conditions will float like a rigid body and it is impossible to solve the system in eq. 4.10, since the global stiffness matrix K is singular. Second, we are aiming to distribute computational load of the simulation among clients, which requires partitioning of the membrane domain between participants. Every client is responsible for solving the system in its local domain and local solutions stay consistent by distributing the states of the nodes on the domain boundaries.

Simplest method used for the application of essential boundary conditions is

to remove i[th] row and column of the stiffness matrix K to introduce i[th] node as a fixed node. Consequently, i[th] equation in the system that we are trying to solve will be deleted. After unfolding the system in eq. 4.10,

$$\begin{aligned} &K_{11}x_1 + K_{12}x_2 + \ldots + K_{1n}x_n = M_{11}\dot{v}_1 - B_{11}v_1 - f_1 \\ &K_{21}x_1 + K_{22}x_2 + \ldots + K_{2n}x_n = M_{21}\dot{v}_2 - B_{21}v_2 - f_2 \\ &\ldots. \\ &K_{n1}x_1 + K_{n2}x_2 + \ldots + K_{nn}x_n = M_{nn}\dot{v}_n - B_{nn}v_n - f_n \end{aligned} \tag{4.13}$$

It's obvious that the balances of the equations are lost, because i[th] column is also removed on the left hand side of the system. The terms $-K_{ji}x_i$ should also be added to the right hand side before attempting to solve the system. This method is computationally intensive and requires significant modification of global stiffness matrix K.

Payne and Irons method introduces an alternative technique for maintaining the essential boundary conditions [41]. Instead of deleting the i[th] row and column, a very large number α added to diagonal element of the stiffness matrix K_{ii} and other elements of the i[th] row is set to 0. Also, left hand side of the i[th] equitation $M_{ii}\dot{v}_i - B_{ii}v_i - f_i$ is replaced with $\alpha\bar{x}_i$, so that the i[th] equitation becomes $\alpha x_i = \alpha\bar{x}_i$, and the system in eq. 4.18 remains symmetric and solvable.

Payne and Irons method is a feasible candidate for our approach to handle essential boundary conditions. Since the vertices of the regular mesh structures have valances of 6, there are 6 non-zero elements in the each row of the stiffness matrix. We have a sparse matrix structure that bounds the complexity of matrix operations with the order of non-zero element count instead of matrix dimensions. Each essential boundary condition reduces the required computation by eliminating 5 non-zero elements out of 6 from the stiffness matrix.

4.4. Network Model

In our approach, we build a simple but effective network model, which is

capable of satisfying the needs of a small scale collaborative virtual environment and a distributed physical simulation among participants.

4.4.1. Network Architecture

Our approach is implemented with a peer-to-peer architecture working on local and wide area networks. We don't introduce any dedicated server, and peer nodes are functioning as both clients and servers. Every peer should have a listening port and address which can be used by other peers by directly specifying connection request through the command listener. User Datagram Protocol (UDP) used for the communication, since the speed and bandwidth requirements are essential for a real time simulation and have a greater priority over packet integrity because proposed synchronization model can handle packet loss to a certain degree.

Peers can run on different computers on the network or can be started in the same application as separate threads sharing the command listener and rendering windows. Handshaking protocol is also simple. As the main purpose of this thesis is to propose a method for the simulation of deformable objects over network, while distributing the computational load, there is no security measure and the peer requesting the connection is always accepted by the listener. They add each other to the list of connected peers. In the proposed method, every peer has an individual virtual environment before the establishment of a connection. The goal is to synchronize these virtual environments and to enable interaction over deformable objects that will lead to collaboration among participants.

Although the peer-to-peer model is accepted, there is one condition, where one of the peers is acting as a server. While determining the domains on the mesh to distribute the simulation, the environment in which the deformable object is first created acts as a master and decides the partitioning. Partitioning occurs after sending a request by slave which selects a face and identifies it as the point of interest where the peer is going to introduce an external force.

Figure 4.14 demonstrates the third-party case of distributed simulation with one deformable object. Initially (a), there exists two virtual environments and one has an object while the other one is empty. One of the environments sends a connection request and both environments add each other to connected environments list. Environments having objects send object description messages (b) after the connection establishment. This message contains a unique object identifier and object subdivision level. After sending the object description, they immediately start sending object state information. Object state information includes the recent state of the deformed object and will be described in detail at the next section. At this stage, second peer renders the object in the first virtual environment without any calculation and interaction. If the user decides to interact with an object by applying a force on any face, send this face id as point of interest. (c) Partitioning algorithm runs on both clients, however object owner is the authority and manages the partitioning among the connected peers. After the partitioning, connected peers start to broadcast state information belonging to their domain, and receive state information of other domains of the object. If a third pair (d) sends a connection request to any of the connected peers, its request is accepted and it receives peers list connected to this virtual environment to establish connection to them immediately. (e) Object owner sends the deformable object description and state to the connected peer and second peer starts broadcasting state belonging to its domain. If the third party receives state information from second peer before receiving object description, simply ignores this package since, the default behavior for receiving irrelevant packages is ignoring. (f) Third part may chose to observe the deformation or send a point of interest describing face id where the interaction is requested. Object owner notifies second peer and domains are re-partitioned to allow third party interaction. Finally, (e) every peer has a domain and broadcasting state of this domain, while receiving the remaining states from connected peers. If any of the peers except for the object owner disconnects from the system, object owner handles the re-partitioning. If the object owner quits, simulation for this object is ended.

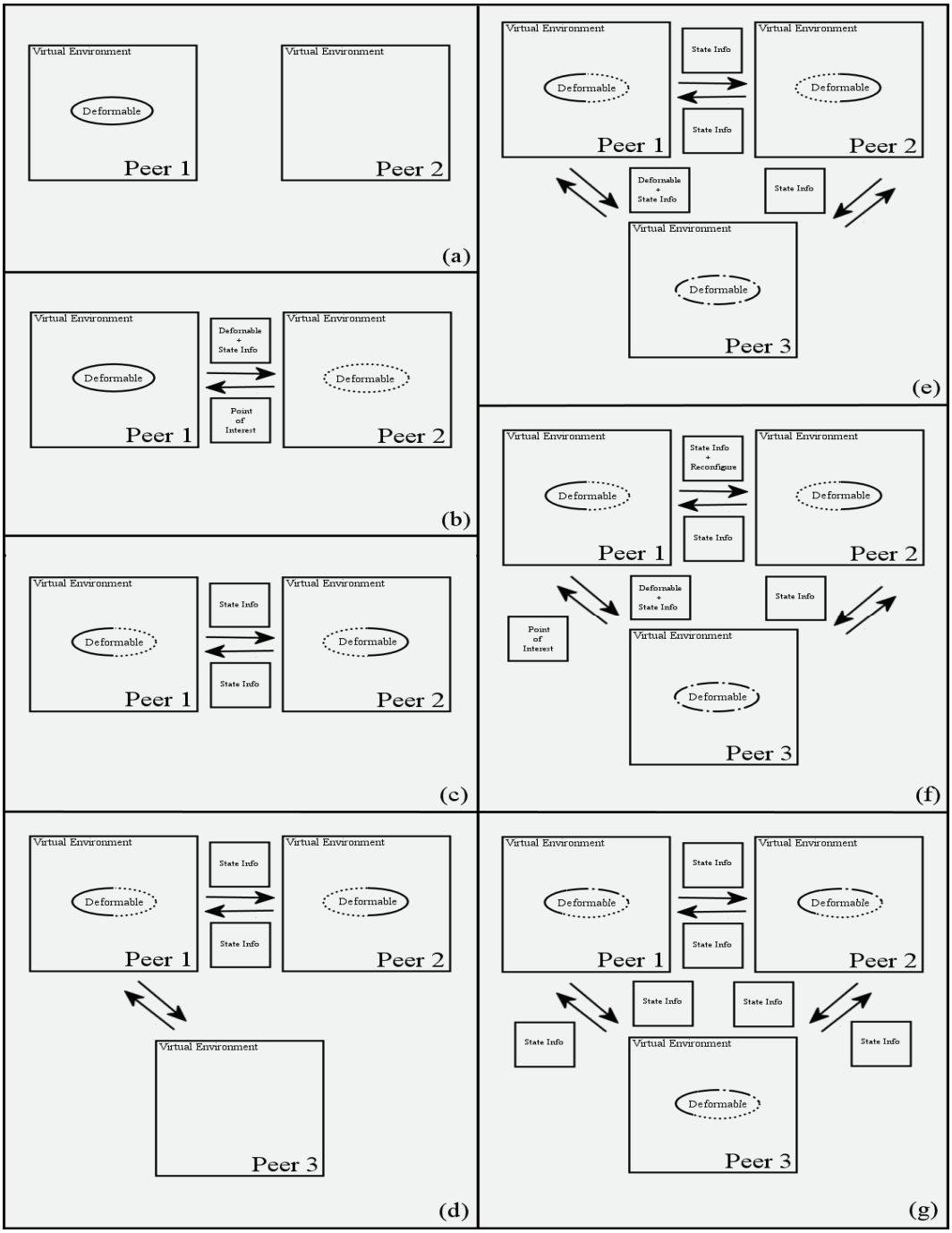

Figure 4.14: Network Protocol: (a) Individual peers having separate VE's. (b) Connected peers, fist peer sending object description and state info, second peer specifying point of interest. (c) Synchronized peers after domain division. (d) Third peer is introduced. (e) Third peer is receiving object description and state info from first peer, also introduced to second peer by first peer. (f) Third peer is specifying point of interest, first pair performs domain division. (e) Synchronized peers after domain division.

4.5. Partitioning and Synchronization of Physical and Geometric Models through the Network

In our approach, partitioning the deformable object and synchronizing among peers is an important issue, since it enables collaboration in the virtual environments with distributed computational load. For the purpose of efficient communication and separation we introduce a quad tree based data structure (Figure 4.15) over 2D grid structure proposed on the previous sections.

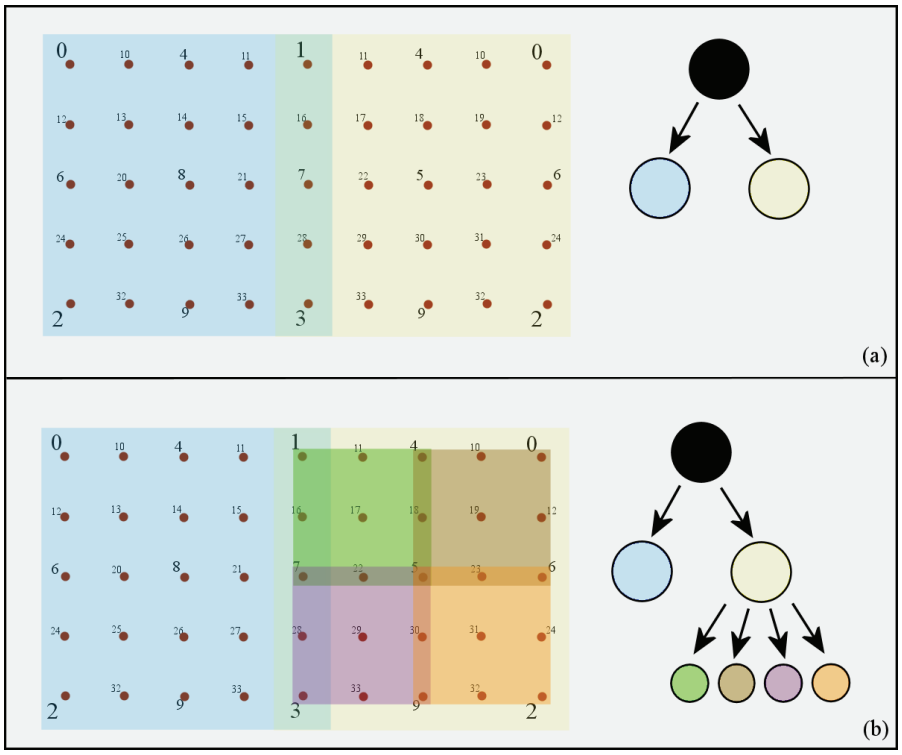

Figure 4.15: (a) Minimal tree structure for tetrahedral domain.
(b) Sample tree structure having depth of two.

This structure (Figure 4.14) is a natural formation for the tetrahedral domain because it has $(2^n + 1)$ x $(2^{n+1} + 1)$ nodes, and 2^n x 2^{n+1} faces that can be divided hierarchically. Tree nodes can be transferred efficiently via network since a tree node contains a range identifier which is actually the combination of upper left and lower right node index numbers, and state information of corresponding region as a 2D array. Minimum depth level for the tree can be adjusted to keep the packaged tree node size smaller then the maximum packet size allowed by the network protocol. It

may look like the boundaries of node regions that are included on more than one tree node occupies wasted space as they might be transferred more than one time via network on the different packages, but the order of wasted space is 4n where the package size is n^2, and this amount is negligible for high resolution meshes, having a tree divided up to a certain level. Also, including boundaries while division has a special importance in physical simulation where the domains are divided in the same manner and states of nodes at the domain boundaries must updated regularly to synchronize the simulations as stated in the previous sections.

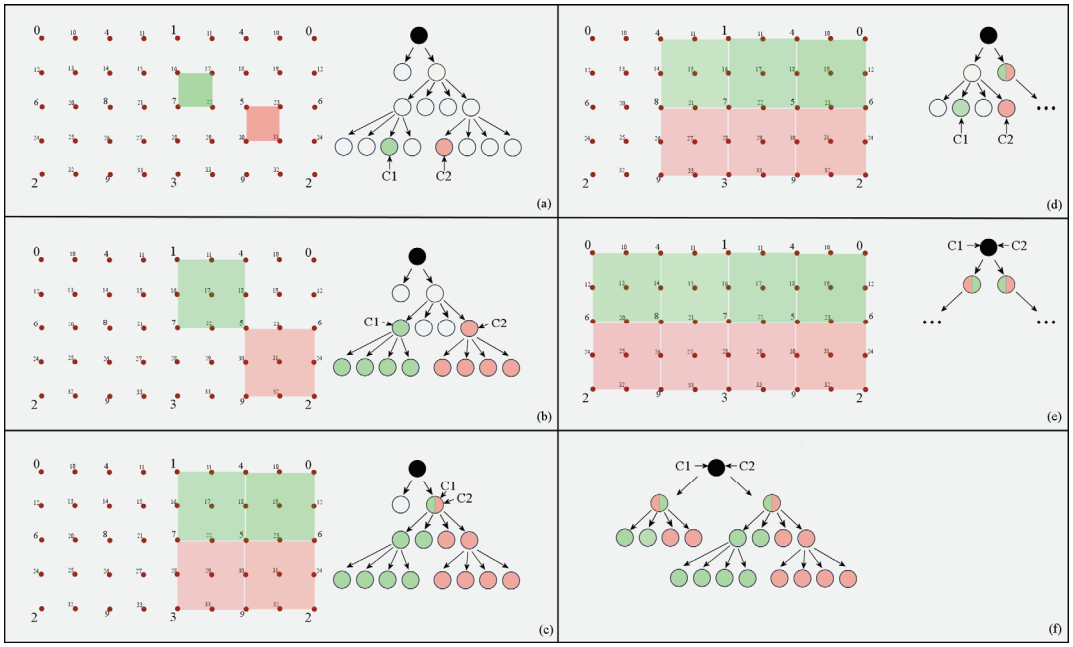

Figure 4.16: Demonstration of the partitioning algorithm.

Domain divisions are designed upon this tree structure. While dividing the domain into sub-domains, equivalence of the sets is an important criterion, however, keeping the domain boundaries shorter for an accurate synchronization for the physical simulation and partitioning minimal for an efficient network transmission are essential.

```
for each peer
    peer->current_node = root
    while peer->current_node is expandible
        expand peer->current_node
        for each peer->current_node ->child  AND peer->current_node ->child
covers selected face
            peer->current_node =  peer->current_node ->child
        end for
    end while
end for

for each peer_area = 1
while the grid is not full
   while peer[all]current_node != root
      current_peer = minArea(all_peers)
      if current_peer->siblings are empty,current_peer->siblings=current_peer->ID
      else closestEmpty(current_peer->siblings) =current_peer->ID
      update(peer_area)
      current_node = current_node->parent
   end while
   if  root->next not expanded
        expand next
        for each peer->current_node =  closest(next->child)
    end if
end while
```

Table 4.1: The pseudo code of the partitioning algorithm.

This algorithm (Table 4.1) works well for the two parties, partitions the domain equally on the average case, and with a ratio of 7/9 on the worst case. For the third party case, sub domain areas are still very close to equal, however, partitioning occurs if the selected facets are close to each other. Investigation of n-party case of the algorithm is left for the future work; however, choosing close facets is not very probable in such a crowded environment and it will provide a reasonable partitioning.

Peers traverse the proposed tree structure and broadcast the state information of the associated nodes after each simulation step. Every peer simulates the nodes of local domain at time ‘t’ using the position and velocity values of the nodes on boundaries at time ‘t-1’. There is no guarantee that the packages arrive on time, but system can handle network lag or packet loss for a certain degree depending on the

simulation parameters. After a certain degree it might become a stability issue and simulation can fail completely. Quantification of error and stability rates is performed on the next chapter.

5. RESULTS & CONCLUSION

5.1. Graphical Result & Performance

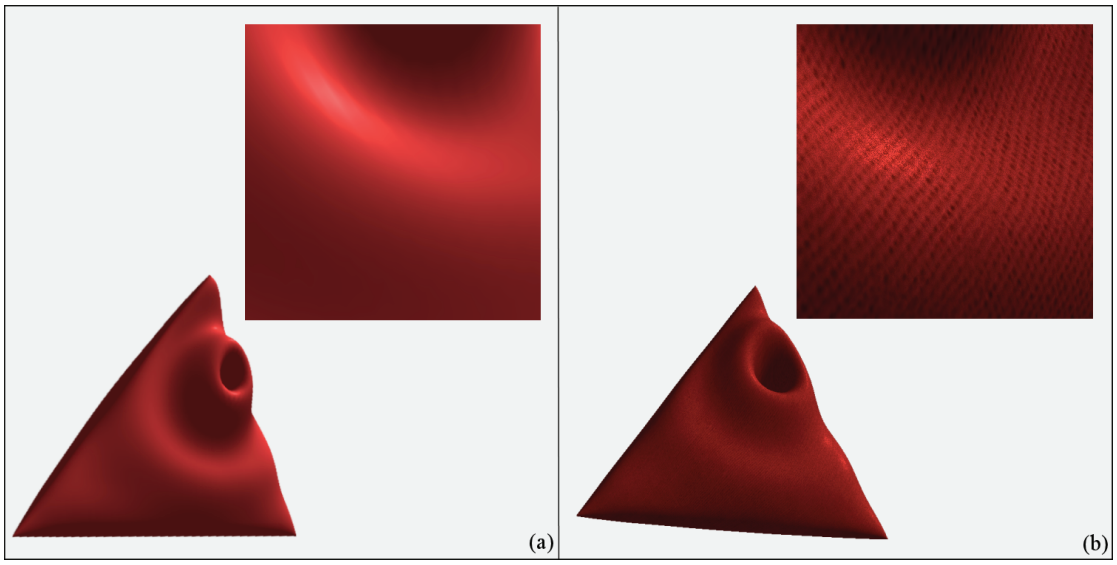

Figure 5.1: (a) Surface rendered using the Phong shading model; (b) Surface rendered using the Phong shading model, and a cloth texture.

Our graphical sub-system can handle very large meshes efficiently, taking advantage of regular-mesh and subdivision methodology as presented in the previous chapter. The system renders meshes using the Phong shading model (Figure 5.1) at interactive frame rates (20 fps) with resolution up to twelve thousand polygons on Intel Dual Xeon 3.4 GHz PC equipped with NVIDIA Quadro FX3400 GPU (Table 1). We implemented Phong shading model on the GPU (Appendix C). Vertex positions are uploaded to texture memory and vertex normals are computed on the fly using texture lookups.

Detail level (n)	# of nodes	# of polygons	Fps.
4	33 X 17	1024	66.67
5 (half res.)	33 X 33	2048	66.67

5	65 X 33	4096	66.67
6 (half res.)	65 X 65	8192	62.25
6	129 X 65	16384	21.27
7 (half res.)	129 X 129	32768	12.10
7	257 X 129	65536	5.81

Table 5.1: Rendering Performance Evaluation.

5.2. Evaluation of the Physical Simulation Environment

We tested the performance our physical deformation engine with the configuration above, and evaluated effects of physical parameters on the stability of the simulation (Figure 5.2). Our solver is using Runge-Kutta Fehlberg method for integration and it allows taking moderately large time steps up to a certain stiffness level defined for the simulated material. Stiff objects require small time steps to maintain the stability of the system.

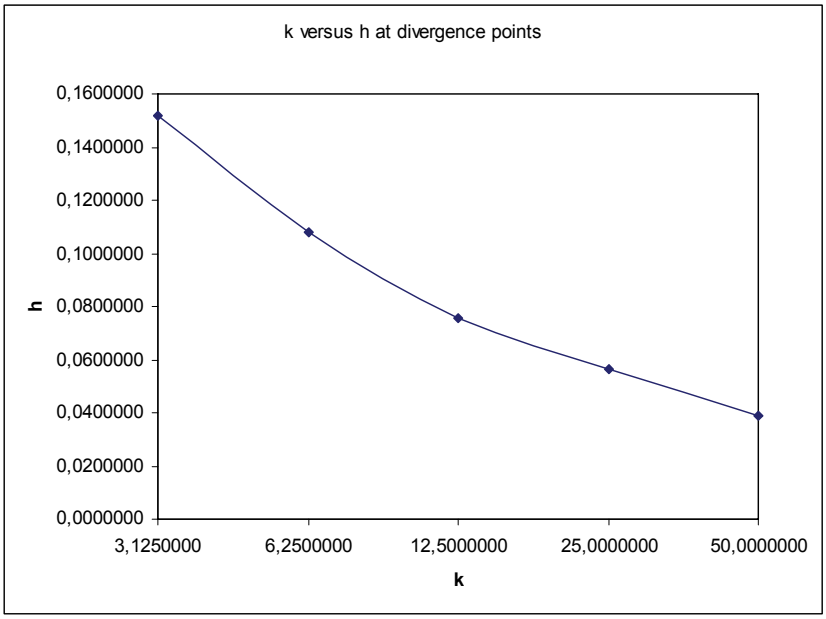

Figure 5.2: Stiffness parameter k (N/m) versus time step h(s), at the divergence points, where the simulation losts stability.

We also tested the performance of the system by comparing computational load and number of simulated nodes. Our deformation engine can handle multi-

resolution meshes up to 33153 vertices, and maintains interactivity at less than %30 CPU utilization.

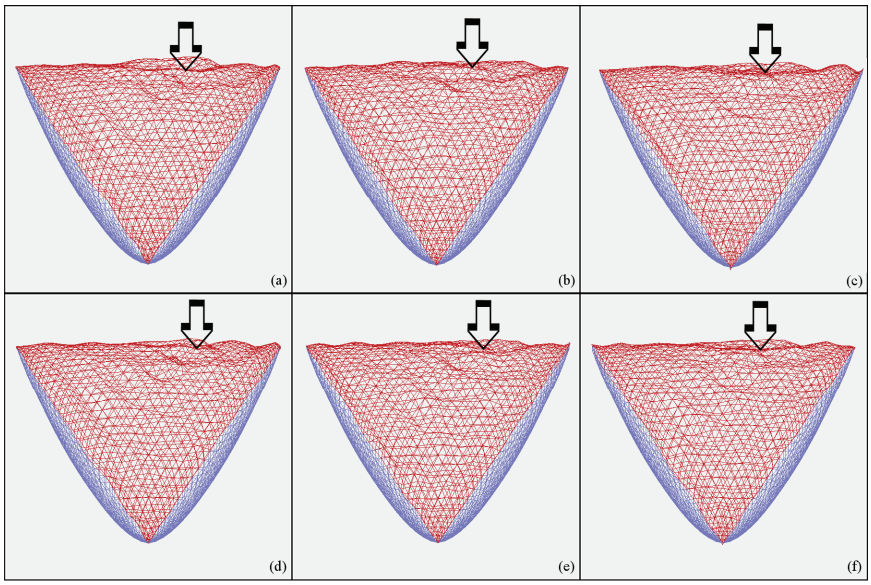

Figure 5.3: Deformation on the tetrahedral domain with wavy surface parameters, applying sinusoidal forces with frequency f on the selected faces, colors represents peers and k=1.0 N/m, b=0.1, f=2.4 N.

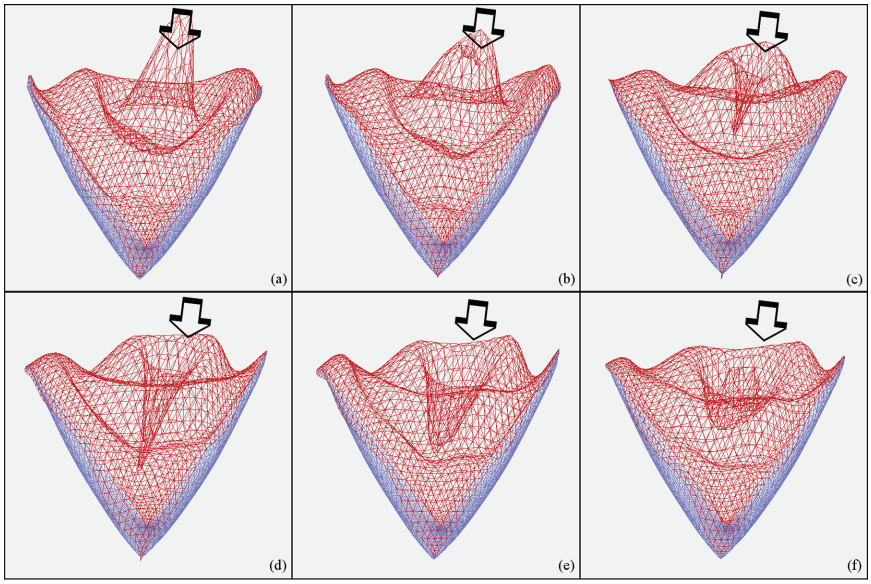

Figure 5.4: Large deformation on the tetrahedral domain, applying sinusoidal forces on the selected faces, colors represents peers k=6.25 N/m b=0.1 f=1.2 N.

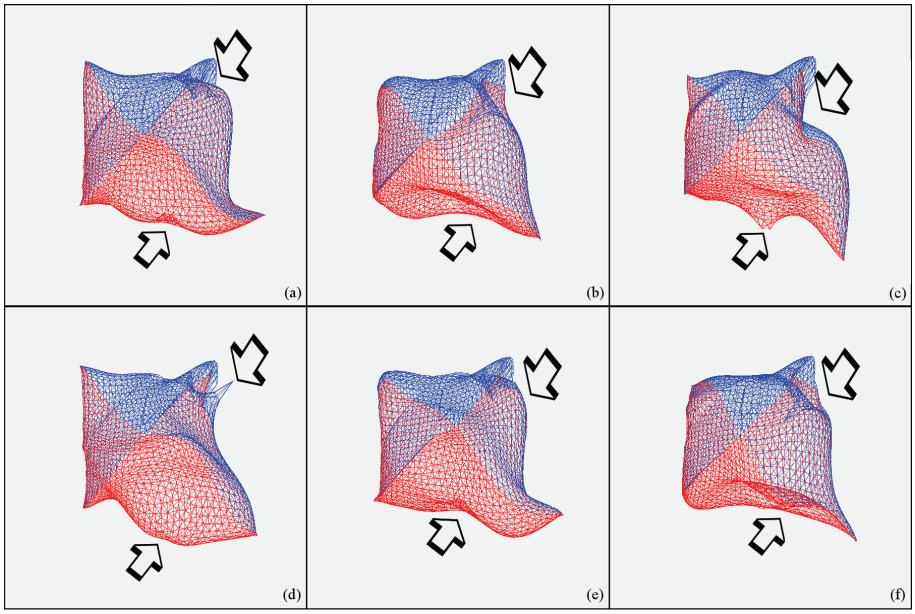

Figure 5.5: Demonstration of 2-party large deformation on the tetrahedral domain, applying sinusoidal forces on the selected faces with frequency f, colors represents peers and k=6.25 (N/m), b=0.1, f=1.2 N.

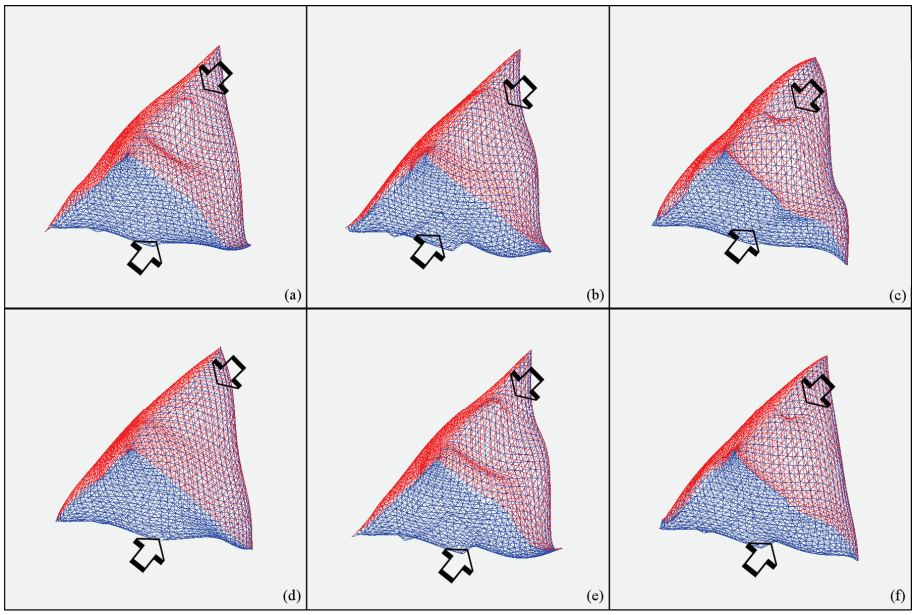

Figure 5.6: : Demonstration of 2-party deformation on the tetrahedral domain with wavy surface parameters, applying sinusoidal forces with frequency f on the selected faces, colors represents peers and k=1.0 (N/m), b=0.1, f=2.4 N.

5.3. Evaluation of the Network Performance

Our network communication model can handle synchronous simulation among two peers up to a surface with 8385 vertices at the local area network. This level has a bandwidth requirement of 10.75 M Bits per second without any compression. Less detailed surfaces are much more convenient for efficient communication requiring less bandwidth. As a future work, we also consider on the fly compression which might significantly reduce the bandwidth requirement but also can increase the computational load. Optimization of the system for the Internet is out of scope of this paper. It is safe to predict that the network lag on public networks will have an impact on performance.

Detail level (n)	# of nodes	Bandwidth
4	33 X 17	735 KBits/sec
5	65 X 33	2.75 MBits/sec
6	129 X 65	10.75 MBits/sec
7	257 X 129	42.49 MBits/sec

Table 5.2: Network Bandwidth Requirements

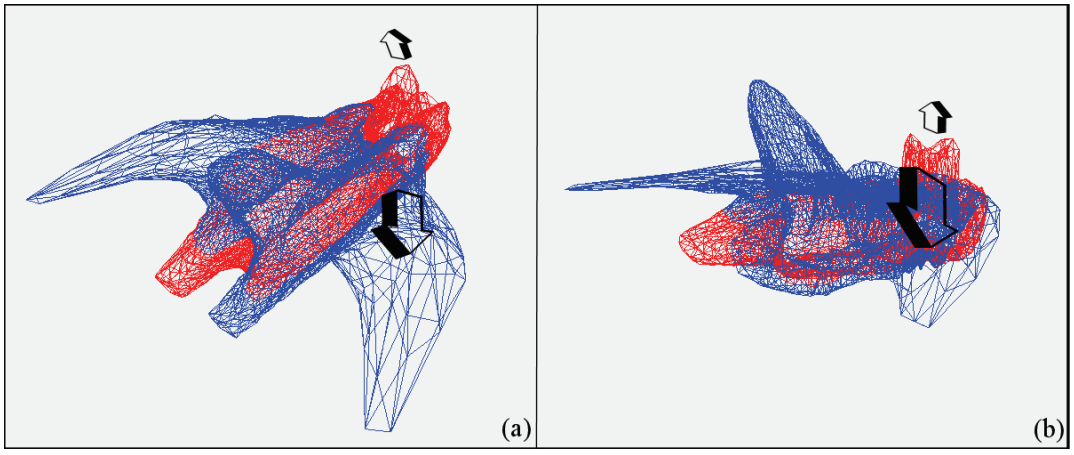

Figure 5.7: Demonstration of 2-party deformation on the regular mesh applying sinusoidal forces with frequency f on the selected faces, colors represents peers and k=1.0 (N/m) b=0.05 f=1.2 N.

5.4. Conclusion & Future Work

We have proposed a new technique for deformable body simulations in the field of collaborative virtual environments and also introduced several improvements over methods that we adopted. Our approach enables real-time simulation of deformable objects in the collaborative environments, and results for two party cases are successful. We found that adaptive refinement and multilevel meshing strategies is a open ended research subject that can be further exploited for increasing network efficiency, and better physical accuracy for CVE's.

Our method needs to be tested and optimized in the Internet over long physical distances for network effects. It needs to be investigated in detail and the system must be tested properly to ensure its robustness and efficiency. The techniques implemented are coming from various research areas including computational geometry, physical based modeling and computer networking, as well as computer graphics.

REFERENCES

1. Churchill, E.F. and D. Snowdon (1998) *CVE'98: collaborative virtual environments 1998*. ACM SIGGROUP Bulletin **Volume**, 21-22
2. Churchill, E.F., D.N. Snowdon, and A.J. Munro, *Collaborative virtual environments : digital places and spaces for interaction*. Computer supported cooperative work. 2001, London ; New York: Springer. xx, 316 p.
3. Dourish, P. and V. Bellotti, *Awareness and coordination in shared workspaces*, in *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*. 1992, ACM Press: Toronto, Ontario, Canada.
4. Singhal, S. and M. Zyda, *Networked virtual environments : design and implementation*. SIGGRAPH series. 1999, Reading, MA: Addison-Wesley. xvi, 331 p.
5. Stytz, M.R., *Distributed virtual environments.* Computer Graphics and Applications, IEEE, 1996. **16**(3): p. 19-31.
6. Hagsand, O., *Interactive multiuser VEs in the DIVE system.* Multimedia, IEEE, 1996. **3**(1): p. 30-39.
7. Macedonia, M.R., et al., *NPSNET- A network software architecture for large-scale virtual environments.* Presence: Teleoperators and Virtual Environments, 1994. **3**(4): p. 265-287.
8. Benford, S., et al., *Collaborative virtual environments.* Commun. ACM, 2001. **44**(7): p. 79-85.
9. Jorissen, P. and Z.M.W.L. Maarten Wijnants, *Dynamic Interactions in Physically Realistic Collaborative Virtual Environments.* IEEE Transactions on Visualization and Computer Graphics %@ 1077-2626, 2005. **11**(6): p. 649-660.
10. Dequidt, J., L. Grisoni, and C. Chaillou, *Collaborative interactive physical simulation*, in *Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia %@ 1-59593-201-1*. 2005, ACM Press: Dunedin, New Zealand. p. 147-150.
11. Xiaojun, S., et al. *A heterogeneous scalable architecture for collaborative haptics environments*. 2003.
12. Zhou, J., X. Shen, and N.D. Georganas. *Haptic tele-surgery simulation*. 2004.
13. Goncharenko, I., et al. *Cooperative control with haptic visualization in shared virtual environments*. 2004.
14. Sederberg, T.W. and S.R. Parry, *Free-form deformation of solid geometric models*, in *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*. 1986, ACM Press.
15. Terzopoulos, D., et al., *Elastically deformable models*, in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. 1987, ACM Press.
16. Baraff, D. and A. Witkin, *Large steps in cloth simulation*, in *Proceedings of*

the 25th annual conference on Computer graphics and interactive techniques. 1998, ACM Press.
17. Desbrun, M., Peter Schröder, and A. Barr, *Interactive animation of structured deformable objects*, in *Proceedings of the 1999 conference on Graphics interface '99*. 1999, Morgan Kaufmann Publishers Inc.: Kingston, Ontario, Canada.
18. James, D.L. and D.K. Pai, *ArtDefo: accurate real time deformable objects*, in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. 1999, ACM Press/Addison-Wesley Publishing Co.
19. Kang, Y.-M. and H.-G. Cho, *Complex deformable objects in virtual reality*, in *Proceedings of the ACM symposium on Virtual reality software and technology*. 2002, ACM Press: Hong Kong, China.
20. Choi, K.-J. and H.-S. Ko, *Stable but responsive cloth*, in *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. 2002, ACM Press: San Antonio, Texas.
21. James, D.L. and K. Fatahalian, *Precomputing interactive dynamic deformable scenes*. 2003, ACM Press. p. 879-887.
22. Choi, K.-S., et al., *Deformable simulation using force propagation model with finite element optimization.* Computers & Graphics, 2004. **28**(4): p. 559-568.
23. Baraff, D. and A. Witkin, *Physically based modeling*, in *Proceedings of the conference on SIGGRAPH 2003 course notes*. 2003, ACM Press: Los Angeles, CA.
24. Press, W.H., *Numerical recipes in C++ : the art of scientific computing*. 2nd ed. 2002, Cambridge [England] ; New York: Cambridge University Press. xxviii, 1002 p.
25. Wikipedia, c. *Genus (mathematics)*. 22 July 2006 08:39 UTC [cited 1 August 2006 17:57 UTC]; Available from: http://en.wikipedia.org/w/index.php?title=Genus_%28mathematics%29&oldid=65180879
26. Wikipedia, c. *Euler characteristic*. 29 July 2006 13:32 UTC [cited 2 August 2006 14:51 UTC]; Available from: http://en.wikipedia.org/w/index.php?title=Euler_characteristic&oldid=66521171
27. Gamelin, T.W. and R.E. Greene, *Introduction to topology*. 2nd ed. Dover books on mathematics. 1999, Mineola, N.Y.: Dover Publications. xii, 234 p.
28. Praun, E. and H. Hoppe, *Spherical parametrization and remeshing.* 2003. **22**(3): p. 340-349.
29. Sander, P.V., et al., *Texture mapping progressive meshes*, in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 2001, ACM Press.
30. Fruchterman, T. and E. Reingold, *Graph Drawing by Force-directed Placement.* Software - Practice and Experience, 1991. **21**(11): p. 1129-1164.
31. Alexa, M., *Recent Advances in Mesh Morphing*. 2002, Blackwell Synergy. p. 173-196.
32. David, L., et al., *GPGPU: general purpose computation on graphics hardware*, in *Proceedings of the conference on SIGGRAPH 2004 course notes*. 2004, ACM Press: Los Angeles, CA.
33. *2000 SIGGRAPH Full Day Course: Subdivision for Modeling and Animation*.
34. Zorin, D., Peter Schröder, and W. Sweldens, *Interpolating Subdivision for*

meshes with arbitrary topology, in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 1996, ACM Press.
35. Bathe, K.-J. and K.-J. Bathe, *Finite element procedures*. 1996, Englewood Cliffs, N.J.: Prentice Hall. xiv, 1037 p.
36. Reddy, J.N., *An introduction to nonlinear finite element analysis*. 2004, Oxford ; New York: Oxford University Press. xv, 463 p.
37. Hughes, T.J.R., *The finite element method : linear static and dynamic finite element analysis*. 1987, Englewood Cliffs, N.J.: Prentice-Hall. xxvii, 803 p.
38. Popov, E.P. and T.A. Balan, *Engineering mechanics of solids*. 2nd ed. 1999, Upper Saddle River, N.J.: Prentice Hall. xvi, 864 p.
39. Atanackovic, T.M. and A. Guran, *Theory of elasticity for scientists and engineers*. 2000, Boston: Birkhäuser. xii, 374 p.
40. Landau, L.D., et al., *Theory of elasticity*. 3rd English ed. 1986, Oxford [Oxfordshire] ; New York: Pergamon Press. viii, 187 p.
41. Zienkiewicz, O.C. and R.L. Taylor, *The finite element method*. 5th ed. 2000, Oxford ; Boston: Butterworth-Heinemann.

Appendix A

General and Sparse Matrix Classes

```
class matrix {
public:

        matrix(): row(0), clmn(0), mtrx(NULL){};
        matrix(int rows, int cols);
        matrix(const matrix& rhs);
        matrix(const matrix& Rotation , const triple & Position);
        ~matrix();
        matrix & setIdentity();
        matrix & scale(double c);
        matrix inverse() const;
        const matrix &operator=(const matrix &rhs);
        const matrix &operator=(const quaternion &rhs);
        matrix  operator* (const matrix& x) const;
        vector  operator* (const vector& x) const;
        matrix  operator- (const matrix& x) const;
        double &operator[](int rhs) const;
        double & operator()(int row, int clmn ) const;
        quaternion toQuaternion() const;
        matrix transpose() const;
        matrix  conv(const matrix& x) const;
        matrix  conv(const matrix& x, const matrix& z, int cell = 0, int xOffset = 0, int
yOffset = 0) const;
        matrix  copy(const matrix& z, int cell=0, int xOffset=0, int yOffset=0) const;
        void saveBin(FILE * output);
        void loadBin(FILE * input);
        const matrix & print() const;
        void DumpFile();
        double average();
protected:
        double*         mtrx;
        int             row;
        int             clmn;
        friend class matrix_ssd;
};
```

```
class matrix_ssd
{
public:
        matrix_ssd(): row(0), clmn(0), idiag(NULL), ndiag(0), val(NULL), lval(0){};
        matrix_ssd(int rows, int cols): row(rows), clmn(cols), idiag(NULL), ndiag(0),
val(NULL), lval(rows){};
        matrix_ssd(const matrix& rhs);
        const matrix_ssd &operator=(const matrix_ssd &rhs);
        double & operator()(int clmn, int row);
        vector  operator* (vector x) const;
        void print() const;
protected:
        int             row;
        int             clmn;
        int *idiag;
        int ndiag;
        double *val;
        int lval;
};
```

```
class matrix_coord
{
public:

        matrix_coord(int rows = 0, int cols = 0);
        const matrix_coord &operator=(const matrix_coord &rhs);
        double & operator()(int row, int clmn);
        vector  operator* (const vector &x);
        void syncronise();
        void print();
        void DumpFile();
protected:
        int             row;
        int             clmn;
        std::vector<coord_element> values;
        int sorted_end;
        double *val;
        int *rows;
        int *cols;
        int capacity;
        bool sync;
};

struct coord_element
{
        coord_element(int *row = NULL, int *clmn = NULL, double *value =
NULL):row(row),clmn(clmn),value(value){};
        bool operator== (const coord_element &rhs) const { return *row == *rhs.row
&& *clmn == *rhs.clmn; } ;
        bool operator!= (const coord_element &rhs) const { return *row != *rhs.row ||
*clmn != *rhs.clmn; };
        bool operator< (const coord_element &rhs) const { return (*row == *rhs.row ?
*clmn < *rhs.clmn : *row < *rhs.row);};
        double *value;
        int *row;
        int *clmn;
};
```

Appendix B

Theoretical Background of the Membrane Model

Consider an infinitesimal circular membrane element of radius r, and thickness of δ, subject to a uniform tension of σ [$p_a = {}^{N}\!/_{m^2}$]. When a distributed load is applied to the membrane in the normal direction, the element will deflect in the normal direction according to the figure B.1.a.

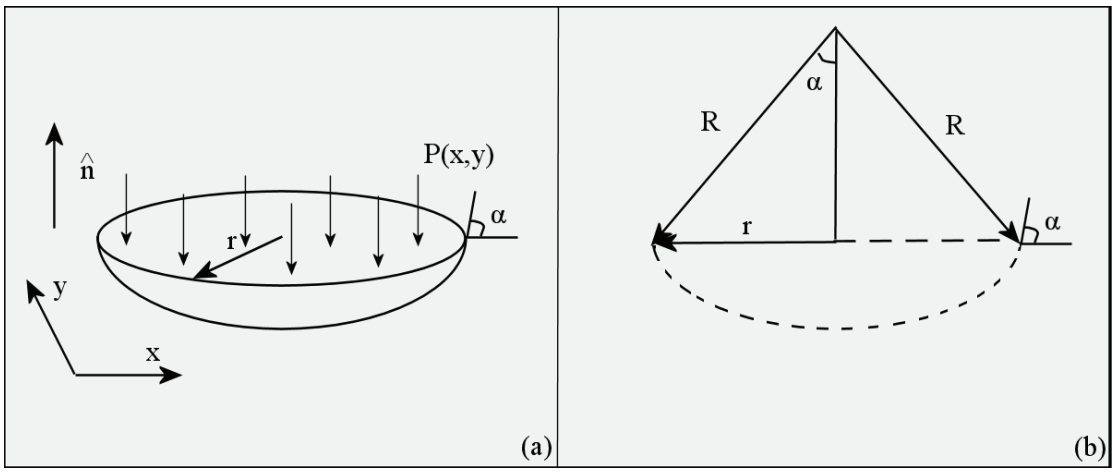

Figure B.0.1: Uniform tension membrane model.

The force balance in the normal direction yields:

$$p.(\pi r^2) = \sigma.\delta.(2\pi r).\sin\alpha \tag{B.1}$$

$$p = 2.\sigma.\delta.(\frac{1}{r}).\sin\alpha \tag{B.2}$$

So, that the deformation of the membrane element is locally spherical and as shown in the figure B.1.b. Then we have:

$$\sin\alpha = r/R \tag{B.3}$$

Hence the equation B.2 yields:

$$p = \delta.\sigma.(\frac{2}{r}) \tag{B.4}$$

Since the curvature in the x-direction is equal to the curvature in the y-direction for the spherical deformation under the uniform load (local),

$$R_x = R_y = R\,, \tag{B.5}$$

Equation B.4 becomes:

$$p = \delta.\sigma.(\frac{1}{R_x} + \frac{1}{R_y}) \tag{B.6}$$

The local curvature of the membrane is given by the 2nd derivative of the deformed membrane surface:

$$\frac{1}{R_x} = \frac{\partial^2 u}{\partial x^2}, \quad \text{and} \quad \frac{1}{R_y} = \frac{\partial^2 u}{\partial y^2} \tag{B.7}$$

Thus the equation of motion (B.6) becomes:

$$p = \delta.\sigma.(\nabla^2 u) \tag{B.8}$$

$$p - \delta.\sigma.(\nabla^2 u) = 0 \tag{B.9}$$

For a membrane element under dynamic loads and deforming in time, we have:

$$pA_c \frac{\partial^2 u}{\partial t^2} = p - \delta.\sigma.\nabla^2 u \tag{B.10}$$

Note that force balance can be simply written down as instead of B.1:

$$p.(\pi r'^2.\delta).\frac{\partial^2 u}{\partial t^2} = p.(\pi r'^2) - \sigma.\delta.(2\pi r).\sin\alpha \tag{B.11}$$

where, $(\pi r^2.\delta)$ is the volume and $\frac{\partial^2 u}{\partial t^2}$ is the acceleration of the element.

Application of Damping

If the membrane is permeable allowing transfer of a fluid through the membrane such as a piece of cloth under tension, the viscous damping due to the relative motion of the fluid through the membrane can be written as:

$$F_{viscous} = -b\frac{\partial u}{\partial t} \tag{B.12}$$

where the damping coefficient can be determined the Darcy's Law governing the flow through a porous medium, according to which, the pressure drop accretes the permeable membrane thickness is given by:

$$\frac{\Delta p}{\delta} = \frac{\mu}{k}\frac{\partial u}{\partial t} \tag{B.13}$$

where μ is the viscosity of the fluid and, k is the permeability of the membrane.

Thus, the final form of the governing equation of motion reaches:

$$p\frac{\partial^2 u}{\partial t^2} = \frac{p}{\delta} - \sigma.\nabla^2 u - \frac{\mu}{k}\frac{\partial u}{\partial t} \tag{B.14}$$

Note that in our discretization the equation of motion we use the finite-element method only for the second-order term, laplaciani on the right hand side.

We use the augmented state equations,

$$w = \begin{bmatrix} u \\ \dot{u} \end{bmatrix} \tag{B.15}$$

to solve the system. In a sense, we use a finite-difference scheme for the time-integration based on the nodal points with a diagonal-only mass matrix:

$$p\frac{\partial^2 u}{\partial t^2} \rightarrow p \quad \frac{d}{dt}\left[w \right] \tag{B.16}$$

Similarly the damping force is also implemented using the state representation and the diagonal damping matrix. The linear system of equations that we solve is given by:

$$p.I\frac{d}{dt}\begin{bmatrix} u \\ \dot{u} \end{bmatrix} = \frac{p}{\delta} - \sigma K\begin{bmatrix} u \\ \end{bmatrix} - \frac{\mu}{k} I\begin{bmatrix} \\ \dot{u} \end{bmatrix} \tag{B.17}$$

$$p.I\frac{d}{dt}\begin{bmatrix} u \\ \dot{u} \end{bmatrix} = \frac{p}{\delta} - \begin{bmatrix} \sigma K & 0 \\ 0 & \left(\mu/k\right) I \end{bmatrix}\begin{bmatrix} u \\ \dot{u} \end{bmatrix} \tag{B.18}$$

where I is the identity matrix.

Appendix C

Implementation of Phong Shading and Vertex Texture Fetch

Vertex Shader GLSL (OpenGL Shading Language) Code

```
varying vec3 View;
varying vec4 color_in;
varying vec2 position_in;
uniform sampler2DRect pos;

void main() {

        vec4 v0 = texture2DRect(pos,vec2(gl_Vertex.x,gl_Vertex.y));

        color_in = gl_Color;
        position_in = gl_Vertex.xy;

        gl_Position = gl_ModelViewProjectionMatrix * v0;
}
```

Fragment Shader GLSL Code

```
varying vec4 color_in;
varying vec2 position_in;
uniform sampler2DRect pos;
uniform sampler2D tex, nrm;

void main(void)
{
	vec3 lightDir = normalize(vec3(gl_LightSource[0].position));
	vec3 halfVector = normalize(gl_LightSource[0].halfVector.xyz);
	float noise = sin(13.0*cos(position_in.y*11.0))+
cos(11.0*sin(position_in.x*13.0));
	float specnoise = mod(noise,0.6);
	float diffnoise = mod(noise,0.3) + 0.7;

	vec4 material = texture2D(tex, position_in/32.0)*color_in;
	vec4 ambient = (gl_FrontMaterial.ambient*material)*
(gl_LightSource[0].ambient + gl_LightModel.ambient);
	vec4 diffuse = (gl_FrontMaterial.diffuse*material) *
gl_LightSource[0].diffuse* diffnoise;
	vec4 specular = (gl_FrontMaterial.specular*material) *
gl_LightSource[0].specular * specnoise;


	vec2 center = floor(position_in + vec2(0.5,0.5));
	vec2 position = position_in - center + vec2(0.5,0.5);
	vec3 v0 = vec3(texture2DRect(pos,vec2(center.x,center.y)));
	vec3 v1 = vec3(texture2DRect(pos,vec2(center.x,center.y-1.0)));
	vec3 v2 = vec3(texture2DRect(pos,vec2(center.x-1.0,center.y-1.0)));
	vec3 v3 = vec3(texture2DRect(pos,vec2(center.x-1.0,center.y)));
	vec3 v4 = vec3(texture2DRect(pos,vec2(center.x-1.0,center.y+1.0)));
	vec3 v5 = vec3(texture2DRect(pos,vec2(center.x,center.y+1.0)));
	vec3 v6 = vec3(texture2DRect(pos,vec2(center.x+1.0,center.y+1.0)));
	vec3 v7 = vec3(texture2DRect(pos,vec2(center.x+1.0,center.y)));
	vec3 v8 = vec3(texture2DRect(pos,vec2(center.x+1.0,center.y-1.0)));

	vec3 n0, n1, n2, n3;

	//face 0: v1 v2 v3 v0
	n0.x = (v0.y - v3.y) * (v0.z + v3.z) + (v3.y - v2.y) * (v3.z + v2.z) +
		(v2.y - v1.y) * (v2.z + v1.z) + (v1.y - v0.y) * (v1.z + v0.z);
	n0.y = (v0.z - v3.z) * (v0.x + v3.x) + (v3.z - v2.z) * (v3.x + v2.x) +
		(v2.z - v1.z) * (v2.x + v1.x) + (v1.z - v0.z) * (v1.x + v0.x);
	n0.z = (v0.x - v3.x) * (v0.y + v3.y) + (v3.x - v2.x) * (v3.y + v2.y) +
		(v2.x - v1.x) * (v2.y + v1.y) + (v1.x - v0.x) * (v1.y + v0.y);
	//face 1: v5 v4 v3 v0
	n1.x = (v5.y - v4.y) * (v5.z + v4.z) + (v4.y - v3.y) * (v4.z + v3.z) +
```

```
            (v3.y - v0.y) * (v3.z + v0.z) + (v0.y - v5.y) * (v0.z + v5.z);
        n1.y = (v5.z - v4.z) * (v5.x + v4.x) + (v4.z - v3.z) * (v4.x + v3.x) +
            (v3.z - v0.z) * (v3.x + v0.x) + (v0.z - v5.z) * (v0.x + v5.x);
        n1.z = (v5.x - v4.x) * (v5.y + v4.y) + (v4.x - v3.x) * (v4.y + v3.y) +
            (v3.x - v0.x) * (v3.y + v0.y) + (v0.x - v5.x) * (v0.y + v5.y);
        //face 2: v7 v6 v5 v0
        n2.x = (v7.y - v6.y) * (v7.z + v6.z) + (v6.y - v5.y) * (v6.z + v5.z) +
            (v5.y - v0.y) * (v5.z + v0.z) + (v0.y - v7.y) * (v0.z + v7.z);
        n2.y = (v7.z - v6.z) * (v7.x + v6.x) + (v6.z - v5.z) * (v6.x + v5.x) +
            (v5.z - v0.z) * (v5.x + v0.x) + (v0.z - v7.z) * (v0.x + v7.x);
        n2.z = (v7.x - v6.x) * (v7.y + v6.y) + (v6.x - v5.x) * (v6.y + v5.y) +
            (v5.x - v0.x) * (v5.y + v0.y) + (v0.x - v7.x) * (v0.y + v7.y);
        //face 3: v1 v8 v7 v0
        n3.x = (v1.y - v8.y) * (v1.z + v8.z) + (v8.y - v7.y) * (v8.z + v7.z) +
            (v7.y - v0.y) * (v7.z + v0.z) + (v0.y - v1.y) * (v0.z + v1.z);
        n3.y = (v1.z - v8.z) * (v1.x + v8.x) + (v8.z - v7.z) * (v8.x + v7.x) +
            (v7.z - v0.z) * (v7.x + v0.x) + (v0.z - v1.z) * (v0.x + v1.x);
        n3.z = (v1.x - v8.x) * (v1.y + v8.y) + (v8.x - v7.x) * (v8.y + v7.y) +
            (v7.x - v0.x) * (v7.y + v0.y) + (v0.x - v1.x) * (v0.y + v1.y);

        n0 = normalize(n0);
        n1 = normalize(n1);
        n2 = normalize(n2);
        n3 = normalize(n3);

        n0 = n0*(1.0-position.x) +     n3*position.x;
        n1 = n1*(1.0-position.x) +     n2*position.x;

        vec3 normal = n0*(1.0-position.y) + n1*position.y;
        normal = normalize(gl_NormalMatrix * normal);
        normal += vec3(texture2D(nrm, position_in/32.0)*0.5- 0.25);
        normal = normalize(normal);

        vec3 halfV;
        float NdotL,NdotHV;

        /* The ambient term will always be present */
        vec4 color = ambient;
        /* compute the dot product between normal and ldir */
        NdotL = max(dot(normal,lightDir),0.0);
        if (NdotL > 0.0) {
            color += diffuse * NdotL;
            halfV = normalize(halfVector);
            NdotHV = max(dot(normal,halfV),0.0);
            color += specular *
                        pow(NdotHV, gl_FrontMaterial.shininess);
        }
        gl_FragColor = color;
}
```