MODELING AND SIMULATION OF SOFT BODIES

by

JARUWAN MESIT

M.S. National Institute of Development Administration, 1999
B.S. Rajabhat Institute Phetchaburi, 1996

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term 2010

Major Professor:
Ratan K. Guha

ABSTRACT

As graphics and simulations become more realistic, techniques for approximating soft body objects, that is, non-solid objects such as liquids, gases, and cloth, are becoming increasingly common. The proposed generalized soft body method encompasses some specific cases of other existing models enabling simulation of a variety of soft body materials by parameter adjustment. This research presents a general method of soft body model and simulation in which parameters for body control, surface deformation, volume control, and gravitation, can be adjusted to simulate different types of soft bodies. In this method, the soft body mesh structure maintains configuration among surface points while fluid modeling deforms the details of the surface. To maintain volume, an internal pressure is approximated by simulated molecules within the soft body. Free fall motion of soft body is generated by gravitational field. Additionally, a constraint is specified based on the property of the soft body being modeled.

There are several standard methods to control soft body volume. This work illustrates the simplicity of simulation by selecting a mass-spring system for the deformation of the connected points of a three-dimensional mesh, while an internal pressure force acts upon the surface triangles. To incorporate fluidity, smooth particles hydrodynamics (SPH) is applied where surface points are considered as free moving particles interacting with neighboring surface points within a SPH radius. Because SPH is computationally expensive, it requires an efficient method to determine neighboring surface points. Collision detection with soft bodies and other rigid body objects also requires such fast neighbor detection. To determine the neighboring surface point, Axis Aligned Bounding Box (AABB), Octree, and a partitioning and hashing schemes

have been investigated and the result shows that the partitioning and hashing scheme provides the best frame rate. Thus a fast partitioning and hashing scheme is used in this research to reduce both computational time and the memory requirements.

The proposed soft body model aims to be applied in several types of soft body application depending on the specific types of soft body deformation. The work presented in this dissertation details experiments with a variety of visually appealing fluid-like surfaces and organic materials animated at interactive speeds. The algorithm is also used to implement animated space-blob creatures in the Galactic Arms Race video game and a human lung simulation, demonstrating the effectiveness of the algorithm in both an actual video game engine and a medical application. The simulation results show that the general model of the soft body can be applied to several applications by adjusting the soft body parameters according to the appearance results.

To my husband, Erin and to my parents, Arom and Tawin Mesit.

ACKNOWLEDGMENTS

Thanks to my advisor, Dr. Ratan Guha, with whose guidance and attention to detail we have published valuable contributions to the field. Thanks to the Computer Science Department at UCF, without whose financial support I would not be able to complete this dissertation. Special thanks to my dissertation committee members Dr. Mostafa Bassiouni, Dr. Sheau-Dong Lang, Dr. Brian Goldiez, and Dr. Matthias R. Brust for generously donating their time, knowledge, and suggestions.

Thanks very much to my parents, Arom and Tawin Mesit, who enabled me to study in the United States and supported me both financially and emotionally. Thanks to Dr. Peter Kincaid and the members of the Hurricane project at the UCF Institute for Simulation and Training. Thanks for Dr. Atem Masunov at the UCF Nanoscience Department who introduced me to monte carlo simulation. Thanks for Dr. Brian Goldiez for his support of the work on high performance computer visualization.

Additionally, I would like to thank fellow lab member Shafaq Chaudhry, who helped keep me on track for all deadlines and graduation paper work. Without her, I might not have graduated on time this semester. Finally, thanks to the Galactic Arms Race team for integrating my soft-body method into their video game.

Jaruwan Mesit

University of Central Florida

Fall 2010

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Deformable models simulate non-solid objects such as cloth [1,2,3,4], hair [5,6,7], elastics [8,9,10,11], and liquids [12,13,14,15] in computer graphics for movies, television, and video games. Deformable models are usually defined by a set of points and, unlike rigid body models, each point moves independently. Individual points in a soft body model have attributes such as position, velocity, and force. Because deformable model vertices constantly rearrange due to interaction with environment, soft body simulation is significantly more complex than rigid body simulation.

There are two general approaches to simulate soft body objects. The first method is to compute interactions between an arbitrary set of points. The arrangement of the points later determines a physical surface mesh which is computed by surface reconstruction algorithms such as Marching Cubes [16]. These methods are commonly employed to simulate viscous materials like liquids [14,15,17]. Other soft body methods simulate semi-soft objects by deforming the points of an existing rigid-body surface without need to periodically recompute the surface mesh. Materials that change shape, yet maintain a relatively coherent structure, include cloth [18,19,20], hair [5,6,7], and elastics [9,21].

1.1 Motivation

Applications of 3D soft body model technology vary widely in the domains of graphics, simulations, and games. Non-real time graphic applications include pre-rendered movie or television special effects, melting or flowing objects, skin, and facial speech animation. Soft-

body models are employed widely in the simulation realm within applications such as interactive surgical training [11,22], where the layers of human tissue may be cut. These simulations include realistically moving human organs such as the heart, lungs, and pulmonary system [23]. In gaming, soft body models may represent cloth and hair of digital avatars, realistic blob-like or slimy creatures, viscous fluids, and many other non-solid surfaces.

1.2 Contribution

This research introduces a general soft body model suitable for many applications that deform existing 3D rigid body models. Several parameters for body control, fluid modeling, volume control, constraints, and gravitational field are applied to a rigid body model to obtain a desired level of deformation. We demonstrate that the proposed general soft body model is able to map into several existing soft body models. For some specific applications, we have selected the certain components where the mass-spring system is used for body control and SPH is applied for fluid modeling. In addition, a partitioning and hashing scheme for both fluid modeling and collision detection is presented to significantly reduce the computation time.

We have experimented with our proposed model in simulating of (1) fluid-like soft body, (2) human surface, (3) blobby objects integrated in Galactic Arms Race (GAR) game engine (http://gar.eecs.ucf.edu), and (4) lung functions. Specifically, in GAR the algorithm creates animated, amorphous blob creatures used as enemies in the game. In the medical application, the lung functions have been simulated where the lung volume is controlled by the Pressure-Volume (P-V) relation. An internal pressure model of P-V relation controls respiratory rhythm while the mass-spring system maintains the body control of the model. The use of the proposed method in

GAR and lung functions demonstrates ways to use the proposed general soft body model in different applications.

1.3 Dissertation overview

The research proceeds as follows. Chapter 2 provides the background of the methods in which our soft body model is developed. Chapter 3 details the proposed general model of soft bodies. Chapter 4 describes the proposed soft body modeling implementation. Chapter 5 presents simulation experiments and comparison results. Chapter 6 demonstrates the soft body model in human lungs for medical application. Finally, chapter 7 discusses the conclusion and future work.

CHAPTER 2: BACKGROUND

This chapter presents previous work on deformable objects related to the methods incorporated into this work. Some of the common approaches to soft body modeling include mass-spring systems, finite element methods (FEM), finite volume methods (FVM), and finite difference methods (FDM), which are discussed in sections 2.2, 2.3, 2.4, and 2.5. Then, in section 2.6 Newton's second law of motion is discussed as it relates to numerical integration in physics. Background work on collision detection is then detailed in section 2.7. Finally, the constraints involved in the proposed model are discussed in section 2.8.

2.1 Previous works

In this section, we present the previous works of deformable objects where several methods such as mass-spring systems, Finite Element Method (FEM), Finite Volume Method (FVM), Finite Differential Method (FDM), etc. have been used to provide behaviors of deformable objects. The element types in triangles, quads, tetrahedrons, or hexahedrons demonstrate the structures of the models in different domain dimensions, such as two dimensions (2D) in clothes or thin shell simulations and three dimensions (3D) in soft tissues or character animations. These deformable objects are later applied to facial, skin, and musculature modeling. In these previous works, we do not present all studies in deformable objects, but rather, we have selected some relevant papers and briefly described the models and the methods that they have used in the models. In this chapter we present the deformable objects in the following categories: cloth simulation in general, cloth simulations with effects, cloth simulations in garment designs,

three dimensional deformable objects in general, and three dimensional facial, skin, and musculature animations.

2.1.1 Two dimensional cloth simulation in general

In [9], Terzopoulos *et al.* propose deformable objects with elastic properties for model surface. They employ FDM to the model structures which provide dynamic and realistic animation. The principles of mathematical physics applied in this model provide the model motions from internal and external components, such as gravitational force, constraints of linkages, viscous fluids, or implemented obstacles (force from the effects of collision). The equation of motion from Newtonian mechanics balances the externally applied force due to the model deformations. The results of this proposed method include surface deformation and deformable solids.

Terzopoulos and Fleischer develop the model of inelastic deformation which presents the viscous and plastic properties in the deformable objects [8]. The spring property that satisfies Hooke's law is used for elongation or contraction of the model surface. In their implementation, they discretize the continuum equations in material coordinates known as a semidiscrete system, which is later integrated through time to simulate the dynamics of deformable models. The semi-implicit time integration produces the elastic displacement that provides the model deformation. The result of the current time step becomes the input data for successive frames of animation. The simulation of a net falling over a spherical obstacle shows the effect of fracture when the deformation is over the limitation of elasticity.

Carignan *et al.* describe the physic-based modeling in cloth for animated synthetic actors in [4]. In this paper, the polygonal panels in 2D define the cloth patterns, which are mapped onto an actor's body in the 3D environment with the physical properties that create the motion of the cloth from the actor's movement. They examine constraints on the cloth when it is attached to rigid moving objects. These constraints include internal elastic force and external forces of gravity, wind, and collision response. The collision detection handles both self collision that might have occurred in each pair of cloth particles and collision between cloth and the human body that happens between the cloth particle and a triangle's vertices of a rigid body. The result shows a sequence of images of a fashion show with the walking motion of synthetic avatars.

Provot presents the mass-spring systems for cloth simulation derived from elastically deformable models and improved them to nonelastic properties of woven fabrics [25]. The network of masses and springs presents the structure of the model. Then numerical integration of the fundamental law of dynamics determines the movement of the end points of each spring. The problem of high stress concentration occurs in a small region of surface at local deformations, which becomes unrealistic when the real deformation of textiles is compared. Thus, Provot proposes a new method that adapts the cloth property to suitable properties of textiles by using dynamic inverse constraints on deformation rates. If the deformation rates of those springs are greater than a critical deformation rate, the dynamic inverse constraints on the super-elongated springs is processed to reduce their elongation. With this method, the cloth simulation in semi-rigid property, as shown in their results, becomes more realistic compared to elastic cloth simulation.

Baraff and Witkin propose cloth simulation with large steps between animated frames to avoid the instability problem [2]. A triangular mesh of particles in cloth models is exerted by internal and external forces on each particle. The most critical forces mentioned in this paper are the internal cloth forces, such as shear, bend, and stretch forces generated by stretching or compression of the cloth. Both shear and bend forces depend on the material of cloth being simulated, but stretch force is set at the same large value for all simulations. The internal cloth forces are derived from a simple continuum formulation to proceed with modeling operations, including stretching or compression. Several constraints are imposed on individual cloth particles. In any simulation, the new positions of the models are evaluated by numerical analysis, such as explicit Euler or implicit Euler methods. The simulation becomes unstable in large deformation where the positions are overshot by the explicit Euler method. To eliminate this problem, the implicit Euler method is used to provide the dynamic motions to overcome the performance limits in the explicit simulation in large step deformation. The simulation of dancers in cloth has shown the effectiveness of this proposed method.

2.1.2 Cloth simulation with drapes, folds, and wrinkles

In cloth simulation, several researchers have proposed techniques to demonstrate the cloth behaviors, such as drapes, folds, and wrinkles, for different properties of textiles. These cloth behaviors involve large deformations in cloth-specific properties like cloth flexibility. Most recently, underwater cloth simulation has also been presented to describe the internal and external dynamics of cloth underwater.

Eberhardt *et al*. describe the fast, flexible, particle-based model to animate the drapes of different types of cloth in some environments, such as air, water, or oil, which require the calculation of exact trajectories of moving particles [26]. A suitable description of internal forces for each particle has been investigated in the form of force plots of tension, shearing, and bending, which later are used to calculate trajectory of a particle via an integration of the Lagrange differential equation. The air resistance and external forces depend on the particle locations, inside or at the border of cloth, giving different levels of air resistance to particular particles. The visual results of cloth simulation with different textures present draping and vibrating effects.

Bhat *et al*. investigate cloth simulation from video data of real fabrics in [27]. Since many parameters need to be adjusted to achieve the appearance of a particular fabric, an algorithm for estimating the parameters of cloth simulation from video data of real fabric is proposed in this paper. The motivated metric compares two consecutive video frames to provide the cloth parameters from the folds of fabrics. The dynamic and static tests on small swatches of fabrics provide appropriate simulation parameters. Then these parameters are used to simulate the fabrics that are worn by a human actor. Four different fabrics (linen, fleece, satin, and knit) are simulated to demonstrate the power of this approach.

Bridson *et al.* present the simulation of clothing with folds and wrinkles in [28]. The proposed methods match the behavior and appearance of clothing that uses a mixed explicit/implicit time integration in numerical analysis. A physically correct bending model combines with an interface forecasting technique that promotes detail in contact regions of the cloth. A post-processing method preserves folds and wrinkles on cloth-character collision.

Additionally, a dynamic constraint mechanism supports the control in large scale folding. The improvement of the simulation realism is achieved by using these techniques to control folds and wrinkles on cloth simulation.

Bridson *et al.* propose the paper for robust treatment of collision, contact and friction for cloth animation that allows actual modeling of cloth thickness in [29]. The post-processed scheme with subdivisions produces the smooth and interference free data in sharp folds and wrinkles in a cloth mesh. The sharp folds can generate the intersections between elements of subdivisions. To eliminate cloth-cloth intersection, the repulsion and collision impulses are used to adjust the cloth positions with no intersection at the end of adjustment. The static friction model in this paper also provides the stable folds and wrinkles as shown in the simulation of a curtain that is draped over a ball. Then, the ball moves and the curtain flips on top of itself.

Decaudin *et al.* present the folds which are generated by the collision with the virtual mannequin [30]. The different buckling patterns for a cylinder of fabric are described in the patterns of diamond buckling, twist buckling, and axis aligned folds. Diamond buckling is the compression of a cylinder of cloth maintaining a zero Gaussian curvature, and has a diamond shape which appears when an elbow or a knee of a character is bent or the sleeves of a sweater are pulled up. When the body twists and when a loose skirt hangs under gravity, the twist buckling and axis aligned folds are the parallel folds generated, respectively. The proposed method for producing these folds works on an ideal fabric cylinder. It also processes a mesh aligned with the main fold direction, called the buckling mesh, which can be compressed and twisted simultaneously. The result of this method shows the realistic 3D mannequin dressed in the designed garment.

Müller *et al.* introduce a method to avoid the velocity layer that usually needs to be updated in every time step to provide the new position of the model [31]. To avoid the instability problem in the explicit Euler method, point based dynamics provide the new position immediately after the model constraints, including constraints of collision. The Solver approach immediately estimates the new locations of the points by trying to satisfy all constraints of the model. By this method, all points in the models can be manipulated immediately during the simulation. Cloth simulation with animated game characters is presented to show the effectiveness of the method.

Ozgen *et al.* simulate underwater cloth with fractional derivatives in [32]. They use a particle-based cloth model that includes half-derivative viscoelastic elements to describe the internal and external dynamics of the cloth. These elements allow the cloth to respond to fluid stresses and the behavior of particles in a viscous fluid. The fractional cloth model uses fluid viscosity to produce bump propagations. The equation of motion that Ozgen *et al.* implement is Fractional Differential Equation (FDE) to where both explicit and implicit numerical solution techniques can be extended. The underwater cloth deformation has been realistically demonstrated and the simulated fractional clothes are compared with real clothes to present the realism of the simulation.

2.1.3 Cloth simulation in Garment design

After the considerable research on cloth simulation is introduced for the properties of the textiles, the computer aided design (CAD) system gains more attention for the garment designs

which try to achieve the incorporation of darts, seams, edges, stiffening pads, and local stretches of the fabric.

McCartney presents the levels of functionalities of CAD in the specifications of the garment details in [33]. The visualization in 2D shapes shows the pattern of the garment, and the 3D specification presents the final form of the garment that is attached around a mannequin. The accurate drape algorithm in cloth simulation constructs the visualization of different fabric types.

Luo and Yuen simulate 3D garments that are generated by 2D garment pattern design to optimize the detail design modification without any change in the topology of 2D patterns [34]. This proposed method emphasizes the smooth geometry where the model deforms from the original boundary to the targeted boundary by using 2D mesh morphing. With this approach, 3D garment fitting simulation allows the 2D pattern modification to work efficiently and speedily since the simulation does not repeat entirely for every modification.

Metaaphanon and Kanongchaiyos present a real-time cloth simulation for garment CAD in [35], where the mass-spring model is parameterized under Newton's laws and cellular structure space is defined for the structure of each garment pattern. The user’s body size is the input to the simulation, which then scales the pattern to fix the user’s virtual body. The structure of the mass-spring includes the shear springs which connect each node with its four diagonally adjacent neighbors and bend springs which connect each node with two horizontal and diagonal neighbors. The cellular structure is based on the theory of algebraic topology which classifies objects into subsets if they are equivalent. In the attaching function presented by Metaaphanon and Kanongchaiyos, the connecting patterns are set to be equivalent to reduce the computational

time of model modification. This proposed method reduces the computational time for pattern making and enables the dress to be remade or refit in real-time.

2.1.4 Three dimensional deformable objects

The 3D deformable objects are used in a wide range of applications such as movies, video games, and surgical training. These 3D deformable objects have additional properties, such as volume, beyond the cloth simulation. However most of the techniques in cloth simulation can be applied to model the objects. Several techniques are proposed for modeling the 3D deformable objects in real-time.

In [36], Sederberg and Parry propose a technique to deform solid geometric models by using free-form deformation (FFD), which can deform surface primitives of any planes, quadrics, parametric surface patches, or implicitly defined surfaces. The deformation of a solid model that requires volume preservation can also be simulated by this method. This proposed technique is based on Bernstein polynomials to provide the deformation effects. The solid model is subdivided into parallelpiped regions, then the control points on the parallelpiped regions control the deformation of the solid body depending on the continuity control. The visual results show several deformations including Coke cans and bottles that are deformed into arbitrary shapes and pipe that is later deformed into a telephone handset.

Chadwick *et al*. present layered construction for deformable animated characters to create and animate the computer generated characters [37]. This methodology combines physically based modeling and geometric modeling with the hierarchy of composite deformations to define both local and global transitions of the character's movement. The skeleton layer provides the

motion specification which is created by a tree structured hierarchy, robotic joint-link parameters, joint angle constraints, and physical attributes. The muscle and fatty tissue layers are implemented by FFD for muscle deformation and the geometric skin data is mapped to the underlying articulated skeleton foundation with smooth quality during squash and stretch behaviors of the characters. The physical model in 3D is composed of a 3D grid of mass points connected by viscously damped Hookean springs. Spring elements that are connected diagonally between mass points on adjacent planes provide the shear strain behavior. The deformation in this animation is constructed from several local and global deformations and the methodology is designed to satisfy specifications of animated characters.

In [38], Müller *et al.* develop a fast and stable simulation in a large deformation where the non-linear technique provides the simulation speed and stability similar to linear technique. This technique provides robustness, speed, and a realistic appearance in the simulation of a large deformation for both FEM and mass-spring systems. Müller *et al.* use the stiffness warping in linear elasticity for deformation, but the stiffness matrix is only computed once and used for the entire simulation to give robust and fast simulations. The elastic forces in the linear elastic technique are evaluated by the element's stiffness matrix, current positions, and the original positions of the points. In the stiffness warping scheme which provides the stability into the system, the local rotation matrices are applied to every vertex instead of using a global rotation matrix. For the rotation tensor field, geometric algebra with multivectors is used to describe the local rotations. The simple and fast way to compute rotation is to evaluate the relative rotation between two orthonormal vectors based on the directions of adjacent edges. This simple method

provides the correct constant rotation matrix for every vertex. The visual results of the long tube, bunny skin, and Dane's skin show the speed and stability of the proposed method.

James and Pai present the Dynamic Response Texture or DyRT that illustrates the simulations of complex, interactive, physically-based, volumetric and dynamic deformation models [39]. With negligible main CPU costs, DyRT is mapped onto any animation in the rendering stage using graphics hardware. The model describes the modal vibration in the form of a linear elastodynamic equation for a finite element model. Then exiting modes with rigid motions are included to produce the realistic modal deformation from bone-based animation. To increase the detail of the surface deformation, correct normal calculation is added to each vertex by considering the neighboring vertex information. This causes more memory requirement, however the normal correction is applied only for some particular modes. The animation of a jumping motion shows the vibrations of thigh and belly that are implemented by this method.

Hauser *et al.* describe the modal analysis with constraints in [40] to present the model manipulation, collision, and other constraints that can be implemented within a modal framework. The modal composition of a physical system is described in the linearized form of the system's stiffness, damping, and mass matrices. To convert the complicated non-linear system to the simple linear system, this modal analysis takes the system's nonlinear description, matches the description to a linear approximation, and finds a coordinate system that diagonally relates to the linear approximation. The system is designed as piecewise-linear tetrahedral finite element method (FEM). Hauser *et al.* combine the modal simulation with a standard rigid body dynamic simulator for dynamic simulation. The main benefit of this approach is that the behavior of the system is calculated efficiently and analytically which provides stability to the system. Several

models are implemented to show the simulation results and the geometric and kinematic complexity of the models.

Müller *et al.* introduce point based animation for the objects that have a range from stiff elastic to highly plastic in [41]. This method animates the objects with large deformation. They use the Moving Least Square (MLS) to provide the derivatives of the discrete displacement fields. The elasticity and plasticity models are presented for model behaviors and then the displacement approach is described to define the displacement field that animates the models. The continuum equations in the elastic model describes the elastic stresses in the volumetric models at the given deformation field. The Green's strain tensor is used to preserve the model volume with additional energy. The strain state variable simulates the plastic behavior in this model. The multi presentation approach is presented when fracturing or merging occurs at some parts of the model, and when a collision between a highly deformable object and rigid body happens. The results show that the different levels of deformation depend on the effects of Poisson’s ratio used for the volume preservation, and that the deformation of the reference shape deviates from the original shape.

Teschner *et al.* describe a versatile and robust model for geometrically complex deformable solids in [42] which can be applied to either deformable tetrahedral meshes or triangle meshes. Both elastic and plastic deformation are considered for a large variety of material properties. The materials ranging from stiff to fluid-like behavior can be presented with this method. These potential energies are derived from the specific constraints applied at each mass point to deform the object. This potential energies in this model support distance, surface area, and volume preservations. For numerical analysis, the Verlet scheme for number

integration [43] is used for linear trajectories of all mass points. In the Verlet scheme, each integration step requires only one force computation. This scheme provide high accuracy, and the integration of positions does not depend on the integration of velocities if the system uses undamped forces. The sequence of highly elastic deformation, plastic deformation, and the melting object demonstrates the performance of this model.

Müller *et al.* propose the method for interactive virtual material in [44] where the warped stiffness finite element approach for linear elasticity is combined with a strain-state-based plasticity model. The internal stress component in finite element computation in the tetrahedrons determines the fracture locations and orientations of the model. The stiffness warped stiffness in [38] has two disadvantages. First, the rotation matrix of the vertex needs to be computed from the location of its adjacent vertices. Second, elastic forces are not guaranteed to add up to zero. These problems can be solved by extracting rotations of the tetrahedral elements instead of vertices. The model becomes perfectly elastic and it turns to the original shape when the external forces are removed. For the plasticity, the method is presented by the linear FEM and warped FEM with implicit integration. The simulation of this method shows the large rotational elastic deformation and the melting of the model under gravity.

Müller *et al.* present the pointed based object and non-connectivity information in [45]. This approach presents the replacement of energies by geometric constraints and forces from the current positions and the goal positions. Each point has an original position and then it is mapped to the deformed shape which is later pulled toward the goal position. These goal positions are defined by shape matching at an undeformed rest state of the models. Since the goal positions are defined, the points do not overshoot the equilibrium or goal positions. Thus, the instability

problem in the explicit Euler method is eliminated by this method. The linear deformation presents the model motion of shear and stretch. To extend the deformation, the quadratic deformation is used for the motion of twist and bending. To extend the motion even further, the cluster based deformation has been presented to provide the deformation based on the model clusters. A reasonable number of deformable objects can be handled by this approach in real time.

Botsch *et al.* present Primo: coupled prisms for intuitive surface modeling in [46] where the surface mesh is embedded in a layer of volumetric prisms with non-linear elastic force. In this model the rigidity of prisms preserves the model shape under high deformation, which supports the numerical stability of the system. Local and global shape matching are employed for body deformation. The goals of this method are to provide robust and physically plausible large deformation, preserve the surface detail of the model, present constraint-based and force-based deformations, and demonstrate intuitive geometric parameters for surface modeling. In this prism-based modeling, the model positions and orientations of an arbitrary subset of prisms are prescribed by the users. Then, the resulting prisms define the positions of the deformed surface mesh from the average transformation of its incident prisms. As another benefit of this method, the surface behavior is controlled by the geometrically intuitive parameter to improve the large and complex deformation. The results show the flexibility of prism-based modeling for the complex shape deformation and general surface processing.

Galoppo et al. propose a fast and novel algorithm for the soft articulated characters with fast contact handling in [47]. The dynamic skeleton skin interacts with elastic deformation in the pose space of the skinned surface. The approximation of Schur complements is used for skeleton and skin computations that bring robustness to the system. The layered representation for soft

characters provides an integration of articulated body dynamics and skinning with displacement corrections. The pose space deformation is the skeletal-subspace deformation with the number of bones. Then elastic deformation of the skin is performed at the rest pose space. The methods of Lagrange multiplier model both joint and contact constraints, and implicit Backward Euler integration provide the numerical approximation for the model. The fast collision detection module is adopted by a fast image-based algorithm which performs on the layered representation of the soft characters. The experiments show that the simulation of the soft articulated characters produces the deformed skin in detail and the simulation handles the contact constraints in interactive speed.

2.1.5 Three dimensional facial, skin, and musculature animations

Significant efforts have been devoted to animate models of human face, skin, and muscles. These animations follow the procedures of designing 3D mesh and animating the 3D mesh from the movement of a human body. Thus, several methods have been developed to approximate the muscle actions that affect facial and skin models.

Gourret *et al*. present the FEM that is used to simulate object and human skin deformations in a grasping task [48]. Both the forces of the fingers on the object and the object on the fingers are presented by a numerical method based on finite elements that control the synthetic human behavior. The finite element approach provides visual realism because the body surface of the model corresponds to an element face at the body boundary. Since FEM requires high memory space and computational time, Gourret *et al*. use zero order continuity instead of higher order continuity. The physical modeling of deformable objects is shown by presenting the

example of a contact problem with the grasping and pressing of a ball. For the example of simulation, the contact force is created by muscular forces acting on bones, then the human skin is deformed by the process of joint flexing. They have successfully developed both object and synthetic human deformations and their contacts by using finite element theory.

In [49], Lee *et al.* describe realistic modeling for facial animation where scanning range sensor creates the realistic human facial models and animates the facial geometries through the dynamic simulation of facial tissues and muscles. The Cyberware scanner automatically constructs an efficient and fully functional model of the subject's head. Then the well-structured face mesh is modified by the control parameters to compensate for geometrical facial features from person to person. For the discrete deformable model, Lee *et al.* use a node-spring-node structure. Node properties include mass, position, velocity, acceleration, and net nodal forces. The tissue model around the eyes and nose involves the force spring that is exerted directly on nodes on both parts. For the muscles of facial expression that spread out below the facial tissue, short elastic tendons are created to attach the facial musculature and skin tissue. This provides the movement of facial tissue caused by contraction of facial muscles. The well-known explicit Euler method is used for numerical simulation. The facial animation examples include expressions of surprise, anger, quizzical look, and sadness.

Scheepers *et al.* present anatomy-based modeling of human musculature which describes the relationship between the exterior form and structure of human muscles [50]. The influence of the musculature on the surface form is considered to develop muscle models that react to the change of postures of an underlying articulated skeleton. They select the ellipsoids to represent the muscles bellies since ellipsoids can be scaled along three major axes to simulate bulging of

the muscles. The dimensions of the muscle belly are adjusted automatically when the muscles move further apart or close together. The multi-belly muscle model is positioned automatically for wide muscles with complex shapes. Scheepers *et al.* construct bicubic patch meshes along a cubic Bezier curve to define the muscles with complex shapes. The control parameters that determine the shape of the muscle include muscle volume, height and width ratio of the muscle's bulge, location, direction, and orientation. The result of muscle deformation is shown by the animation sequence of muscle bulges when the forearm is flexed at the elbow joint.

Teran *et al.* develop a simulation of skeletal muscle that uses FVM in [18]. They show that FVM is able to simply interpret the stress inside a tetrahedron experiencing multidimensional force that pushes on each face. The time stepping scheme is selected by a mixed explicit/implicit method for the equation of motion. The stress exerted by a volume element gives a measure of the material deformation using a hyperelastic component, a quasi-incompressible component, and a transversely isotopic component. Hyperelastic material refers to a soft elastic material that can create the large deformations in a muscle. The FVM and a quasi-incompressible component simulate the contracting muscle tissue while B-spline models the fiber direction of the muscle. The simulations of skeletal muscle of isometric contraction of both biceps and triceps muscles show the effectiveness and robustness of FVM.

Capell *et al.* introduce the physically based rigging for deformable characters in [51] where the rigging characters are modeled as dynamic elastic bodies. The forces from the building blocks of rigging guide the character shape and the models are combined with other forces such as gravity, physical constraints, and user interaction during animation. This deformable character is defined by elastic domains as the structure of triangular mesh with FEM approximation, a

control lattice, skeleton consisting of prescribed edges, and rigging force fields. The specifications of elastic domains include mass density, Poisson ratio, and Young's modulus. The motion of a character is solved by Euler-Lagrange equations. A pose-dependent linear system has to satisfy two goals: forces and deformations need to be in correspondence for a given pose and for the computation of rig forces for a given surface deformation at a given pose, and the simulation needs to rely on the static equilibrium solution to the equation of motion. A surface deformation rig affecting a chest flex is shown to present the effects of rigging forces introduced by this paper.

Stoiber *et al.* present facial animation retargeting and control based on human appearance space in [52] that combines with the parameter-based animation method to offer a precise control on facial configuration and performance-based animation to naturally capture human motion. The human appearance space presents a coherent and continuous parameterization of human facial movement. The topological characteristics of the appearance provide the principal variation patterns of a face and then reorganize them on a low-dimension control space. This control space manipulates the facial expression of a synthetic face. Each dominant direction in appearance space on a human face in high-dimensional space is mapped to the one direction for the facial expression on a virtual character in low-dimensional space. The facial expression has been successfully transferred to the synthetic face from the information between appearance space and control space.

In deformable objects, mass-spring systems, FEM, FVM, and FDM are common techniques that present body control, volume control, and fluid modeling. At each point of the model, the force generated by the body control, volume control, or fluid modeling is modeled

differently depending on the technique that is applied in the model. Each technique affects either real-time or accuracy requirements. In the next four sections, we briefly describe some examples of those techniques that can be applied to body control, volume control, or fluid modeling in the general soft body model.

2.2 Mass-spring Systems

Soft bodies using mass-spring systems can be classified into two different categories: modeling a 3D object as a 2D grid structure e.g. cloth, and modeling a 3D object as a 3D structure e.g. a bouncing ball. In these models, a soft body is represented as a triangular, rectangular, or tetrahedral mesh where each point has its own properties such as mass, velocity, force, and position. The force exerted at each point is cumulative of the forces of its neighbors and is represented by a differential equation which can be evaluated using numerical integration.

For 2D grid modeling of a 3D object, Terzopoulos *et al.* [9] propose deformable objects with elastic properties which have successfully been used in soft body cloth simulation [1,53]. The cloth simulation presented in [9] is shown in figure 2—1. Provot [54] uses a mass-spring system to present the structure of cloth as a mesh of $m \times n$ mass points which are relocated at each time step. In Provot's method, the internal spring force, $\boldsymbol{F}(P_{i,j})$, acting on a particular surface point $P_{i,j}$, is calculated by spring forces that connect the point to its neighbors and is given by:

$$\boldsymbol{F}_{\text{int}}(P_{i,j}) = -\sum_{(k,l)\in R} k_{s(i,j,k,l)} \left[l_{i,j,k,l} - l^{0}_{i,j,k,l} \frac{l_{i,j,k,l}}{|\, l_{i,j,k,l} \,|} \right], \tag{2.1}$$

where R is the set of all points (*k, l*) linked to $P_{i,j}$ by a spring; $l_{i,j,k,l}$ is the vector $\overrightarrow{P_{i,j}P_{k,l,}}$; $l^0_{i,j,k,l}$ is the rest-length of the spring that links $P_{i,j}$ and $P_{k,l}$; and $k_{s(i,j,k,l)}$ is the stiffness of the spring $(P_{i,j}, P_{k,l})$. The damping force at point $P_{i,j}$ is given by:

$$\boldsymbol{F}_{dis}(P_{i,j}) = \quad -k_d \boldsymbol{v}_{i,j}, \tag{2.2}$$

where $-k_d$ is the damping coefficient, and $\boldsymbol{v}_{i,j}$ is the velocity of point $P_{i,j}$. To avoid unrealistic deformation of a hanging cloth, he proposes to increase stiffness of the spring. Fuhrmann *et al.* [55] use collision detection to simulate interactive animation of cloth and use their results to simulate virtual garments. Since cloth simulation collapses in explicit integration methods for large spring constants, several researchers propose implicit Euler integration method to change position, velocity, and force of a point in each time step [1,2,53,56].

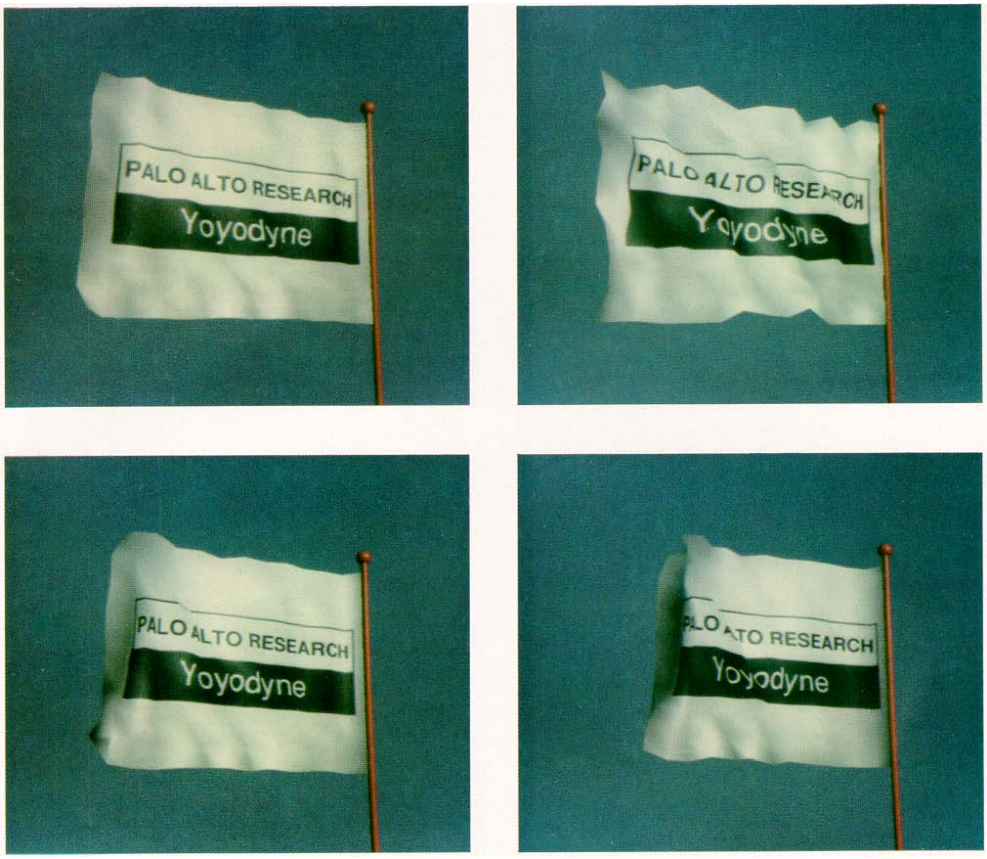

Figure 2—1: Flag waving in the wind simulated by Terzopoulos et al. [9]. He proposed deformable objects with elastic properties which have successfully been used in soft body cloth simulation

A lot of work has been done on 3D modeling of 3D structures like simulation of surgery, fluid-based soft body, and bouncing ball. A layer mass-spring system for facial animation was first presented in [57]. In craniofacial surgery simulation [22,58], the patient's skin, muscle, and bone layers are modeled using mass-spring systems where the deformation of each layer is based on its mechanical spring constant. Padilla *et al.* [59] present the Transurethral Resection of the

Prostate (TURP) method to remove inner prostate tissue using a mass-spring system in 3D structure. The mass-spring system in this model calculates the internal force, $\boldsymbol{F}_i$, acting on point i by:

$$\boldsymbol{F}_i = \sum_{j \in N(i)} k_{i,j} \frac{\left|p_i - p_j\right| - l^0_{i,j}(p_i - p_j)}{\left|p_i - p_j\right|}, \tag{2.3}$$

where $k_{i,j}$ is the stiffness coefficient of the spring connecting point i to any point j in the neighborhood $N(i)$ of i; p_i is the current position of point i; p_j is the current position of point j; and $l^0_{i,j}$ is the spring length at rest position.

Nixon and Lobb [60] present mass-spring system that uses classical fluid mechanic's Navier-stokes differential equations to simulate soft body surfaces. The force interacting with the surface at one particular point is generated from the fluid force exerted by neighboring points. The spring forces, $\boldsymbol{F}_{ij}$, between two points, i and j, at positions, p_i and p_j, with velocities, $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$, is given by:

$$\boldsymbol{F}_{ij} = \left[k_s(|\, p_j - p_i \,| - l^0_{ij}) + k_d \frac{(\boldsymbol{v}_j - \boldsymbol{v}_i)\cdot(p_j - p_i)}{|\, p_j - p_i \,|} \right] \frac{p_j - p_i}{|\, p_j - p_i \,|}, \tag{2.4}$$

where k_s and k_d are the spring and damping constants respectively, and l^0_{ij} is the spring's rest-length. The force $\boldsymbol{F}_{ji}$ is equal and opposite to $\boldsymbol{F}_{ij}$, i.e., $\boldsymbol{F}_{ji} = -\boldsymbol{F}_{ij}$.

2.3 Finite Element Method (FEM)

The finite element method (FEM) is used in many applications to simulate body deformation. FEM solves an equation by approximating a continuous set of discrete points, usually grid or mesh. Problems of complexity and unusual geometry can be evaluated by FEM. Thus, FEM is one of the powerful tools in the problem of heat transfer, fluid mechanics, and mechanical systems. To pursue the physical realism in soft body simulation, it is important to keep computation efficient. The main benefits of FEM are accuracy and realism in performing the soft body deformation. Since FEM can accurately simulate soft body deformation, it requires more computation which can reduce the simulation speed. Several methods are proposed to increase the simulation speed while maintaining the simulation accuracy.

The volume of deformable objects can be preserved by FEM better than mass-spring system but the computation is more complicated, which causes the simulation speed to slow down. To provide a fast and robust simulation, the FEM with warping stiffness is employed in interactive virtual material [44] for simulating elasto-plastic materials. FEM is employed for the body deformation by using the model structure of a tetrahedral mesh. For the elasticity model, the deformation of an object is described by a vector field $u(x)$ which is given by:

$$u(x) = H_e(x) \cdot \hat{u}\,, \tag{2.5}$$

where $H_e(x)$ contains the shape functions of the tetrahedral element, $\hat{u} = [u_1, v_1, w_1, ..., u_4, v_4, w_4]^T$ is the collection of the displacement vectors at the four vertices of a tetrahedron.

By applying Cauchy's linear strain tenser [61], the strain, ε, in each tetrahedron becomes

$$\varepsilon = B_e \cdot \hat{u}, \tag{2.6}$$

where B_e is computed for each tetrahedron.

Hooke's law is applied for the stress, σ, within each element. Then we get

$$\sigma = E \cdot \varepsilon = EB_e\hat{u}, \tag{2.7}$$

where E depends on two scalars, Young's modulus and Poisson's ratio [61], for isotropic material.

Thus, the elastic force is defined by:

$$\boldsymbol{F}_e = K_e\hat{u}, \tag{2.8}$$

where $K_e = V_e B_e^T E B_e$ is the stiffness matrix computed for each tetrahedral element and V_e is the volume element. Then the global stiffness of entire tetrahedral mesh, K, is calculated from K_e.

In Lagrange's form, the dynamic behavior of the object is described as:

$$M\ddot{x} + C\dot{x} + K(x - x_0) = \boldsymbol{F}_{ext}, \tag{2.9}$$

where $\dot{x}$ and $\ddot{x}$ are the first and second derivatives of x with respect to time, M is mass matrix, C is the damping matrix, and $\boldsymbol{F}_{ext}$ is a vector of external forces.

Under large deformation, element-based warped stiffness is applied to the rotation at the elements of the model instead of the vertices of the model. By using the warped stiffness concept to each tetrahedral element the elastic force, $\boldsymbol{F}_e$, is computed by

$$\boldsymbol{F}_e = K_e^{'}x + \boldsymbol{F}_{oe}^{'}, \tag{2.10}$$

where $K_e^{'} = R_e K_e R_e^{-1}$, R_e is the rotation matrix of the tetrahedron and a force offset $\boldsymbol{F}_{oe}^{'} = R_e\boldsymbol{F}_{oe}$, $\boldsymbol{F}_{oe}$ is the elastic force at time 0.

Then, the global elastic force is generated by:

$$\boldsymbol{F} = K^{'} x + \boldsymbol{F}_{\boldsymbol{o}}^{'}. \tag{2.11}$$

For the plastic model , the plastic forces, $\boldsymbol{F}_p$, generated by linear warped FEM, is expressed by:

$$\boldsymbol{F}_p = R_e P_e \cdot \varepsilon_p , \tag{2.12}$$

where $P_e = R_e P_e \cdot \varepsilon_{plastic}$ is the plastic matrix.

Finally, the force for the elasticity combined with plasticity is computed by:

$$M\ddot{x} + C\dot{x} + K^{'} x + \boldsymbol{F}_o^{'} - \boldsymbol{F}_p = \boldsymbol{F}_{\boldsymbol{ext}} . \tag{2.13}$$

2.4 Finite Volume Method (FVM)

Finite volume refers to a small volume surrounding each node point on a mesh. Fluxes can be evaluated at the surfaces of each finite volume. In the Finite Volume Method (FVM), volume integrals in a partial differential equation are converted to surface integrals. An advantage of the FVM is the simplicity of formulating unstructured meshes that can be applied in many computational fluid dynamics. For example, in [62] the FVM has been used to solve the incompressible Navier-Stokes equations to provide more accuracy in arbitrary boundary conditions and sharp geometric features in fluid simulation. The implementation is based on the grid mesh data in the form of a tetrahedral mesh in 3D simulation. The differences of this simulation and the existing FEM are the applied mathematics and computational physics which aim to provide a fast and plausible visual simulation. The simulations include free surface fluid simulations and smoke simulations flowing around objects that have complex geometries.

In [18] FVM is presented for muscle tissue by using B-spline solids to model fiber directions and muscle movements that are derived from key frame animation. Since muscle tissue fibers are rearranged uniformly to exhibit several regions of fiber directions, B-spline solid models capture the detail of fiber directions and then assign fiber direction to individual tetrahedrons for muscle simulation meshes. To find the movement of muscles, the muscle force distribution is computed between the sets of which they span.

In the configuration of the model, it is divided up into the number of discrete regions that have nodes at the midpoints of the regions. Thus the force on node x_i surrounded by the region Ω is calculated by:

$$\boldsymbol{F}_i = \frac{D}{D_t}\int_{\Omega} \rho \boldsymbol{v}\, dx = \oint_{\partial\Omega} t\, dS = \oint_{\partial\Omega} \sigma n\, ds\,, \tag{2.14}$$

where σ is the density, $\boldsymbol{v}$ is the velocity, t is the surface traction on $\partial\Omega$, n normal unit vector of the region, and $t = \sigma n$ in the Cauchy stress. In the triangle, the force on node x_i is computed by:

$$\boldsymbol{F}_i += -\frac{1}{2}\sigma(e_1 n_1 + e_2 n_2)\,, \tag{2.15}$$

where e_1 and e_2 are the edge lengths of the triangles that are connected, and n_1 and n_2 are normal vectors of the triangles.

For the tetrahedral mesh, each tetrahedron has three faces that contributes the force on each node. Thus the combined force at each point on the tetrahedron is described as:

$$\boldsymbol{F}_i += -\frac{1}{3}\sigma(a_1 n_1 + a_2 n_2 + a_3 n_3)\,, \tag{2.16}$$

where a_1, a_2, and a_3 are areas of the faces that are connected on x_i .

2.5 Finite Difference Method (FDM)

The finite difference method (FDM) is a mesh discretization technique applicable to several branches of mechanics static and dynamic problems. To apply FDM, discrete functional values are specified at the center of the node. Then, either Euler's equation or energy function computes the approximated value of the function. Euler's equation expresses the conditions of dynamic equilibrium of the continuous function and obtains the mathematical form of the partial differential equation. In the general soft body model, fluid modeling with incompressible fluid provides the detail of bump propagation of surface which can be simulated by Smooth Particle Hydrodynamics (SPH). SPH is considered as a form of FDM because it is an interpolation method for particles where field qualities, such as density, fluid pressure, and fluid density, are defined at the center of a particle's location within a specified distance. Since we use incompressible fluid with SPH for fluid modeling, in the this section we describe mechanics for incompressible fluid and then summarize SPH.

2.5.1 Fluid mechanics

The fundamental equations of fluid dynamics are the conservation laws; *conservation of mass* and *conservation of momentum*, which are based on classical mechanics, modified in quantum mechanics, and general relativity presented in [60].

2.5.1.1 Continuity equation (conservation of mass)

Let ∇ be gradient vector, the equation for Conservation of Mass is given by:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{v}) = 0, \tag{2.17}$$

where ρ and $\boldsymbol{v}$ are density and velocity vectors, respectively.

In incompressible fluid, eq. 2.17 becomes

$$\nabla \cdot \boldsymbol{v} = 0 \tag{2.18}$$

2.5.1.2 Navier-Stokes equation (conservation of momentum)

For the conservation of momentum, the Navier-Stokes equation derived for incompressible fluid states that:

$$\rho\left[\frac{\partial v}{\partial t}\right] = -\nabla L + \rho \boldsymbol{g} + \mu \nabla^2 \boldsymbol{v}, \tag{2.19}$$

where the fluid or liquid pressure force is the first term of the equation, $-\nabla L$, the gravity force is the second term, $\rho \boldsymbol{g}$, and the viscous force is the third term, $\mu \nabla^2 \boldsymbol{v}$.

We use the Navier-stokes equation to apply the fluid pressure force to surface tension in the soft body by SPH.

2.5.2 Smooth Particles Hydrodynamics (SPH) in fluid dynamics

We simulate the fluid-layer by using the SPH fluid model in [63,64]. The SPH in fluid modeling presented in [63] is shown in figure 2—2. SPH is an interpolation method for particles where field qualities such as density, force, etc., are defined at discrete particle locations and are evaluated within a specified distance [65,66,67]. The smoothing kernel, $W(r, h_w)$, is a function

for the smoothing kernel with the radius, h_w, and the distance, r, where r is calculated from the positions of surface points i and j. The gradient and laplacian of the smoothing kernel are $\nabla W(r, h_w)$ and $\nabla^2 W(r, h_w)$, respectively. Based on interpolation, the scalar quality A at position r is the weighted sum of all particles in the sampled area. The basic interpolation formula for any quality A is given by:

$$A_S(\mathbf{r}) = \sum_j m_j \frac{A_j}{\rho_j} W(r - r_j, h_w), \tag{2.20}$$

where j is a local particle in the sampled area, m_j is the mass of particle j, r_j is position of particle, ρ_j is density, A_j is the field quantity at r_j, and $W(r - r_j, h_w)$ is kernel function of the sampled area.

Since particle mass in SPH never changes, the conservation of mass property in fluid eq. 2.17 is satisfied. From eq. 2.19, there exist three terms: fluid pressure force, $-\nabla L$, gravity force , $\rho \boldsymbol{g}$, and the viscous force, $\mu \nabla^2 \boldsymbol{v}$.

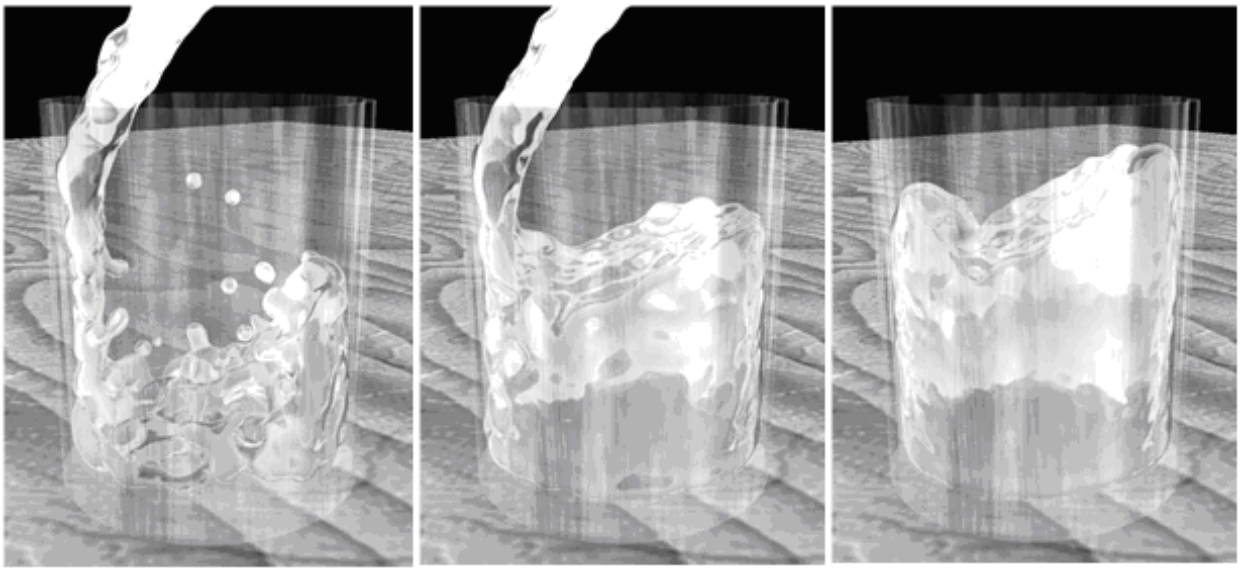

Figure 2—2: Pouring water into a glass simulated by Müller et al [63]. The SPH in fluid modeling has been use in this simulation.

For the choice of smoothing kernel to be used, there are several properties needed to be considered such as stability, accuracy, and speed. In this research, we focus on application in fast simulation where simple calculation is the main factor. In general quality field, Matthias et al. [63] suggest the kernel which is given by:

$$W_{poly6}(r,h_w) = \frac{315}{64\pi h_w^{9}} \begin{Bmatrix} (h_w^{2} - |r|^2)^3 & 0 \le |r| \le h_w \\ 0 & otherwise \end{Bmatrix}, \text{ and} \tag{2.21}$$

$$\nabla W_{poly6}(r,h_w) = \frac{945}{32\pi h_w^{9}} \begin{Bmatrix} (h_w^{2} - |r|^2)^2 r & 0 \le |r| \le h_w \\ 0 & otherwise \end{Bmatrix}. \tag{2.22}$$

However, for the fluid pressure force and viscosity force the different kernels are used to simulate different behaviors as described next.

2.5.2.1 Fluid pressure force kernel

Under high pressure, particles cluster close to each other, and the relative force approaches to zero in the gradient of the kernel. The *spiky kernel* is ideally suited for this behavior. Thus, to compute the fluid pressure force, we use the spiky kernel presented in [63] which is given by:

$$W_{spiky}(r, h_w) = \frac{15}{\pi h_w^{\,6}} \begin{Bmatrix} (h_w - |r|)^3 & 0 \le |r| \le h_w \\ 0 & otherwise \end{Bmatrix}, \text{ and} \tag{2.23}$$

$$\nabla W_{spiky}(r, h_w) = \frac{45}{\pi h_w^{\,6}} \begin{Bmatrix} (\frac{h_w^{\,2} + |r|^2}{r} - 2h_w)r & 0 \le |r| \le h_w \\ 0 & otherwise \end{Bmatrix}. \tag{2.24}$$

Fluid pressure force is computed as

$$\boldsymbol{F}_{fpi} = -\sum_{j=0}^{n-1} m_j \frac{L_i - L_j}{2\rho_j} \nabla W(r_i - r_j, h_w), \tag{2.25}$$

where $\boldsymbol{F}_{fpi}$ is the fluid pressure force, m_j is the mass of particle j, L_i and L_j are pressures at particles i and j, i is a *considering particle,* and j is a *local particle* in the sampled area of particle i.

2.5.2.2 Viscosity kernel

Viscosity should have a smoothing kernel that affects velocity field, whereas a standard smoothing kernel does not have this property. Additionally, if two particles move toward to each other, the Laplacian of the smoothed velocity field should produce a positive value. Thus, to compute the viscosity force, we use the *viscosity kernel* presented in [63] which is given by

$$W_{viscosity}(r, h_w) = \frac{15}{2\pi h_w^3}\begin{Bmatrix} -\frac{|r|^3}{2h_w^3} + \frac{|r|^2}{h_w^2} + \frac{h_w}{2|r|} - 1 & 0 \le |r| \le h_w \\ 0 & otherwise \end{Bmatrix}, \text{ and} \tag{2.26}$$

$$\nabla^2 W_{viscosity}(r, h_w) = \frac{45}{\pi h_w^6}\begin{Bmatrix} (h_w - |r|) & 0 \le |r| \le h_w \\ 0 & otherwise \end{Bmatrix}. \tag{2.27}$$

To compute the viscosity force from fluid, we use

$$\boldsymbol{F}_{fvi} = \mu \sum_{j=0}^{n-1} m_j \frac{\boldsymbol{v}_j - \boldsymbol{v}_i}{\rho_j} \nabla^2 W(r_i - r_j, h_w), \tag{2.28}$$

where $\boldsymbol{F}_{fvi}$ is the viscosity force from fluid, μ is the viscosity of fluid, m_j is mass at particle j, $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ are velocities at particles i and j, respectively, i is a considering particle, and j is a local particle in the sampled area of particle i.

2.6 Newton's Second Law of Motion for Soft Body Animation

To animate the soft body, we consider a mesh of points drawn on its surface and use Newton second law of motion to animate those points. The second law of motion states that the

time rate of change of a body's momentum is equal to the vector sum of the external forces acting on it [68]. This equation can be expressed by using the following first order differential equation:

$$\frac{d\boldsymbol{v}(t)}{dt} = \frac{\boldsymbol{F}(\boldsymbol{v}(t), t)}{m}, \tag{2.29}$$

where $\boldsymbol{v}$ is the velocity of an object; $\boldsymbol{F}$ is the cumulative force that applies on the object and is a continuous function of velocity, $\boldsymbol{v}$, and time, t; and m is the mass of the object assumed to be constant in our simulation. By a basic theorem of calculus, integrating eq. 2.29 over the interval $[t, t+h]$ yields

$$\boldsymbol{v}(t+h) = \boldsymbol{v}(t) + \int_{t}^{t+h} \frac{\boldsymbol{F}(\boldsymbol{v}(t), t)}{m} dt, \tag{2.30}$$

where h is the interval time step.

This equation expresses that the velocity can be computed at time $t+h$ if we know (a) the initial velocity of the object at time t and (b) the net force acting on the object at every instant between t and $t+h$. Since the net force is a function of $\boldsymbol{v}$ and velocity is not known over the interval of integration, the integral in eq. 2.30 cannot be exactly calculated. This requires approximating the value of $\boldsymbol{F}$ across the interval of integration by using either the explicit, implicit, or trapezoid Euler methods. In the next section, we discuss the explicit and implicit Euler methods.

2.6.1 Explicit Euler Integration

Explicit Euler integration is a straight-forward approach for soft body simulation with pressure forces where velocity, point, and force are related by the following set of equations [1,53]:

$$\boldsymbol{v}_i^{t+h} = \boldsymbol{v}_i^t + \boldsymbol{F}_i^t \frac{h}{m_i} \tag{2.31}$$

$$p_i^{t+h} = p_i^t + \boldsymbol{v}_i^{t+h} h \tag{2.32}$$

Here, $\boldsymbol{v}_i^t$ is the velocity of point i at time t, $\boldsymbol{F}_i^t$ is the force acting on point i at time t, p_i^t is the position of point i at time t, and h is the time interval between simulation steps. The position of point i at time $t+h$ can be easily evaluated by the current values of p_i^t, $\boldsymbol{v}_i^t$, and $\boldsymbol{F}_i^t$.

In the explicit Euler method, the velocity at time $t+h$ is evaluated from force at time t and stability is achieved only when the time steps are small [69]. In fact, for stability, the step size must be inversely proportional to the square root of the stiffness [69]. Otherwise, the simulation will fail.

2.6.2 Implicit Euler Integration

The advantage of implicit Euler integration over explicit Euler integration method is that it solves the stability problem. The example of cloth simulation presented in [1] using implicit Euler integration is shown in figure 2—3. A large step can be applied to implicit Euler integration without simulation failure. To find the value of variables in the subsequent time step, implicit Euler integration method replaces $\boldsymbol{F}_i^t$ by $\boldsymbol{F}_i^{t+h}$ as follows [1,53]:

$$\boldsymbol{v}_i^{t+h} = \boldsymbol{v}_i^t + \boldsymbol{F}_i^{t+h} \frac{h}{m_i} \tag{2.33}$$

$$p_i^{t+h} = p_i^t + \boldsymbol{v}_i^{t+h} h \tag{2.34}$$

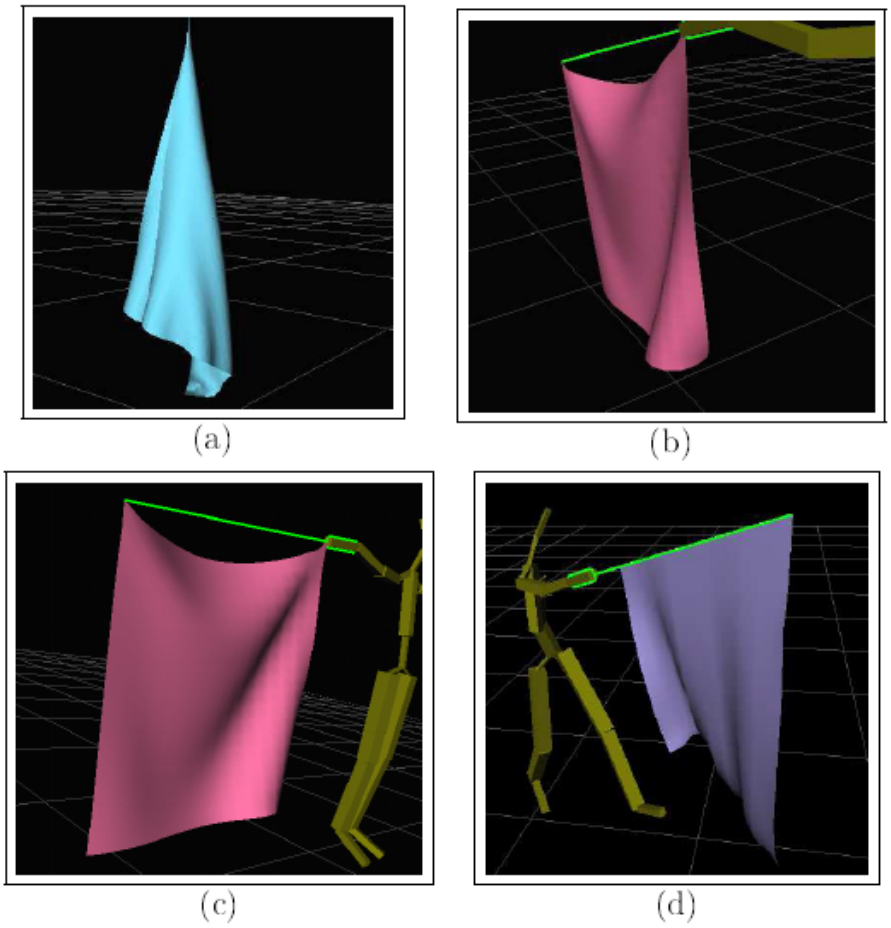

Figure 2—3: Generated clothes simulated by Kang et al [1]. The implicit Euler method is used in this cloth simulation.

This small change has been proven to solve unstable conditions in the explicit Euler method [70]. After the new positions of all surface points are evaluated, those positions can be collided with other objects. Thus it is necessary to detect the collision between soft bodies and environment, which is described next.

2.7 Collision detection

Collision detection is a well known problem in deformable or soft objects. When interactive speed is considered, the limitation of algorithm is increased. Many algorithms are based upon types of bounding volume hierarchies and spatial subdivision. For bounding volume hierarchies, some examples are bounding spheres [71,72,73,74], axis-aligned bounding boxes (AABBs) [75,76], oriented bounding boxes (OBBs) [77], quantized orientation slabs with primary orientations (QuOSPOs) [78], and discrete-oriented polytopes (K-DOPs) [79]. CLOD with dual hierarchy [80] has been also proposed for polyhedral objects whereas Separation-sensitive collision detection [81] was presented for convex objects. For spatial subdivision, octtree [82], BSP tree [83], brep-indices [84], k-d tree [85], bucket tree [86], and uniform spatial subdivision [87] are proposed. The GPU processor also has been used to detect collision for complex models [88,89,90]. Some of the relevant works in collision detection are briefly explained here.

Axis-aligned bounding box (AABB) presented in [75] defines the minimum point and maximum point to bound an object. AABB tree for deformable object is presented in this research. The binary tree is created for AABB trees to speed up the performance by recursive subdivision. The subdivision is chosen by the longest axis of the AABB, called fat subdivision

(cube-like) in which better performance can be accomplished for the intersection test. The subdivision continues until the subset in the AABB tree contains one element. For n primitives in the deformable objects, the AABB tree contains n leaves and $n-1$ internal nodes. The result shows that it is fast to build the tree but it is slow for test for collision compared to other tree structure such as Octree.

Oriented bounding box (OBB) in [77] is the data structure for exact interference detection for long shape object. The oriented bounding box is created to tightly bound the long and thin model based on the orientation of the object. A new separating axis theorem is proposed to test the overlapping between oriented boxes. The OBB provide the tight fit for long and thin objects. The OBB in tree structure algorithm is implemented by using two steps. First a tight-fitting OBB is placed around the objects and second OBB is nested into tree hierarchy. The benefit of this algorithm is that the performance can significantly improved for long-thin shape of models compared to AABB tree. However, it is computational expensive to build the tree in deformable object simulation because the orientation of the object changes over time.

An Octree applied to a bucket tree is presented by Ganovelli *et al.* [86] to construct a tree structure in which root of the tree is attached to AABB of the object, and each leaf on the tree contains a set of primitives for a corresponding box or a bucket. The bucket index can be identified by 3 values of n_x, n_y, and n_z: $0 <= n_x, n_y, n_z < 2^l$ where l is the level on the tree. In every time step, coordinates of AABB are changed. As a result, the tree must be reconstructed and all primitives are replaced into the correct buckets. Then, the collision detection is tested. If there is an overlap between non-leaf node, the children of the non-leaf node in the smaller volume is tested against the one in the bigger. If overlap occurs between leaf nodes, the set of

primitives in the bucket are tested for collision. However, the octree in this proposed method requires updating during the simulation which is a disadvantage of tree structure in collision detection. In the worst case, if a deformable object has m primitives where $m-1$ primitives are very close to each other and one primitive is far away. Then, one bucket could have $m-1$ primitives where as another one could have only one primitive. Spatial hashing is widely used for collision detection for deformable objects and rigid bodies.

Spatial hashing is widely used for collision detection for deformable objects and rigid bodies. The primitives are placed into the hash table by using hash function with respect to x, y, and z coordinates of primitives. This method can be used and adapted to any type of primitives. In [11], spatial hashing and a bounding box are used for collision detection in deformable objects and rigid bodies. A deformable object is bounded with AABB bounding box and then grid cells are created for spatial hashing. Each primitive is placed into the hash table. Then collision detection is performed for all primitives in the same hash index.

Since the tetrahedral mesh is common used in deformable objects, optimized spatial hashing is proposed for tetrahedral meshes in [92]. Each vertices and tetrahedrons with respect to small AABB bounding boxes is map to hash table index. For spatial hashing of vertices, the coordinates of the vertex position are hashed with respect to the given cell size. Then, the vertex will be placed into hash table according to hash index. For spatial hashing of tetrahedrons, the minimum point and maximum point of AABB are to cover a tetrahedron in the second pass. All grid cells are in AABB are hashed in to hash table. To get hash index, hash function uses the operation, XOR, with the big random prime numbers for better distribution in the hash table.. Thus, each hash table contains a number of object primitives. Then, collision detection is

performed among the number of the primitives in the same hash table index. Because each hash table index contains the primitives of itself and primitives of other objects, the collision and self collision can easily performed.

In interactive haptic rendering of high resolution deformable objects [93], Galoppo et al. have presented the efficient method for the force computation of contact that interacts with the haptic force feedback in haptic rates. They present the contact detection or collision detection between soft body and haptic device in three main steps. First they perform the query for the collision detection in low-resolution polygon proxies. Each convex proxies has only few vertices. Second they refine the query by using higher resolution of the soft body. They also develop an algorithm based on the image-space to compute direction of penetration depth. Finally, they check for the high resolution skin surface of collision and compute collision response if collision occurs on the area of the surface.

The background of collision detection has been presented in this section. In the next section, we explain the constraints that occur in deformable object simulation.

2.8 Constraints

Several constraints, such as volume control, surface deformation, internal pressure, gravity, and contact surface, influence the deformations of the objects. In this section we describe how these constraints can be manipulated in the simulation of deformable objects.

2.8.1 Volume control

The volume control significantly affects the physical behaviors of the models. The nonlinear and linear systems in FEM, mass-spring, modal analysis, etc. have been used to generate volume control in deformable objects.

In interactive deformation using modal analysis [90], the modal analysis takes the nonlinear description of the system. This analysis finds a good linear approximation that is mapped to the coordinate system, representing the volume of the model. The instability problem in numerical analysis can be eliminated by this modal analysis because each of decoupled equations is solved analytically. The volumetric tetrahedral mesh is generated for the model structure and a linear set of equations is applied to describe the system's behavior. Then the system interacts with a rigid body, depending on the contact or inertial forces.

For a versatile and robust model for geometrically complex deformable solids in [91], the deformable solids are discretized into tetrahedral and mass points for dynamic behaviors of the models. The forces at the mass points are derived by the potential energies that preserve the surface area and volume of the objects. Then the weighted stiffness coefficients describe the material properties of the deformable objects. The deformation behavior is a combination of elasticity and plasticity where the FFD is applied on tetrahedral meshes to define the volume of the objects.

The meshless deformation in [54] presents the deformation based on shape matching. The meshless deformation preserves the model volume since the system has the original locations of the model points and then the model points are mapped to the goal locations that are calculated to satisfy the volume preservation of the model. The deformations have been described in forms

of linear, quadratic, and cluster based deformations. The linear transformation only represents shear and stretch, and the quadratic deformation extends the motion of twist and bending behaviors. Besides those deformations, the cluster based deformation extends the range of motion to the goal positions that respect to the clusters of model points.

2.8.2 Surface deformation

Unlike the volume control, the surface deformation does not significantly affect the physical behaviors of the models. The surface deformation is the internal energy that applies only on the surface of the model to present the deformed appearance of the model surface.

To increase the visual realism of the model surface in modal analysis, the normal correction has been added in DyRT [51] that uses an approximation of neighboring vertex information for dynamic elastic in FEM. Since the normal correction adds memory cost into the system, correcting normals are applied only on some particular modes such as the dominant and torsional modes.

In interactive virtual material [92], two representations are used in the same object to increase the appealing quality of the simulation. These representations include low resolution volumetric mesh for the FEM simulation and high resolution surface mesh for rendering. Since the surface deformation does not provide the physical behaviors of the modes and low resolution produces enough information to simulate the model behavior, the mesh coupling and consistent watertight fracturing are applied only on the surface mesh to provide the surface detail.

In PriMo [93], the surface deformation is presented by a layer of rigid prisms to develop the mesh face of the model. This simulation provides robust and physically plausible large-scale

deformation that still preserves the surface detail. The prism representation connected to the elastic points is used to avoid instability of large deformation in FEM. Local and global shape matchings provide the local deformation at each prism and global deformation of the entire model.

Soft layer of articulated characters in [94] is presented by linear elasticity theory and linear FEM that formulate the displacement field of the soft layer in pose space. Linear elasticity provides the elastic energy that affects on the soft skin layer, not on skeleton because the deformation of skeleton is simulated by pose space of the model. Then the collision detection of surface contact only operates on the local area that contact occurs on model surface.

2.8.3 Internal energy

The methods, FEM, mass-spring system, or modal analysis, in volume control can be also used to generate the internal energy for model deformation.

In modal analysis in [51], the bone-based animation is designed for realistic modal deformation. The motion movement of the body is captured during the dynamics simulation and the movement is specified by bone transformation as rigid body. Then the correct modal forcing function is used for deformable objects as skin that attached to the bone.

The dynamics simulation in interactive deformation of modal analysis is discussed in [90] where the model system is embedded in a rigid-body reference frame to interact with each other though inertial effects. The modal decomposition applies a set of linear equations that describe the behavior of the system. The internal pressure is generated by the system's stiffness, damping, and mass matrices that are applied to the piecewise-tetrahedral finite elements.

The element-based warped stiffness is used in [92] to remove the artifacts under large deformation in the linear elastic system that controls the internal energy in the model. The problem has been solved by allowing the integration of the plasticity model that apply to each tetrahedral to generate internal energy of the model that does not produce the artifacts under large deformation. The method provides stability and speed into the system

2.8.4 Gravity

Gravitation force is an external force that is acting on vertical direction of each object primitives. By gravitational forces, the deformable objects are moved downward along y direction which causes the deformable objects to collide with other objects or environments [94]. This gravity refers to the weight of the objects and gravitational field to generate the realism in physics which is essentially used in physically-based modeling [9].

Gravity introduces the large deformation that causes the unrealism in the simulation. This problem is avoided by modifying (lowered) gravitational field in the simulation system [53]. Gravitation force has been used to demonstrate the level of deformations if the one end of the deformation object is fixed and another end of the object deforms under the influence of the gravitational force [95].

In stable real-time deformations [38], non-linear, warped, and linear strain measures have been used to simulate three bars that have one ends attach to a wall and then the gravity pulls another ends downward. This simulation shows that the linear strain measure (linear elasticity) that is commonly used in real time simulations can cause the large deformation under gravity and other conditions. The problem is solved by using the stiffness warping scheme that uses global

rotation component for the body frame and local matrices for every model vertex. The results of the simulation show that the method is stable and robust in large deformation under gravitational force.

2.8.5 Contact constraints

The constraints of collision include the constraints that occur from touching, poking, or pinching. Several collision response methods have been proposed to handle the contract constraints.

The collision response in [96] uses the penalty-based and constraints-based methods for interactive deformation in modal analysis. The penalty-based leads to instability problem but fast, while the constraints-based solves instability problem but require more computational time. Thus, this modal analysis uses both methods to handle the collision to provide both fast and sable to the system.

In [97], the contact representation in point based animation of elastic, plastic, and melting occurs when an object is melted and flows into another objects, such as solid objects. For collision response, the colliding surface elements in deformable objects are projected onto Moving Least Square (MLS) surface of the solid object. The normal vectors of the colliding surface elements are set to the normal vectors of the MLS on the solid surface to respond to the collision.

In [31], the self collision response is presented in position based dynamics by using spatial hashing presented in [92] to find vertex triangle collisions in cloth simulation. The collision detection method checks between a vertex and a triangle to ensure that the vertex does

not move through the triangle. The collision constraints are referred to the position of the vertex, the positions of the triangle, and the thickness of the cloth. Then, the collision response uses these constraints to maintain the original side of the cloth vertices.

In [47], the collision response for contact constraints is formulated by velocity constraints of the collision area. The method of Lagrange multipliers solves the implicit motion equation which guarantees the effectiveness of the collision response that acts on the skeletal motion. The collision detection performs in the low-resolution proxies and the high-resolution skin surface to collect all colliding skin vertices. Then, the computation of bone and skin response is computed by condensed contact constraints to provide the appearance detail of the contact surface.

The background of soft body simulation has been presented. In the next chapter, we discuss the definitions of rigid body and soft body in detail.

CHAPTER 3: A GENERAL MODEL OF SOFT BODIES

The major objective of this research is to present a general model for deformable objects. In our observation, a soft body is generally based on body control, fluidity control, volume control, constraints, and gravity. Based on this fact, we propose a parametric based general formal soft body model utilizing these components. The body control preserves the structure of the model whereas fluidity control determines the deformation of the model surface. The volume control defines the variability in the volume of the model. Constraints may impose special relations among these three components. The effect of gravity is determined by the gravity component. In order to show that the proposed general model encompasses some other existing models, we map this general model to some existing models. This mapping indicates that many models are specific cases of the proposed general model.

In our general model, specific methods are selected to implement the general model. Other specific methods can also be selected to implement the proposed general model. For the specific methods, the soft body 3D model is defined by a triangular mesh of surface points and the volume inside the soft body is maintained by internal pressure. Mass-spring forces, fluid forces, and gravitational forces are applied to each surface point to model the soft body. Instead of considering all points inside the soft body, internal pressure force maintains an approximated volume without explicit internal structure. This approach significantly reduces simulation time required to determine the configuration of the soft body. In addition, we also present a partitioning and hashing scheme to reduce simulation time of both fluid modeling and collision detection.

Consider two general approaches to deform a triangle mesh: (1) the internal pressure soft body method [98] and (2) the fluid-based soft body method [63,64]. Each method has both benefits and drawbacks. The internal pressure model with a mass spring system can generate smooth surfaces, which are suitable for balloon-like models. However, a drawback is that the wavy surfaces of liquid-like substances, such as silicone gel, cannot be simulated by this method. The main limitations of the fluid model are (1) fluid particle physics is computationally expensive, and (2) the soft body surface must be reconstructed every animation frame by the Marching Cubes algorithm, which can be prohibitively expensive to fast compute as mesh detail increases.

To obtain the benefits of both models while avoiding the drawbacks, the method proposed in this research utilizes a hybrid model that (1) applies internal pressure force to a mass-spring system surface, and (2) applies fluid equations to more realistically deform the soft body surface. Thus, rippling fluid-like surfaces can be simulated while maintaining the interactive rendering speeds of a pre-defined surface mesh.

This chapter is structured as follows. Section 3.1 describes the definitions of rigid body and soft body. Section 3.2 discusses the definitions of force parameters. Section 3.3 presents how to map the proposed general model to some specific models. Section 3.4 shows a specific soft body model. Finally section 3.5 discusses the physics motion in the soft body.

3.1 Definitions of rigid body and soft body

This section formally describes the 3D soft body motion model. First, the rigid body in motion, $R3D$, is defined as:

$R3D = \langle O, F, V \rangle$,

where O is the object defined by a list of points and other components, such as edges, triangles, and tetrahedrons,

$P = \{p_i | i = 0,..,n-1\}$ is a set of surface points of the rigid body and (x_i, y_i, z_i) is a coordinate of surface point p_i,

for example if the object is defined by a list of edges and triangles then

$E = \{p_i p_j | p_i, p_j \in P\}$ is a set of edges that connect p_i and p_j,

$T = \{p_i p_j p_k | p_i, p_j, p_k \in P$, and $| p_i p_j, p_i p_k$, and $p_j p_k \in E\}$ is a set of triangles of rigid body surface defined by p_i, p_j, and p_k,

$F = \{\boldsymbol{F}, \boldsymbol{F}_g\}$ is a set of forces consisting of $\boldsymbol{F}$, an external force, and $\boldsymbol{F}_g$, a gravitational force, applied at the center of the rigid body, and

V is the velocity at the center of the rigid body.

Correspondingly, we formally define the 3D soft body in motion, $S3D$, as

$S3D = \langle O, F, V, N, C \rangle$,

where O is for the soft body as defined in the rigid body model,

$F = \{\boldsymbol{F}_i | i = 0,..,n-1\}$ is a set of forces for each surface point of the soft body,

$V = \{\boldsymbol{v}_i | i = 0,..,n-1\}$ is a set of velocities for each surface point of the soft body, and

N is a number proportional to the number of molecules inside the soft body to maintain the approximated volume of the soft body, and

C is a set of deformation constraints for various types of the object being modeled for specific applications.

If the model uses the structure of a triangle mesh in 3D rigid or soft body in motion, we also define $l_{ij} = |p_i p_j|$, length of the edge $p_i p_j \in E$ and $a_{ijk} = |p_i p_j p_k|$, area of the triangle, $p_i p_j p_k \in T$. l_{ij} and a_{ijk} are constants for rigid body whereas l_{ij} and a_{ijk} change in soft body due to deformation and interaction with environment. In this model, it is assumed that the number of surface points, edges, and triangles remain constant after any deformation.

3.2 Definitions of force parameters

We propose several forces at each surface point for soft body. These forces are due to the following assumptions.

1) The body control is generated by the structure of the model that connects surface points for primitive structure defining the model. For example, a mass-spring system can be applied on the edge of surface points p_i and p_j. The edge is represented by a spring, the length of which varies depending on the deformation of the soft body.
2) For the fluidity control, each surface point is a free moving particle which interacts with nearby neighboring surface points within a radius by fluid force.
3) In the volume control, the molecules inside the soft body can be used to exert internal pressure force on the surface area of the soft body. This force maintains an approximated volume of the soft body without explicitly computing each point in the 3D soft body.
4) Gravity is calculated at each surface point and applied to it.

To obtain a desired level of softness and deformation, we propose a parametric model of different forces. As indicated previously, the composite force, $\boldsymbol{F}_i^t$, of the surface point i at time t is based on components created by body control force, fluidity control force, volume control force, and gravity. With this assumption, $\boldsymbol{F}_i^t$ is defined as:

$$\boldsymbol{F}_i^t = \alpha \boldsymbol{F}_{bi}^t + \beta \boldsymbol{F}_{fi}^t + \gamma \boldsymbol{F}_{vi}^t + \delta \boldsymbol{F}_{gi}^t, \tag{3.1}$$

where $\boldsymbol{F}_{bi}^t$ is the force of the body control at surface point i, $\boldsymbol{F}_{fi}^t$ is the force of the fluidity at surface point i, $\boldsymbol{F}_{vi}^t$ is the force of the volume control at surface point i, and $\boldsymbol{F}_{gi}^t$ is the force of gravity at surface point i. Parameters, α, β, γ, and δ are parameters of body control, fluid control, volume control, and gravitational field, respectively, where $\alpha > 0$ and $\beta, \gamma, \delta \geq 0$. Constraints may exist for relations among these components.

The purpose of each parameter is as follows:

- The value of $\alpha > 0$ determines body control, enabling the model to deform while maintaining a relative configuration among the surface points. If body control (structure of the model) is significant for modeling, then the value of α may dominate among all parameter values.
- The value of $\beta \geq 0$ affects fluidity, such as pressure and viscosity of fluid force of soft body on the surface points. If fluidity is important for the model, then the value of β may dominate among all parameter values.

- The value of $\gamma \geq 0$ influences body volume as well as deformation due to the molecules inside. If the volume of the model is important for the model, then the value of γ should be carefully determined.
- The value of $\delta \geq 0$ simulates the soft body with or without the impact of gravitational force. $\delta = 0$ is for no gravity.

In this manner, the various force parameters exerted on each soft body surface point can be adjusted to obtain specific types of soft body behaviors.

The velocity generated by the force $\boldsymbol{F}_i^t$ is defined as:

$$\boldsymbol{v}_i^t = \alpha \boldsymbol{v}_{bi}^t + \beta \boldsymbol{v}_{fi}^t + \gamma \boldsymbol{v}_{vi}^t + \delta \boldsymbol{v}_{gi}^t, \qquad (3.2)$$

where $\boldsymbol{v}_{bi}^t$ is the velocity from body control at surface point i, $\boldsymbol{v}_{fi}^t$ is the velocity from fluidity control at surface point i, $\boldsymbol{v}_{vi}^t$ is the velocity from volume control at surface point i, and $\boldsymbol{v}_{gi}^t$ is the velocity from gravity at surface point i.

3.3 Mapping the general model to some specific models

In this section we demonstrate that the proposed general soft body method can map to other existing models to simulate variety of soft bodies by selecting appropriate components and parametric values. Within the soft body, the forces generated from the selected components and parametric values are combined to create a realistic simulation of the soft bodies. As presented in previous section, the combined force, $\boldsymbol{F}_i^t$, is evaluated by:

$$\boldsymbol{F}_i^t = \alpha \boldsymbol{F}_{bi}^t + \beta \boldsymbol{F}_{fi}^t + \gamma \boldsymbol{F}_{vi}^t + \delta \boldsymbol{F}_{gi}^t$$

, where $\boldsymbol{F}_{bi}^t$,is the internal force for body control, $\boldsymbol{F}_{fi}^t$ is the internal force generated by fluidity control, $\boldsymbol{F}_{vi}^t$ is the internal force for volume control, and $\boldsymbol{F}_{gi}^t$ is the external force from gravitational field. α, β, γ, and δ are parameters for body control, fluidity control, volume control, and gravitation, respectively.

Since our soft body model is designed for a specific case of most existing models, in this section we map our general model to some specific models. Seven model mapping are discussed here.

3.3.1 A versatile and robust model for geometrically complex deformable solids

The deformable object in [42] is designed for the model both 2D deformable objects and 3D deformable objects with high deformation by FFD. The solid model is tessellated into tetrahedrons and mass points. For dynamic behavior, potential energies provide the combined forces at each mass-point. The constraints of stiffness and damping coefficients are considered for the potential energy, which depends on the mass point positions and stiffness coefficient. The coefficient defines the type of potential energy and if the body is in non-deformed state, the potential energy becomes zero. The potential energy, E, is evaluated by:

$$E(p_0,..., p_{n-1}) = \frac{1}{2} kC^2 \qquad (3.3)$$

, where k is a stiffness coefficient, C is a set of constraints defined for this model, and p_i is the mass points of the model. Then the force at mass point p_i incorporated with damping force to improve the robustness of the dynamic simulation is defined as:

$$\boldsymbol{F_i}(p_0,...,p_{n-1},\boldsymbol{v}_0,...,\boldsymbol{v}_{n-1}) = (-kC - k_d \sum_{0 \le j < n} \frac{\partial C}{\partial p_j}\boldsymbol{v}_j)\frac{\partial C}{\partial p_i} \tag{3.4}$$

, where $\boldsymbol{v}_i$ is the velocity at a mass point and k_d is damping coefficient.

In this model, the distance preservation considers all pairs of mass points that are connected by the tetrahedral edges. The potential energy for distance preservation, E_D, is computed by:

$$E_D(p_i, p_j) = \frac{1}{2}k_D\left(\frac{|p_j - p_i| - D_0}{D_0}\right)^2 \tag{3.5}$$

, where D_0 is the rest or initial distance of the tetrahedral edge and $D_0 \neq 0$. Then the force for distance preservation, F_D, is evaluated as mentioned in (3.4).

The surface area preservation depends on the energy which is evaluated by the difference of current area and initial area of the model triangles. The potential energy for surface area preservation, E_A, is defined as:

$$E_A(p_i, p_j, p_k) = \frac{1}{2}k_A\left(\frac{\frac{1}{2}|(p_j - p_i)\times(p_k - p_i)| - A_0}{A_0}\right)^2 \tag{3.6}$$

, where A_0 is the initial and rest area of a triangle and $A_0 \neq 0$. Then the force for the area preservation, F_A, is evaluated again as mentioned in (3.4).

The volume preservation considers the mass point of a tetrahedron derived from the difference of the current volume and the initial volume of the tetrahedron. The potential energy for volume preservation is described as:

$$E_V(p_i, p_j, p_k, p_l) = \frac{k_v}{2} \frac{(\frac{1}{6}(p_j - p_i) \cdot ((p_k - p_i) \times (p_l - p_i)) - V_0)^2}{\tilde{V}_0^2} \quad (3.7)$$

, where $\tilde{V}_0 = V_0$ if the models are the volumetric tetrahedral meshes and V_0 is assume not to be zero but if the models are plates or discrete shells, then $\tilde{V}_0 = \frac{\sqrt{2}}{12}\bar{l}$, where $\bar{l}$ is the average edge length of a tetrahedron. Then the force for the volume preservation, F_V , is defined as stated in (3.4).

Finally, the combined force is computed as $\boldsymbol{F}^t = \boldsymbol{F}_D^t + \boldsymbol{F}_A^t + \boldsymbol{F}_V^t$. To map this model, the body control is the forces that are derived from the distance preservation and area preservation ($\boldsymbol{F}_D^t + \boldsymbol{F}_A^t$) in (3.5) and (3.6). The volume control is the force evaluated by the volume preservation ($\boldsymbol{F}_V^t$) in (3.7). The constraints in this model are specified by $C(p_0,...,p_{n-1})$. Thus, the parameters for this specific model are $\alpha = 1, \beta = 0, \gamma = 1$, and $\delta = 1$.

3.3.2 A fast, flexible, particle-system model for cloth draping

In a flexible particle system model for cloth draping [26] , the model structure is defined by lists of particles and edges that connect among those particles. Cloth properties such as anisotropic behavior and hysteresis are specified for the forces that are generated to interact with environment. The measured force data is introduced for specific cloth properties for the effect on

falling cloth from air resistance, wind, moving bodies, and surface friction. The model has been described as the masses of the particles which have the properties of positions, velocities, and forces. The Lagrange equations provide the trajectory of the moving particles for this model. The forms of force plots of tension, shearing and bending describe the internal energy of the cloth model. Consider a particle p_0 at location $p_0 = (x_0, y_0, z_0)$ with four neighbors p_1, p_2, p_3, p_4. By approximation of the plots with piecewise linear functions, the sharing and bending energies in this model are given by:

$$E_s = \sum_{i=1}^{4} \frac{1}{2} C_s (\varphi_i - \frac{\pi}{2} - h_{si})^2 \tag{3.8}$$

$$E_b = \sum_{i=1}^{2} \frac{1}{2} C_b (\psi_i - \pi - h_{bi})^2 \tag{3.9}$$

, where (C_s, C_b) and (h_s, h_b) are the slopes of the plot in weft and warp directions, φ is a set of angles formed around the four-connected neighbors of particles i, and ψ is a set of angles formed by the segment of connecting particles i and its nearest horizontal and vertical neighbors. Then, the tension energy on the cloth surface is implemented as:

$$E_t = \begin{cases} \sum_{i=1}^{4} \frac{1}{2} C_{i,t1} ((p_0 - p_i) - d_i - h_{ti})^3 & if \ \ p_0 - p_i \geq d_i \\ \sum_{i=1}^{4} \frac{1}{2} C_{i,t2} ((p_0 - p_i) - d_i - h_{ti})^5 & if \ \ p_0 - p_i < d_i \end{cases} \tag{3.10}$$

, where d is the un-stretched distance between particles p_0 and p_i. The tension energy distinguishes between repelling and stretching forces on the textiles.

For the Lagrange function, the following energy, E_{kin}, is required:

$$E_{kin} = \frac{1}{2} m_i \cdot v_c^2 \tag{3.11}$$

, where m_i is a mass at particle p_i, and v_c is velocity in weft and warp directions.

Also, the potential energies, E_{pot}, is:

$$E_{pot} = m_i \cdot \boldsymbol{g} \cdot z_0 \tag{3.12}$$

, where $\boldsymbol{g}$ is gravitational field.

From (3.8) through (3.12), the energies for particles with four neighbors are defined. For Lagrange function, all the energies for n particles must be added and the redundant energies for particles with fewer than four neighbors must be eliminated.

Then the air resistance and external force, $\boldsymbol{F}_{air}$, for the particles at the edge of cloth is computed by:

$$\boldsymbol{F}_{air} = \frac{1}{2} \cdot \rho \cdot \boldsymbol{c}_w \cdot A \cdot \boldsymbol{v}_i^2 \tag{3.13}$$

, where ρ is specific weight of air, c_w is the resistance coefficient, and A is the surface perpendicular to the velocity $\boldsymbol{v}_i$.

In this model, the body control is specified by the sharing and bending energies in (3.8) and (3.9). Then the tension and potential energies in (3.10) (3.11), and (3.12) are also included for the body control with the gravitational field (in potential energy). To control the body movement along the edge, the formulation of air resistant in (3.13) has been added. Body control and gravitational field components of the proposed soft body model are used to create this cloth model for draping. The constraints are defined for the effects of draping along the edge of the cloth. Thus, the parameters for this specific model are $\alpha = 1$, $\beta = 0$, $\gamma = 0$, and $\delta = 1$.

3.3.3 Estimating Cloth Simulation Parameters from Video

The cloth simulation from video in [27] presents the parameters of four different fabrics, linen, fleece, satin, and knit. The information data is collected from the video that captures the movement of fabrics. Then the movement of fabric is simulated using the information data and then the simulation is compared to the real movement of cloth in the video data. The energy of the cloth model for different types of fabric is applied on the model points, edges, and triangles. The condition functions for the internal energy associate with the stiffness and damping coefficients. Within the triangle area, the energy distributes linearly in terms of condition function $C(x)$. The energy of a particular triangle, E_u, is defined by

$$E_u = \frac{k_s}{2} C(x) C^T(x) \tag{3.14}$$

, where k_s is a stiffness coefficient in a particular condition function. Then the forces are given by:

$$\boldsymbol{F} = \nabla_x E_u \ . \tag{3.15}$$

Then the damping forces, d, are computed in the term of the condition function, $C(x)$, as:

$$d = -k_d \frac{dC}{dx} C(x) \tag{3.16}$$

, where k_d is the damping coefficient of the system.

To capture the dynamic aspects of cloth, the simple non-linear drag force has developed a function of non-linearity, where a linear function of tangential velocity and a quadratic function of normal velocity are combined with k_f that controls the degree of non-linear function. Then the drag force, $\boldsymbol{F}_{drag}$, is expressed by:

$$\boldsymbol{F}_{drag} = -a\left(\frac{k_N |\boldsymbol{v}_N|^2}{1 + k_f |\boldsymbol{v}_N|^2}\frac{\boldsymbol{v}_N}{|\boldsymbol{v}_N|} + k_T \boldsymbol{v}_T\right) \tag{3.17}$$

, where a is the area of the given triangle, the average velocities on the face are from two components, one normal to the surface $\boldsymbol{v}_N$ and one tangential $\boldsymbol{v}_T$, and drag coefficients k_N and k_T are acting on the normal of the surface and the triangle of the surface.

The gradient mask, M_k, of each video for each frame of video is given by:

$$M_k(i,j) = \begin{cases} 1, \|\delta(i,j)\| \geq \tau \\ 0, \|\delta(i,j)\| < \tau \end{cases} \tag{3.18}$$

, where τ is the threshold defined by a user, and $\|\delta(i,j)\|$ is the magnitude of the gradient of the angle map at (i,j).

For fold comparison, the frame by frame sum of squared differences (SSD) between masked angle that maps in simulation with video data is computed by the metric to generate the errors of the comparison between the simulation data and the video data. The error at any particular frame k along the sequence, E_k^{fold}, is computed by:

$$E_k^{fold} = \sum_{i=0}^{S_x}\sum_{j=0}^{S_y} M_k(i,j)\cdot(\theta_k^{real}(i,j) - \theta_k^{sim}(i,j))^2 \tag{3.19}$$

, where S_x and S_y are the size of angle maps, θ^{real} are the angles from real data, and θ^{sim} are the angles from simulation data.

The silhouette comparison is an addition to the angle map. The penalty of silhouette mismatch is computed by the difference between the two silhouettes, i.e., the number of mismatched pixels. The penalty from silhouette comparison, E_k^{silh}, is computed by:

$$E_k^{silh} = \sum_{i=0}^{S_x} \sum_{j=0}^{S_y} \left| A_k^{real}(i,j) - A_k^{sim}(i,j) \right| \tag{3.20}$$

, where $A_k(i,j) = \begin{cases} 1, & inside\ silhouette \\ 0, & otherwise \end{cases}$.

The total error in frame k is:

$$E_k = E_k^{fold} + \alpha E_k^{silh}. \tag{3.21}$$

, where α is a user-defined weight that controls the relative contribution of two terms which is 0.1 in this simulation. Finally, the error across the entire sequence of N frames is presented by:

$$E = \sum_{k=1}^{N} E_k \tag{3.22}$$

This model defines the body control in cloth simulation by using the potential energies with condition functions in (3.14) through (3.17). The comparisons between the simulated cloth and video data in (3.18) through (3.22) provide the specifications of the simulation that are considered as the constraints of the simulation. Additionally, a gravitational field has been added for realistic simulation. Thus, the parameters for this specific model are $\alpha = 1$, $\beta = 0$, $\gamma = 0$, and $\delta = 1$.

3.3.4 Underwater cloth simulation with fractional derivatives

Underwater cloth simulation is presented in [32] where the point-based or particle-based cloth model with the list of points, edges, and triangles generates the cloth behavior and fractional derivatives depict the dynamics the cloth interacting with fluid. The technique involves fractional derivatives to improve the speed of computation. The damping force is derived from the viscoelastic elements that generate Basset drag in Stokes flows. This cloth simulation is

described as viscoelastic microstructure that has the behavior of fluid-like (viscous) to solid-like (elastic). The wave propagation on cloth shows the response between cloth and water property that has added in this model. For the force computation, the dynamic system is governed by Newton's second law of motion, which is:

$$\boldsymbol{F} = m\boldsymbol{a} \tag{3.23}$$

, where m is the mass and $\boldsymbol{a}$ is the acceleration of a particle.

In the model, the combined forces are the gravitational force, the buoyancy of water, a regular viscous drag, and the history drag. The gravitational force and the buoyancy are computed by:

$$\boldsymbol{F}_g = (\rho_c - \rho_w)V_c\boldsymbol{g} \tag{3.24}$$

where ρ_c is the density of a cloth particle, ρ_w is the water density, V_c is an estimation of the volume of one cloth particles, and $\boldsymbol{g}$ is the gravitational acceleration.

The regular viscous drag, $\boldsymbol{F}_{vd}$, yields:

$$\boldsymbol{F}_{vd} = -k_{vd}\dot{\boldsymbol{x}} \tag{3.25}$$

, where k_{vd} is the viscous drag coefficient, and $\dot{\boldsymbol{x}} = \boldsymbol{v}$ is the velocity of the particle.

The history drag is then computed for the inertial response with the fluid as:

$$\boldsymbol{F}_{hd} = -k_{hd}(D^{1/2}x \cdot \hat{\boldsymbol{v}})\hat{\boldsymbol{v}} \tag{3.26}$$

, where k_{hd} is the history drag coefficient, $\hat{\boldsymbol{v}} = \boldsymbol{v}/|\boldsymbol{v}|$ is the unit velocity vector, and $D^{1/2}$ is the half derivative term which is computed by:

$$D^{1/2}x_n = \frac{h}{6\sqrt{\pi}}\sum\left[\frac{\dot{\boldsymbol{x}}_{i-1}}{(nh-(i-1)h)^{1/2}} + + \frac{2(\dot{\boldsymbol{x}}_{i-1}+\dot{\boldsymbol{x}}_i)}{(nh-(i-1/2)h)^{1/2}} + \frac{\dot{\boldsymbol{x}}_i}{(nh-ih)^{1/2}}\right]$$

$$+\frac{0.15h}{\sqrt{\pi}}\left[\frac{\dot{\boldsymbol{x}}_{\boldsymbol{n}-1}}{h^{1/2}}+\frac{2(\dot{\boldsymbol{x}}_{\boldsymbol{n}-1}+\dot{\boldsymbol{x}}_{\boldsymbol{n}})}{(0.55h)^{1/2}}+\frac{\dot{\boldsymbol{x}}_{\boldsymbol{n}}}{(0.1h)^{1/2}}\right]+\frac{0.05h}{\sqrt{\pi}}\left[\frac{8\sqrt{2}}{3}\frac{\dot{\boldsymbol{x}}_{\boldsymbol{n}}}{(0.05h)^{1/2}}-\frac{4}{3}\frac{\dot{\boldsymbol{x}}_{\boldsymbol{n}}}{(0.1h)^{1/2}}\right] \tag{3.27}$$

, where h is the time step, i is time step index, and n is the index of most recent computed time step.

Then, the spring force, $\boldsymbol{F}_s$, added on the particle i by the fractional viscoelastic spring element to connect particles i and j is modeled as:

$$\boldsymbol{F}_s=-k_s(|l|-r)\frac{l}{|l|}-k_d\left[\frac{(D^{1/2}x_i-D^{1/2}x_j\cdot l}{|l|}\right]\frac{l}{|l|} \tag{3.28}$$

, where k_s is the linear spring constant, k_d is the viscoelastic damping constant, r is the rest length of spring, and $l=x_i-x_j$ is the length of spring between particles i and j.

In this underwater cloth simulation, the body control is defined by the mass-spring systems with the fractional viscolastic spring element. The half derivative term in (3.27) supports the body control to provide more accurate approximation which has been used for both drag force and spring force in (3.25) and (3.26). The Newton's second law of motion is presented for the motion of physics in (3.23) and has included the interaction of water using the buoyancy in (3.24), which is considered to support fluid modeling. Thus, the parameters for this specific model are $\alpha=1$, $\beta=1$, $\gamma=0$, and $\delta=1$.

3.3.5 Position based dynamics

In position based dynamics [31], model points are connected with neighboring points by the edges of triangles. The velocity layer in this model has been omitted and the positions are directly computed during simulation. Instability problems in the explicit integration method can

be avoided and controllability can be improved by using this position based dynamics. The collision constraints are simply handled by projecting positions of the points to the correct locations. The simulations of cloths and inflated characters show the system stability and effectiveness of the collision and self collision.

In this model, a set of N vertices, M constraints, and a mass m_i at vertex i are defined. A constraint $j \in [1,...,M]$ is composed of a cardinality n_j, a function $C_j : \Re^{2n_j} \to \Re$, a set of indices $\{i_i,...,i_{nj}\}, i_k \in [1,...,N]$, a stiffness parameter $k_j \in [0...1]$, and a type of either *equality* or *inequality*. The equality is defined by $C_j(x_{ij},...,x_{in_j}) = 0$ and the inequality is defined by $C_j(x_{ij},...,x_{in_j}) \geq 0$.The strength of the constraints is determined by the stiffness parameter k_j, which has the range of zero to one. In each frame of the simulation, the solver function runs multiple times to find the new locations of the points that satisfy all of these constraints. The constraint projection scheme moves a set of points according to a constraint specified for the simulation. Let Δp_i be the displacement of vertex i by the projection. Linear momentum is conserved if:

$$\sum_i m_i \Delta p_i = 0 \tag{3.29}$$

, where m_i is the mass at vertex i.

The angular momentum is conserved if:

$$\sum_i r_i \times m_i \Delta p_i = 0 \tag{3.30}$$

, where r_i is distance of vertex i to an arbitrary common rotation center.

For each p, the correction Δp needs to satisfies $C(p + \Delta p) = 0$ which can be approximated by

$$C(p + \Delta p) \approx C(p) + \nabla_p C(p) \cdot \Delta p = 0 \tag{3.31}$$

solving for Δp between two points p_1 and p_2, the final correction is computed as:

$$\Delta p_1 = -\frac{w_1}{w_1 + w_2}(|p_1 - p_2| - d)\frac{p1 - p2}{|p1 - p2|} \tag{3.32}$$

$$\Delta p_2 = -\frac{w_2}{w_1 + w_2}(|p_1 - p_2| - d)\frac{p1 - p2}{|p1 - p2|} \tag{3.33}$$

, where w_i is the inverse masses $w_i = 1/m_i$, and d is the distance between two points.

In cloth simulation, at each edge the stretching constraint is represented by the constraint function:

$$C_{stretch} = (p_1, p_2) = |p_1 - p_2| - l_0 \tag{3.34}$$

, where l_0 is the initial length of the edge.

Similarly, the bending constraint is defined by the constraint function:

$$C_{bend}(p_1, p_2, p_3, p_4) = acos\left(\frac{(p_2 - p_1) \times (p_3 - p_1)}{|(p_2 - p_1) \times (p_3 - p_1)|} \cdot \frac{(p_2 - p_1) \times (p_4 - p_1)}{|(p_2 - p_1) \times (p_4 - p_1)|}\right) - \varphi_0 \tag{3.35}$$

, where φ_0 is the initial dihedral angle between the two triangles.

In collision detection, if the vertex q moves through a triangle p_1, p_2, and p_3, the constraint function is defined as:

$$C(q, p_1, p_2, p_3) = (q - p_1) \cdot \frac{(p_2 - p_1) \times (p_3 - p_1)}{|(p_2 - p_1) \times (p_3 - p_1)|} - h \tag{3.36}$$

, where h is cloth thickness.

If the vertex moves from below with the respect to the triangle normal, the constraint function is computed by:

$$C(q,p_1,p_2,p_3)=(q-p_1)\cdot\frac{(p_3-p_1)\times(p_2-p_1)}{|(p_3-p_1)\times(p_2-p_1)|}-h. \tag{3.37}$$

The balloon effect for cloth modeling is also defined by the constraint function as:

$$C(p_1,...,p_N)=\left(\sum_{i=1}^{n_{triangles}}(p_{t_1^i}\times p_{t_2^i})\cdot p_{t_3^i}\right)-k_{pressure}V_0 \tag{3.38}$$

In this model, each point position has been directly computed for the new location by omitting the velocity layer. The constraint projection processes the new positions of the models according to the constraints which define the body control in (3.29) through (3.37) and volume control in (3.38). The new locations of points need to satisfy the constraints in (3.29) through (3.38) that are specified especially for this simulation. Thus, the parameters for this specific model are $\alpha=1$, $\beta=0$, $\gamma=1$, and $\delta=1$.

3.5.6 Soft Articulated Characters with Fast Contact Handling

Soft articulated characters with fast contact handling, as presented in [47], show the layered representation for articulated bodies. The volumetric mesh of the deformable layer can be implemented by any method, such as tetrahedral elements, that preserves the original outer surface points. The high-resolution deformable skin is built in real-time. The pose space of the skinned surface and collision queries presents the dynamic skeleton-skin deformation that interplays with other objects. The new formulation for elastic deformation in pose space has been proposed for the soft articulated characters by considering the skeletal-subspace deformation model with k bones. The deformed position x of a material point is based on the position u in pose space $T_{0,i}$ and bone transformations T_i as

$$x = \sum_{i=1}^{k} w_i T_i u_i = \left(\sum_{i=1}^{k} w_i T_i T_{0,i}^{-1} \right) u \tag{3.39}$$

, where blend weight w_i obeys the affine constraint $\sum_i w_i = 1$.

By substitute the deformation field in the previous equation with its discrete version, the position x becomes:

$$x = \sum_{i=1}^{k} w_i (c_i + R_i (c_{0,i} + R_{0,i} (u_0 + S q_s))) \tag{3.40}$$

, where c_0 and $c_{0,i}$ are constant transformations, R_i and $R_{0,i}$ are rotation matrices, S is shape matrix, and $\boldsymbol{q}_s$ is deformation field.

For the joint constraint, the constraints on the collision-free velocities are defined by:

$$J_\mu \boldsymbol{v}^- = -\alpha \boldsymbol{g}(q_b) \Rightarrow -J_j \Delta \boldsymbol{v}_b + \alpha \boldsymbol{g}(q_b) \tag{3.41}$$

, where J_μ is joint Jacobians, $q_b = \left[c_1^T \ \theta_1^T \ \ldots c_k^T \ \theta_k^T \right]^T \in \Re^{7k}$ are quaternions that represent the orientations θ, and $\boldsymbol{v}_b = \left[c_1^T \ \omega_1^T \ \ldots c_k^T \ \omega_k^T \right]^T \in \Re^{6k}$ is angular bone velocity ω expressed in the bone's local frame.

The contact constraint is then defined by:

$$J_\lambda (\boldsymbol{v}^- + \delta \boldsymbol{v}) = 0 \Rightarrow -J_b \delta \boldsymbol{v}_b - J_s \delta \boldsymbol{v}_s = J_b \boldsymbol{v}_b^- + J_s \boldsymbol{v}_s^- \tag{3.42}$$

, where $J_\lambda = \left[J_b \ J_s \right] \in \Re^{m \times (6k+3n)}$, m is the number of colliding surface nodes, k is the number of bones, n is the total number of surface nodes, and $\boldsymbol{v}_s$ is shape velocity.

This model uses a layered representation for body control in soft characters, which is integration of articulated body dynamics and skinning with displacement in (3.39) and (3.40). Soft contact with contact constraints defines dynamic simulation when the soft character is

contacted by other object. This contact constraints bases on three different components: coupled layered dynamics, joint constraints, and contact constraints in (3.41) and (3.42). Thus, the parameters for this specific model are $\alpha = 1,\ \beta = 0,\ \gamma = 1$, and $\delta = 1$.

3.3.7 PriMo: coupled prisms for intuitive surface modeling

In Primo [46], 3D shape modeling is presented by volumetric prisms coupled through non-linear for elastic deformation. To satisfy user constraints, local and global shape matching techniques are combined to provide the mesh deformation. The various geometrical parameters produce the control the behavior of the physical surface. In the prism representation, the face of p_i neighboring p_j is a rectangular bi-linear patch $f^{i\to j}(u,v),\ (u,v)\in[0,1]^2$, interpolating its four corner vertices $\left\{f_{00}^{i\to j}, f_{10}^{i\to j}, f_{01}^{i\to j}, f_{11}^{i\to j}\right\}$ and the opposite face is defined by $f^{j\to i}(u,v)\subset P_j$ Then the energy between p_i and p_j is computed as:

$$E_{ij} = \int_{[0,1]^2} \left\| f^{i\to j}(u,v) - f^{j\to i}(u,v) \right\| dudv \tag{3.43}$$

, corresponding to an integral over infinitesimal elastic forces.

Deformation energy is computed by the accumulation of pairwise energies E_{ij} as:

$$E = \sum_{\{i,j\}} w_{ij} \cdot E_{ij}\ ,\ w_{ij} = \frac{\left\| e_{ij} \right\|^2}{\left| F_i \right| + \left| F_j \right|} \tag{3.44}$$

, where w_{ij} is the weight at the area of the corresponding mesh faces F_i and F_j, and e_{ij} is the length of their shared edge .

For local shape matching, the faces $f^{i\to j}$ needs to match to the corresponding faces $f^{j\to i}$ to find the best rigid motion of rotation and translation (R_i, t_i) that yields a weighted pair-wise shape matching problem. Thus local shape matching is defined as:

$$\min_{R_i t_i} \int_{[0,1]^2} \left\| R_i f^{i\to j}(u) + t_i - f^{j\to i}(u) \right\|^2 du \,. \tag{3.45}$$

For the global shape matching, the matching process applies the formulation to each prism and updates until the system is converged. The first order approximations A_i of rigid motion (R_i, t_i) are defined in terms of linear and angular velocities v_i and ω_i by:

$$R_i(\cdot) + t_i \approx (\cdot) + \omega_i \times (\cdot) + v_i =: A_i(\cdot) \tag{3.46}$$

Then the energy minimization in terms of the approximation A_i is

$$\min_{\{v_i, \omega_i\}} \sum_{\{i,j\}} w_{ij} \int_{[0,1]^2} \left\| A_i(f^{i\to j}(u)) - A_j(f^{j\to i}(u)) \right\|^2 du \tag{3.47}$$

This method requires small updated steps, which is not appropriate for the large deformation. The proposed method projects the approximation A_i by finding the closest rigid motion (R_i, t_i), where the distances of transformations are compared their effects on the prism p_i. Thus the energy minimization becomes

$$\min_{R_i t_i} \int_{[0,1]^2} \left\| R_i f^{i\to j}(u) + t_i - A_j(f^{j\to i}(u)) \right\|^2 du \,. \tag{3.48}$$

In this model, a prism representation is used the structure of the model. The elastic force for body and volume control is expressed in (3.43) and (3.44). The elastic joint energy defines the deformation state between connected prisms. Local shape matching and global shape matching are considered as the constraints of this simulation because they need to be satisfied to

provide the local and global areas of deformation. The local shape matching and global matching tries to find the best rotation using energy minimization in sections (3.45) through (3.48) that give small errors to the model. Thus, the parameters for this specific model are $\alpha = 1$, $\beta = 0$, $\gamma = 1$, and $\delta = 1$.

This section has described how the existing models can be mapped into the proposed general soft body model. In the next section, the details of soft body are presented, where the mass-spring system, SPH, internal pressure have been selected for body control, fluidity control, and volume control for this specific model.

3.4 Specific methods of a soft body model

Our soft body model is defined by $M = \langle O,P,E,T \rangle$, where P, E, and T are the lists of points, edges, and triangles, respectively. The mass-spring system is selected to act on the mesh points, which are connected by springs, enabling the model to deform while maintaining a relative configuration among the surface points. SPH for fluid modeling creates a realistic fluid-like motion on the soft body surface. Volume inside the 3D soft body model is modeled by simulated molecules and ideal gas approximation to maintain volume. Constraints determine the properties of soft bodies, which is defined by specific applications. Gravitational force provides soft body motion from natural force. In this section, we present the model components that are suitable for our specific soft body model.

The surface elasticity is created by a mass-spring system in which surface mesh points move freely, yet retain a relative configuration to the original mesh. Mass-spring force $\boldsymbol{F}_{bi}^{t}$ is

determined by two parts, spring force $\boldsymbol{F}_{si}^{t}$ and damping force $\boldsymbol{F}_{di}^{t}$. Thus, $\boldsymbol{F}_{bi}^{t}$ is defined as follows [53,99]:

$$\boldsymbol{F}_{bi}^{t} = \boldsymbol{F}_{si}^{t} + \boldsymbol{F}_{di}^{t}, \tag{3.49}$$

where $\boldsymbol{F}_{si}^{t}$ is spring force at surface point i and $\boldsymbol{F}_{di}^{t}$ is damping force at surface point i.

In a mass-spring system, the s*pring force* generated by each spring depends on the *spring constant*, the *spring length*, and *spring rest position.* The equation for cumulative spring force among neighboring surface points is given by:

$$\boldsymbol{F}_{si}^{t} = \sum_{\forall j|(i,j)\in E} k_s (l_{ij}^{t} - l_{ij}^{0}) \frac{(p_j^t - p_i^t)}{l_{ij}^{t}}, \tag{3.50}$$

where $\boldsymbol{F}_{si}^{t}$ is the net internal spring force exerted on surface point i for every surface point j in the neighborhood E, k_s is the spring constant of the spring connecting surface points i and j, p_i and p_j are the positions of surface point i and j respectively, l_{ij}^{0} is the initial length of the spring between surface points i and j, and t is current time step.

Damping force works against the velocities of connected surface points in order to slow their relative velocities. The equation of cumulative damping force among a surface point's neighbors is given by:

$$\boldsymbol{F}_{di}^{t} = \sum_{\forall j|(i,j)\in E} k_d h(\boldsymbol{v}_i^{t} - \boldsymbol{v}_j^{t}), \tag{3.51}$$

where $\boldsymbol{F}_{di}^{t}$ is the damping force, k_d is the damping constant between surface point i and j, $\boldsymbol{v}_i$ is the velocity at surface point i, $\boldsymbol{v}_j$ is the velocity at surface point j, t is current time step, and h is time elapsed.

In order to model complex surface interactions, forces beyond mass-spring systems are required. For example the deformation of fluid-like on the surface of a deformed soft body. Since 3D fluid simulation is computationally expensive, we perform fluid calculation on the mass-spring surface of the soft body only while we generate the volume by internal pressure force which acts on each surface point.

For modeling behavior of fluids, we consider the impact of fluid pressure and fluid viscosity of the fluid. For that purpose, we simulate the surface points as free moving particles interacting with nearby surface points within a radius. Since the mass-spring system provides surface tension, we consider both fluid pressure force generated due to the density and fluid viscosity force generated due to the viscosity of the fluid. To model these forces in this simulation, we adopt the smoothed particle hydrodynamics (SPH) model developed originally for astrophysical problems and later used in interactive applications of particles based on fluid simulation [63,64]. In general, SPH is simple compared to other fluid modeling such as FEM or FVM. SPH is an interpolation method that distributes quantities in a local neighborhood of each particle using radial symmetrical smoothing kernels. We utilize poly6, spiky, and viscosity smoothing kernels in [63,64] to model fluid density, fluid pressure, and viscosity forces. Thus, during fluid force calculation, fluid density and fluid pressure are computed to generate fluid pressure force and fluid viscosity force as presented as follows.

In accordance with the model developed in [63,64], fluid density is given by:

$$\rho_i^t = \sum_j m_j W_{poly6}(l_{ij}^t, h_w), \ \forall j \text{ such that } l_{ij}^t \le h_w, \tag{3.52}$$

where ρ_i^t is the density at surface point i at time t, m_j is the mass at surface point j, and h_w is the core radius of SPH.

Next, fluid pressure is generated from fluid density as:

$$L_i^t = k(\rho_i^t - \rho^0), \tag{3.53}$$

where L_i^t is the fluid or liquid pressure at surface point i at time t, k is the gas constant, ρ_i^t is the density at surface point i at time t, and ρ^0 is the initial density.

Fluid pressure force at the soft body surface point i, $\boldsymbol{F}_{fpi}^t$, is computed as:

$$\boldsymbol{F}_{fpi}^t = -\sum_j m_j \frac{L_i^t - L_j^t}{2\rho_j^t} \nabla W_{spiky}(l_{ij}^t, h_w), \ \forall j \text{ such that } l_{ij}^t \le h_w, \tag{3.54}$$

where m_j is the mass at surface point j, L_i^t and L_j^t are fluid or liquid pressure values at surface points i and j respectively at time t, ρ_j^t is the density at surface point j at time t, and h_w is the core radius of SPH.

Finally the fluid viscosity force at the soft body surface point i, $\boldsymbol{F}_{fvi}^t$, is generated by:

$$\boldsymbol{F}_{fvi}^t = \mu \sum_j m_j \frac{\boldsymbol{v}_j^t - \boldsymbol{v}_i^t}{\rho_j^t} \nabla^2 W_{viscosity}(l_{ij}^t, h_w), \ \forall j \text{ such that } l_{ij}^t \le h_w, \tag{3.55}$$

where μ is the viscosity of fluid, m_j is the mass at surface point j, $\boldsymbol{v}_i^t$ and $\boldsymbol{v}_j^t$ are the velocities at surface points i and j respectively at time t, ρ_j^t is the density at surface point j at time t, and h_w is the core radius of SPH.

The fluid force, $\boldsymbol{F}_{fi}^{t}$, is the combination of two different forces; (1) fluid pressure force, $\boldsymbol{F}_{fpi}^{t}$, and (2) fluid viscosity force, $\boldsymbol{F}_{fvi}^{t}$. Hence, we define the fluid force as follows:

$$\boldsymbol{F}_{fi}^{t} = \boldsymbol{F}_{fpi}^{t} + \boldsymbol{F}_{fvi}^{t} \tag{3.56}$$

where $\boldsymbol{F}_{fpi}^{t}$ is fluid pressure force at surface point i and $\boldsymbol{F}_{fvi}^{t}$ is fluid viscosity force at surface point i.

In order to model volume of the soft body an internal pressure force must push the surface points outward. This volume is created by pressure force generated by the molecules within the soft body. Without volume, the soft body may become flat, much like fabric or cloth after colliding with the environment.

We are proposing a model for maintaining the volume of the soft body by internal pressure force. This model is conceptually similar to the idea presented in [98]. The internal pressure force is defined as:

$$\boldsymbol{F}_{vi}^{t} = \sum_{\forall jk(i,j,k)\in E} a_{ijk}\hat{n}\frac{1}{B}N, \tag{3.57}$$

where $\boldsymbol{F}_{vi}^{t}$ is the net pressure force in bounding box volume of soft body experienced at surface point i at time t, a_{ijk} is the surface area of the face connecting surface point i to all surface point pairs (i,k) in the neighborhood of i, $\hat{\mathbf{n}}$ is a normal vector to the surface where the pressure force is acting, B is the volume of the soft body bounding box at current frame, and N is proportional to the number of molecules inside the soft body to maintain the approximated volume of the soft body.

In the real-world gravity affects all objects, thus the simulation model must account for it. The force of gravity experienced by an object earth is equal to weight of the object experiencing the gravitational pull. Gravitational force at a point on the soft body surface is computed by:

$$\boldsymbol{F}_{gi}^{t} = m_i \, \boldsymbol{g} \,, \tag{3.58}$$

where $\boldsymbol{F}_{gi}^{t}$ is gravity force at time t acting on surface point i of mass, m_i, and $\boldsymbol{g}$ is acceleration due to Earth's gravity.

When computing the sum of all individual forces on a soft body, the forces must be combined in this step for each surface point. The combination of all forces proportionally selects of particular forces suitable for specific applications.

3.5 Physics motion in the soft body

After the effective application at each surface point, we determine the new position of the surface point due to the motion physics for the next animated frame. There are several standard numerical methods for performing soft body motion updates. In general, explicit, implicit, and trapezoid Euler methods have been used to approximate the motion of soft bodies. Explicit methods require constraints, such as time step or spring constant, should be small in each animation step because the simulation may become unstable when the large constraints are applied. Thus, they are not suitable for applications that require stability and accuracy. These problems have been addressed in [1,53,100].

To avoid stability issues, we use the implicit method in [100] to update soft body motion. In our model, velocity is generated from combination of all forces. Thus, the velocity at surface point i at time $t+h$, $\boldsymbol{v}_i^{t+h}$, is calculated by:

$$\boldsymbol{v}_i^{t+h} = \boldsymbol{v}_i^t + \Delta \boldsymbol{v}_i^{t+h}, \tag{3.59}$$

$$\boldsymbol{v}_i^{t+h} = \boldsymbol{v}_i^t + \boldsymbol{F}_i^{t+h} \frac{h}{m_i}, \tag{3.60}$$

To compute the force at surface point i at time $t+h$, $\boldsymbol{F}_i^{t+h}$ is derived from the force at surface point i at time t, $\boldsymbol{F}_i^t$. Then the surface point i at time $t+h$, p_i^{t+h} is as follows:

$$p_i^{t+h} = p_i^t + \boldsymbol{v}_i^{t+h} h, \tag{3.61}$$

where p_i^{t+h} and $\boldsymbol{v}_i^{t+h}$ is the position and velocity of surface point i at time $t+h$, p_i^t is the position of surface point i at time t, and h is the time interval between simulation steps.

Simplified Implicit Integration

Our proposed integration method is inspired from [1,53] to model 3D soft objects as a 2D grid e.g. cloth. Ref. [1] offers a fast and stable calculation method as compared to [53]. We extend this method to model 3D objects as 3D structures and include an approximation for implicit integration to animate the soft bodies. Rather than computing velocity at each neighboring point, we use an approximation of the forces exerted by neighboring points to generate the velocity at the considered point. Our proposed simplified approximated implicit method bears less computation as compared to the method in [1]. The derived method is described below.

As mentioned earlier, implicit Euler method equations are given by:

$$\boldsymbol{v}_i^{t+h} = \boldsymbol{v}_i^t + \boldsymbol{F}_i^{t+h} \frac{h}{m_i} \tag{3.62}$$

$$p_i^{t+h} = p_i^t + \boldsymbol{v}_i^{t+h} h \tag{3.63}$$

$\boldsymbol{F}_i^{t+h}$ is not easily evaluated at the current value of p_i^t, $\boldsymbol{v}_i^t$, and $\boldsymbol{F}_i^t$, but can be approximated by a first-order derivative as [1,53]:

$$\boldsymbol{F}^{t+h} = \boldsymbol{F}^t + \frac{\partial \boldsymbol{F}}{\partial x} \Delta p^{t+h}, \tag{3.64}$$

where $\boldsymbol{F}^t$ is the net force at point i, and is given by $\boldsymbol{F}^t = [\boldsymbol{F}_1^t, \boldsymbol{F}_2^t, ..., \boldsymbol{F}_r^t]^T$, where r is number of surface points defining the soft body. The difference of positions, Δx^{t+h}, at times t and $t+h$, is given by:

$$\Delta p^{t+h} = p^{t+h} - p^t \tag{3.65}$$

It is also written as:

$$\Delta p^{t+h} = (\boldsymbol{v}^t + \Delta \boldsymbol{v}^{t+h}) h, \tag{3.66}$$

where $\Delta \boldsymbol{v}^{t+h}$ is the difference of the velocities at times t and $t+h$.

To calculate $\boldsymbol{F}^{t+h}$, Desbran et al. [53] use the negated Hessian matrix $H = \frac{\partial \boldsymbol{F}}{\partial p}$ of the mass-spring system to solve implicit method integration in (3.64). Substituting (3.66) in (3.64), the equation becomes:

$$\boldsymbol{F}^{t+h} = \boldsymbol{F}^t + H(\boldsymbol{v}^t + \Delta \boldsymbol{v}^{t+h}) h \tag{3.67}$$

Then, substituting (3.67) in (3.62) and simplifying, we obtain [53]

$$(I - \frac{h^2}{m} H)\Delta \boldsymbol{v}^{t+h} = (\boldsymbol{F}^t + H\boldsymbol{v}^t h)\frac{h}{m} \tag{3.68}$$

Now, the approximated Hessian matrix is given as [53]:

$H_{ij} = k_{ij}$, if $i \neq j$; and

$H_{ii} = -\sum_{j \neq i} k_{ij}$, otherwise,

where k is the spring constant.

The calculation in (3.68) becomes expensive as there is an $n \times n$ matrix H involved, where n denotes the number of surface points on the soft body. Since H_{ij} is 0 if points i and j are not linked, the velocity change of point i can be updated by considering only the linked points. This approximation leads to [1]:

$$(1 - \frac{h^2 H_{ii}}{m_i})\Delta \boldsymbol{v}_i - \frac{h^2}{m^i} \sum_{\forall j | (i,j) \in E} (H_{ij} \Delta \boldsymbol{v}_j) = \frac{\boldsymbol{F}_i^t h}{m_i}. \tag{3.69}$$

If k is constant for all springs, then H_{ii} and H_{ij} can be approximated as $-kn_i$ and k, respectively, to get [1]:

$$\frac{(m_i + h^2 k n_i)}{m_i} \Delta \boldsymbol{v}_i^{t+h} = \frac{\boldsymbol{F}_i^t h}{m_i} + \frac{h^2 k \sum_{(i,j) \in E} \Delta \boldsymbol{v}_j^{t+h}}{m_i}. \tag{3.70}$$

Simplifying for $\Delta \boldsymbol{v}_i^{t+h}$ and substituting the value $\Delta \boldsymbol{v}_j^{t+h} = |\Delta \boldsymbol{v}_j^t| \frac{\boldsymbol{F}_j^t}{|\boldsymbol{F}_j^t|}$ in (3.69), we get:

$$\Delta \boldsymbol{v}_i^{t+h} = \frac{(\boldsymbol{F}_i^t h + kh^2 \sum_{(i,j) \in E} |\Delta \boldsymbol{v}_j^t| \hat{\boldsymbol{F}}_j^t)}{m_i + kh^2 n_i}, \tag{3.71}$$

where $\hat{\boldsymbol{F}}_j^t$ is the normalized force at point j, evaluated by $\boldsymbol{F}_j^t / |\boldsymbol{F}_j^t|$.

We simplify this equation by considering the velocity difference at times t and $t+h$. In our approximation, the value of

$$\sum_{(i,j)\in E} |\Delta \boldsymbol{v}_j^t| \hat{\boldsymbol{F}}_j^t \tag{3.72}$$

is approximated by:

$$n_i |\Delta \boldsymbol{v}_i^t| \hat{\boldsymbol{F}}_i^t, \tag{3.73}$$

where n_i is the number of neighbors of point i and $\hat{\boldsymbol{F}}_i^t$ is the normalized force at point i, evaluated by $\boldsymbol{F}_i^t / |\boldsymbol{F}_i^t|$. Equation 3.72 computes the summation from the number of neighboring surface points that are connected to point i, all of which are assumed to be of identical mass and form part of a symmetrical mesh that defines the surface of the soft body. Since this is a mass-spring system, force computed at point j is equal and opposite to the force exerted at the connected point i. Thus, we have $\boldsymbol{F}_j^t = -\boldsymbol{F}_i^t$. Now, force is used to compute velocity, as shown in (3.62). Since we assume the same parameters, i.e. mass, spring and damping constants, for each surface point and each spring, we may assume $|\Delta \boldsymbol{v}_j^t| = -|\Delta \boldsymbol{v}_i^t|$ without loss of generality. Then, the summation of (3.72) is approximated by using the number of neighboring points n_i multiplied by the magnitude of velocity difference and normalized force at point i. Using the approximation of (3.73) in (3.71), we get:

$$\Delta \boldsymbol{v}_i^{t+h} = \left(\frac{\boldsymbol{F}_i^t h + kh^2 \left(n_i |\Delta \boldsymbol{v}_i^t| \hat{\boldsymbol{F}}_i^t \right)}{m_i + kh^2 n_i} \right).$$

We can simplify the equation as follows:

$$
\begin{aligned}
\Delta \boldsymbol{v}_i^{t+h} &= \frac{\boldsymbol{F}_i^t h}{m_i + kh^2 n_i} + \frac{kh^2 n_i \mid \Delta \boldsymbol{v}_i^t \mid \hat{\boldsymbol{F}}_i^t}{m_i + kh^2 n_i} \\
&= \frac{\boldsymbol{F}_i^t h}{m_i + kh^2 n_i} + \frac{kh^2 n_i \mid \Delta \boldsymbol{v}_i^t \mid \boldsymbol{F}_i^t}{m_i \mid \boldsymbol{F}_i^t \mid + \mid \boldsymbol{F}_i^t \mid kh^2 n_i} \\
&= \frac{\boldsymbol{F}_i^t h}{m_i + kh^2 n_i} + \frac{kh^2 n_i \mid \Delta \boldsymbol{v}_i^t \mid \boldsymbol{F}_i^t}{\mid \boldsymbol{F}_i^t \mid \left(m_i + kh^2 n_i\right)} \\
&= \frac{\mid \boldsymbol{F}_i^t \mid \boldsymbol{F}_i^t h + kh^2 n_i \mid \Delta v_i^t \mid \boldsymbol{F}_i^t}{\mid \boldsymbol{F}_i^t \mid \left(m_i + kh^2 n_i\right)} \\
&= \frac{\boldsymbol{F}_i^t h\left(\mid \boldsymbol{F}_i^t \mid + khn_i \mid \Delta \boldsymbol{v}_i^t \mid\right)}{\mid \boldsymbol{F}_i^t \mid \left(m_i + kh^2 n_i\right)} \quad .
\end{aligned}
\tag{3.74}
$$

The simplified implicit integration takes into account, spring, damping, fluid modeling, internal pressure forces, and gravitational acting at each point on the surface of the body. All external and internal forces are calculated and aggregated and then the velocity of point i at time $t+h$ is computed by using either of the explicit, implicit, or proposed implicit methods. The result is passed onto the next step and then a new position is calculated for point i. Finally, a display function renders the soft body. This loop repeats until the user terminates the simulation.

We have presented the proposed generalized soft body method that employs some specific cases of other existing models to simulate a wide variety of materials by adjusting parameters such as body control, surface deformation, volume control, constraints, and gravitation. The combined forces within the soft body model combine all these parameters to create a realistic simulation of the intended material. In the next chapter we describe how the soft body can be efficiently implemented to provide the simulation that displays each animated frame in interactive speed.

CHAPTER 4: IMPLEMENTATION OF THE SPECFIC METHODS OF SOFT BODIES

This section details each phase of our soft body implementation. In the specific methods, the edge of any two connecting surface points is a spring between those points, where the length of the edge is initially considered as the rest state of the spring. Each spring is represented by a spring force and a damping force at each end. To maintain volume within the soft body, internal pressure is exerted onto the soft body triangular mesh surface area. The internal pressure force is applied to each triangle area in the mesh, spring forces exert on connected points, and gravitational force is applied to each surface point.

To incorporate fluidity, surface points are free moving particles that exert fluid forces on all neighboring points within a radius. Because determining SPH fluid force between all free moving neighboring surface points is an $O(n^2)$ process, an efficient method is required to quickly determine neighboring points. In this chapter, we present several data structures, such as AABB, Octree, and a partitioning and hashing scheme, and examine how each structure increases performance. We choose the most efficient method for the fluid modeling based on those results. The same selected data structure is also applied to collision detection so the soft body can interact with other soft body and rigid body objects in the environment.

This chapter is organized as follows. Section 4.1 describes how to determine the neighboring surface points. Section 4.2 shows the comparison of the methods for fluid modeling. Section 4.3 presents the dynamically resizing grid cell scheme for collision detection. Section 4.4

discusses the soft body simulation in details. Finally section 4.5 provides the overview of performance efficiency and complexity analysis.

4.1 Determining the neighboring surface points

Since eq. 3.52, 3.54, and 3.55 in fluid modeling concern only the surface points in the radius, the surface points outside the radius can be ignored. If we determine the neighboring surface points at the beginning of each animation frame, then the speed of the simulation can be improved. To determine the neighboring surface points of a point effectively, we present three different methods, AABB, Octree, and a partitioning and hashing scheme. Then, we compare the performance of each method and discuss what method is suitable for our fluid modeling in the next section.

4.1.1 Determining neighboring surface points by AABB

AABB has been used in many computer graphics applications, such as collision detection or ray tracing [75,101]. The reason why AABB is widely used in many applications is because AABB can be simply computed and it is fast for the simple models.

To find the neighboring surface points for the surface point p for fluid modeling using the AABB, each soft body model is bounded by a AABB. This AABB is simply computed by the soft model's minimum and maximum points, (min_x, max_x), (min_y, max_y), and (min_z, max_z). The distance between surface point p to each surface point in the soft body's AABB needs to be

computed for neighboring surface points in the core radius h_w. The AABB that bounds a soft body is shown in figure 4—1.

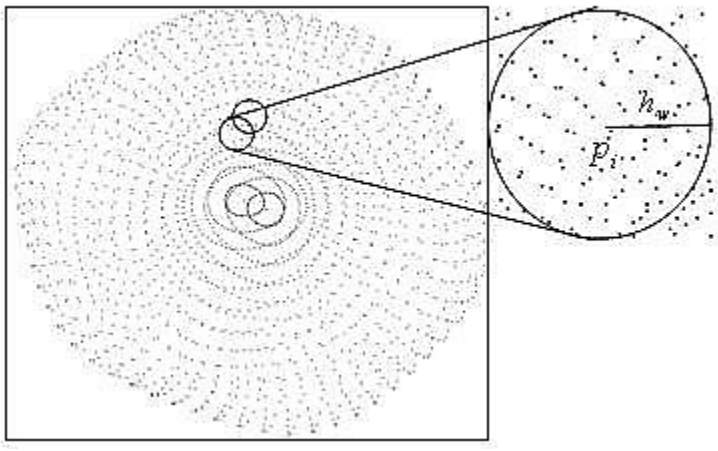

Figure 4—1: Axis Aligned Bounding Box (AABB) for determining the neighboring surface points of surface point p. The AABB is used to bound a soft body model and then all surface points in the soft body need to be tested if they are in the core radius h_w of surface point p.

For the surface point p, fluid force in SPH method is affected by the surface points within the core radius h_w. We crate the *fluid particle list* to contain neighboring surface points of surface point p. Each element p of the fluid particle list has a pointer to the surface points within the core radius h_w of the surface point p as shown in figure 4—2.

We find the neighboring surface points each surface point by computing the distance between that surface point and each surface point in the soft body. If the surface points are in the core radius h_w of surface point p then we place those surface points to the table fluid particle list table.

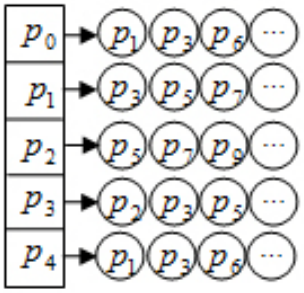

Figure 4—2: Fluid particle list table. Each soft body object has a fluid particle table of size *n*, where *n* is the number of surface points in the soft body. Each entry in the table lists contains the surface points within the kernel radius of the surface point correlation.

The fluid particle table list contains the list of neighboring surface points of each surface point. We use these neighboring surface points to approximate the fluid density, fluid pressure force, and fluid viscosity force at surface point p in fluid modeling. The next section we present Octree structure that can also be implemented to determine the neighboring surface points that later we use for fluid modeling.

4.1.2 Determining neighboring surface points by Octree

The Octree is the tree data structure where each inner node contains up to eight children [86,102]. The 3D space is recursively subdivided the space into eight octants. In point based subdivision, the Octree subdivides the 3D space from the locations of points. Each node in an Octree contains the points that reside in one of subdivisions. Since in our implementation each soft body has its own Octree, an AABB is used to bound each soft body model and to define as 3D space for Octree subdivisions. Then, determining the neighboring surface points by Octree is accomplished in two steps: 1) creating the Octree and 2) finding the surface point in the core radius h_w of surface point p. We present the details of each step here.

4.1.2.1 Creating the Octree

Each surface point of the soft body model is passed into Octree to add the new node into the tree. This function uses location of the surface point and recursively subdivides the 3D space of the model into eight octants, shown in figure 4—3, and the surface point resides in one of the octants. In our implementation, we create a function to add new node in the tree which returns the tree level of the surface point. If tree level at surface point p is defined as T_{p,t_l}, the function to add a new node in the Octree is defined as:

$f_{addnode}(p) = T_{p,t_l}$,

where t_l is the level of the tree. Note that the root of the Octree is level 0, the next level is 1, and so on. This step is performed for all surface points p, and then the Octree is filled with all surface points. The Octree structure is shown in figure 4—4.

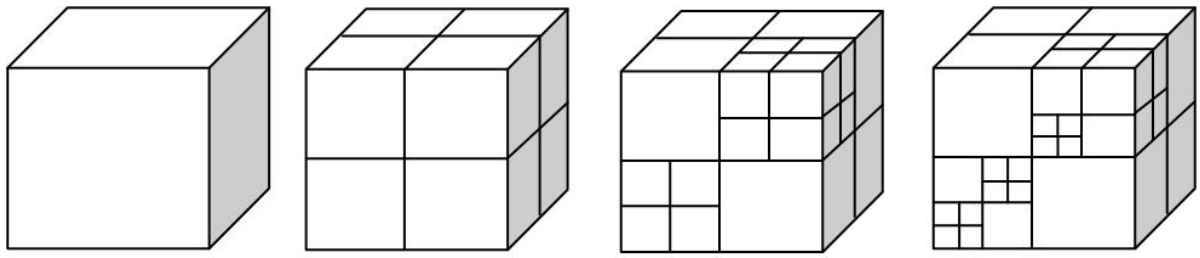

Figure 4—3: Octree subdivision. The three dimension space is recursively subdivided the space at the location of the surface points into eight octants.

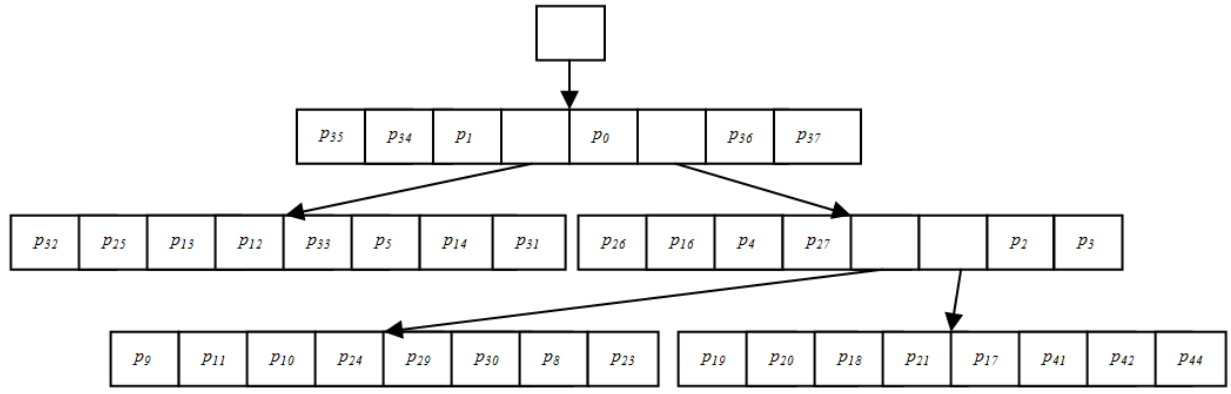

Figure 4—4: Octree structure. The add new node function uses location of the surface point and recursively subdivides the 3D space of the model into eight octants. The child pointer of each tree node references the subdivided spaces where surface point can reside in one of those spaces.

4.1.2.1 Finding the surface points in the core radius h_w of surface point p

To find the neighboring surface points of surface point p, the bounding box B_p as a size of the core radius h_w is created around surface point p .The B_p is defined as the minimum and maximum points, $(minB_{px}, maxB_{px})$, $(minB_{py}, maxB_{py})$, and $(minB_{pz}, maxB_{pz})$.

Each node of Octree contains its bounding box that has been created during subdivisions. Let ‘s assume that the bounding box of each tree node is defined by B_{t_l}. The B_{t_l} is also defined as the minimum and maximum points, $(minB_{tx,t_l}, maxB_{tx,t_l})$, $(minB_{ty,t_l}, maxB_{ty,t_l})$, and $(minB_{tz,t_l}, maxB_{tz,t_l})$, of tree subdivisions at level t_l. To determine the neighboring surface points, the method uses these two bounding boxes to perform the intersection test between B_{t_l} and B_p and then the surface points in B_{t_l} is checked whether or not it is in the surface point core radius.

In the Octree, we use bounding box interactions to find the path in the tree to find the neighboring surface points. The B_{t_l} is tested for intersection with B_p. If B_{t_l} intersects with B_p, the surface point in the tree node is tested whether or not it is in the core radius of surface point p. If so, the surface point is placed in the fluid particle list table at index p, shown in figure 4—2. Then the process repeats for the higher levels of the non-leaf nodes that B_{t_l} intersects B_p. After all surface points p are tested for the list of surface points in the core radius h_w, the fluid particle list table is filled and then we can compute the fluid density, fluid pressure, fluid viscosity for each surface point.

4.1.3 Neighboring surface points by hashing for fluid model

To reduce the time complexity, we have applied a custom spatial hashing scheme in [92] to partition 3D space into grid cells to determine the neighboring surface points in SPH. During animation, the grid cell of each surface point in the frame is determined by spatial partitioning. The surface point within that grid cell is then mapped into a 3D hash table. Determining all neighboring points of a surface point is accomplished in two steps: (1) creating a 3D hash table for all surface points, and (2) obtaining a list of neighboring points for each surface point using the 3D hash table.

4.1.3.1 3D hash table creation

To determine the neighboring surface points of a particular surface point, the simulation environment is partitioned into cubic grid cells. Each grid cell has the same size as the kernel of fluid model represented by SPH core diameter, $2h_w$, where h_w is core radius of SPH. In this partitioning, a three dimensional (3D) array for these grid cells requires a large amount of memory space. For example, a $100x100x100$ array structure needs memory space for 10^6 grid cells. However, our models consist of only the surface points in the animated scene. A majority of the grid cells do not contain any information. To avoid large memory allocation for the large number of grid cells to search for the neighboring surface points, we propose a set of lists indexed by a 3D hashing scheme to utilize memory space effectively and develop an efficient search technique for neighboring surface points. In this hashing scheme, we map the surface points into a small 3D hash table. Since the hash table size is an important parameter for the

execution time of each animated frame, experiment on hash table size is included in the future work.

Conceptually, our spatial partitioning method divides the environment into grid cells. However in reality we do not create all these grid cells. Instead, for each surface point we determine the grid cell index by using the position of the surface point. This grid cell index of the surface point is mapped into the hash table index. The surface point is then placed into the list pointed by the hash index shown in figure 4—5. The grid cell index and hash table index of the surface point p are as follows.

The surface point $p = (x, y, z)$ belongs to the grid cell $c = (c_x, c_y, c_z)$, where $c_x = \lfloor x / 2h_w \rfloor$, $c_y = \lfloor y / 2h_w \rfloor$, and $c_z = \lfloor z / 2h_w \rfloor$, and h_w is the core radius of SPH. (4.1)

The hash index $H = (H_x, H_y, H_z)$ of the grid cell $c = (c_x, c_y, c_z)$ is determined by

$H_x = c_x \ \mathbf{mod}\ HT$, $H_y = c_y \ \mathbf{mod}\ HT$, and $H_z = c_z \ \mathbf{mod}\ HT$, (4.2)

where HT is the hash table size in each direction. Hence, HT^3 is the total size of the hash table.

Let's us assume that the minimum and maximum grid cell index in x, y, and z directions are given by $(minc_x, maxc_x)$, $(minc_y, maxc_y)$, and $(minc_z, maxc_z)$.

The neighboring grid cells $c' = (c'_x, c'_y, c'_z)$ of the grid cell $c = (c_x, c_y, c_z)$ is given by

$c'_x = c_x + I_c$, $c'_y = c_y + I_c$, and $c'_z = c_z + I_c$, (4.3)

where $I_c = 0, \pm 1$ and $minc_x \le c'_x \le maxc_x$, $minc_y \le c'_y \le maxc_y$, $minc_z \le c'_z \le maxc_z$.

Then the hash index $H' = (H'_x, H'_y, H'_z)$ of a neighboring cell $c' = (c'_x, c'_y, c'_z)$ is

$H'_x = c'_x \ \mathbf{mod}\ HT$, $H'_y = c'_y \ \mathbf{mod}\ HT$, and $H'_z = c'_z \ \mathbf{mod}\ HT$. (4.4)

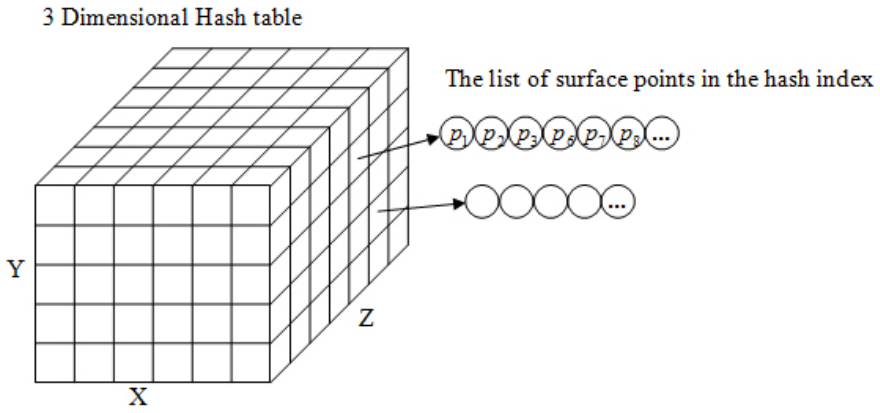

Figure 4—5: 3D hash table for SPH. Each grid cell index $c = (c_x, c_y, c_z)$ is mapped into the hash index $H=(H_x, H_y, H_z)$. Then, each hash index points to a list of surface points mapped to that index.

4.1.3.2 Obtaining list of neighboring points from the 3D hash table

Each surface point p has the grid cell index $c = (c_x, c_y, c_z)$ and the hash index $H = (H_x, H_y, H_z)$ indicated in eq. 4.1 and 4.2. All surface points in the hash index $H = (H_x, H_y, H_z)$ are tested to determine if they are within the core radius, h_w, of the surface point p shown in figure 4—6. If so, the surface points within the radius of the surface point p are placed into the fluid particle list table at the index for surface point p as shown in figure 4—2. Since the surface points within the core radius, h_w, of the surface point p may reside in the neighboring grid cells $c' = (c'_x, c'_y, c'_z)$ with the hash indices $H' = (H'_x, H'_y, H'_z)$ indicated in eq. 4.3 and 4.4, the list of surface points in the hash indices $H' = (H'_x, H'_y, H'_z)$ are also checked for the surface points within the core radius, h_w, of surface point p.

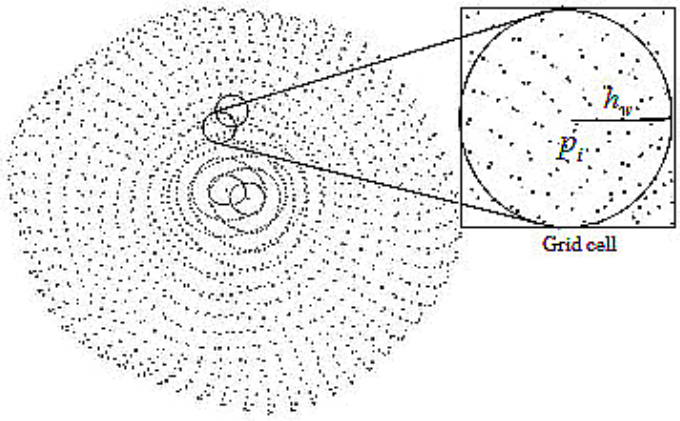

Figure 4—6: SPH with grid cells of the soft body surface. This figure shows surface points within the radius h_w of surface point p_i , where h_w is core radius of SPH.

For each surface point, we determine a list of neighboring surface points. These neighboring surface points affect the fluid density, fluid pressure force, and fluid viscosity force at surface point p in fluid modeling. After fluid modeling, the fluid force is combined with other forces as detailed in chapter 3. Motion physics are then applied to move particles through space. After presenting three different methods for fluid modeling, in the next section we compare the performance among those methods and select the suitable method for the fluid modeling.

4.2 Comparing the methods for fluid modeling

We compare the presented method, AABB , Octree, and the partitioning and hashing schemes, in order to select the suitable method to determine neighboring surface points in our fluid modeling. The experiments are conducted on a mid-range laptop, 2.0 GHz Core-2 Duo processor, 2GB memory, and NVIDIA GeForce 6800 Mobile GPU. The performance of each method is measured by frame per second (FPS). We set up the experiment that generates 500 to 6,000 points. Since we would like to only find the neighboring surface points of a surface point, all points are randomly moving around the scene without computation of any forces. Then we implement AABB, Octree, and the partitioning and hashing scheme to determine the neighboring surface points. The result is shown in figure 4—7.

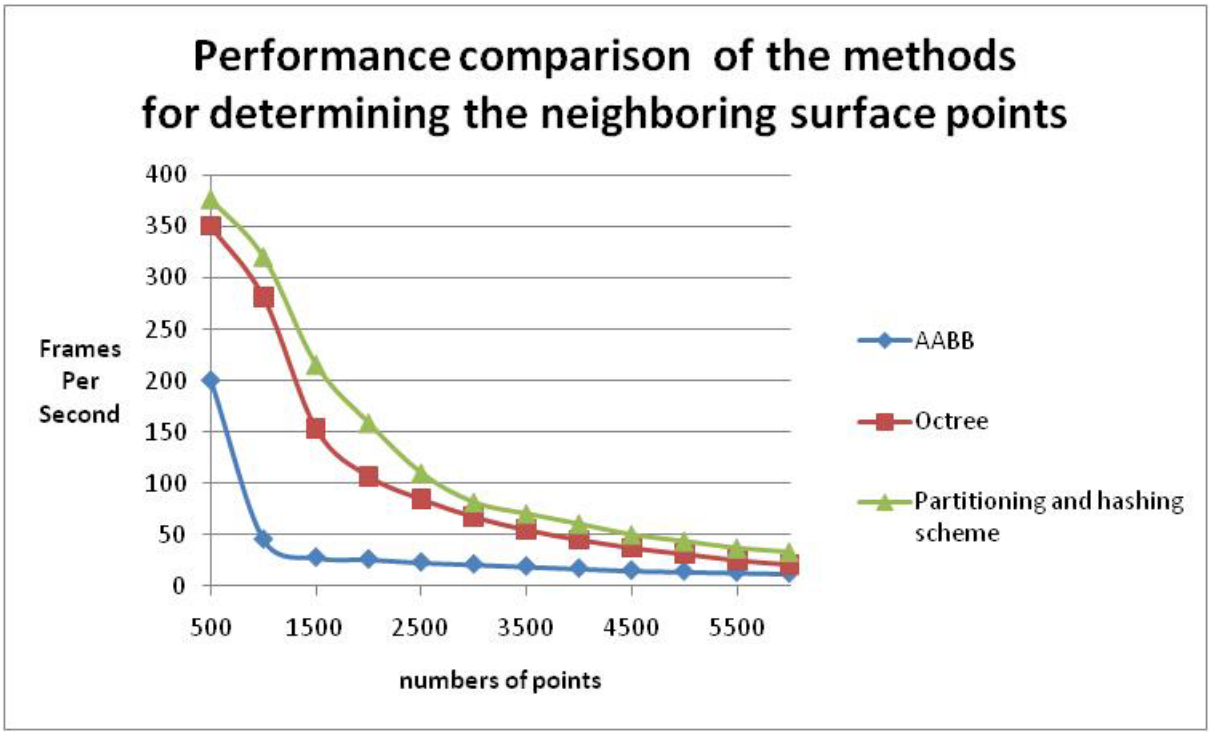

Figure 4—7: Performance comparison of the methods for determining the neighboring surface points. The best frame rate is obtained by using the partitioning and hashing scheme.

When the numbers of points are 500 and 1,000, the AABB scheme provides acceptable frame rates, varied from 200 to 40 FPS. However, when we increase the number of points from 1500 to 6000 then frame rates decrease from 40 down to 5 FPS, which is unacceptable for the simulation. The Octree scheme produces better frame rate compared to the AABB scheme. Even though the number of fluid points are increased to 6000, it still provides an acceptable frame rate at 25 FPS. However, when the partitioning and hashing scheme is applied, it provides even better than Octree. For the number of particles of 6000, the simulation runs at 40 FPS. Thus, in our simulation we use the partitioning and hashing scheme to determine the neighboring surface

points in fluid modeling. To increase in realism of the simulation, the fast collision detection is required by most applications and the partitioning and hashing scheme can also determine the surface points nearby a particular point. In the next section, the collision detection that we plan to use in our simulation is explained.

4.3 Dynamically resizing grid cell scheme for collision detection

In fluid modeling, the grid cell size is based on the size of the kernel, where more points in each grid cell is preferred for higher density fluid modeling. However, fewer points are suited for collision detection to reduce time computation because collision detection considers only the surface points for contact surface. Since our partitioning and hashing schemes are efficient to find the neighboring surface points, we use a similar approach to determine the neighboring surface points for contact surface in collision detection. In this collision detection scheme, all surface points in the environment and surface point of the soft models are determined by location in the hash table. Then the contact surface identifies if there is the collision between soft models and the environment. If so, we determine the collision points and collision response.

Because grid cell size is one of the most important parameters in partitioning and hashing, we have conducted experiments to find the most suitable grid cell size for collision detection. To find the colliding points, we use point-based collision detection where the distance between two surface points determines if there is collision or not. If the distance between two surface points is less than collision threshold, then the collision occurs between those two surface points. Only surface points are considered for collision detection; thus, during collision detection the model components such as mass-spring system, fluid modeling, and internal pressure force can be

ignored until after detection. Therefore, effects from these components are omitted in this experiment.

The experiment is composed of four scenarios of 1,000, 3,000, 5,000, and 7,000 points moving randomly inside an AABB of size $200x200x200$. In each scenario, 26 simulations are run where the number of grid cells is varied from 64,000 down to 343 cells, as shown in table 4—1. The testing measurement in the experiments is frame per second (FPS). Each simulation is run over 10,000 animated frames and is repeated with the same parameters 10 times. Finally, the averages of FPS are recorded and the results are in the graphs shown in figures 4—8, 4—9, 4—10, and 4—11.

Table 4—1: Results of collision detection experiment with grid-based partitioning.

Simulation	Grid cell size	Number of cells $(200/\text{Grid cell size})^3$	1000 points		3000 points		5000 points		7000 points	
			Density (Avg num of points per grid cell)	FPS	Density (Avg num of points per grid cell)	FPS	Density (Avg num of points per grid cell)	FPS	Density (Avg num of points per grid cell)	FPS
1	5	64000	0.016	105	0.047	84	0.078	67	0.109	56
2	6	39304	0.025	161	0.076	111	0.127	86	0.178	64
3	7	24389	0.041	221	0.123	136	0.205	98	0.287	73
4	8	15625	0.064	268	0.192	154	0.320	108	0.448	76
5	9	12167	0.082	311	0.247	156	0.411	109	0.575	78
6	10	8000	0.125	355	0.375	179	0.625	115	0.875	85
7	11	6859	0.146	392	0.437	180	0.729	116	1.021	83
8	12	4913	0.204	428	0.611	184	1.018	114	1.425	85
9	13	4096	0.244	436	0.732	181	1.221	111	1.709	82
10	14	3375	0.296	458	0.889	178	1.481	109	2.074	80
11	15	2744	0.364	466	1.093	177	1.822	107	2.551	74
12	16	2197	0.455	476	1.365	177	2.276	104	3.186	75
13	17	1728	0.579	482	1.736	175	2.894	101	4.051	70
14	18	1728	0.579	472	1.736	166	2.894	100	4.051	65
15	19	1331	0.751	466	2.254	164	3.757	93	5.259	57
16	20	1000	1.000	471	3.000	158	5.000	89	7.000	55
17	21	1000	1.000	460	3.000	149	5.000	85	7.000	54
18	22	1000	1.000	459	3.000	145	5.000	81	7.000	52
19	23	729	1.372	458	4.115	138	6.859	77	9.602	50
20	24	729	1.372	450	4.115	130	6.859	70	9.602	47
21	25	512	1.953	443	5.859	124	9.766	66	13.672	44
22	26	512	1.953	422	5.859	121	9.766	58	13.672	41
23	27	512	1.953	425	5.859	115	9.766	57	13.672	39
24	28	512	1.953	420	5.859	110	9.766	55	13.672	36
25	29	343	2.915	407	8.746	105	14.577	53	20.408	34
26	30	343	2.915	401	8.746	102	14.577	53	20.408	33

This experiment is conducted to find a suitable grid cell size for a specific number of points. The highlighted cells denote the best frame rate results from the experiment. The best frame rates of 471-476, 175-179, 114-115, 80-85 are obtained for collision detection between 1000, 3000, 5000, 7000 points, respectively.

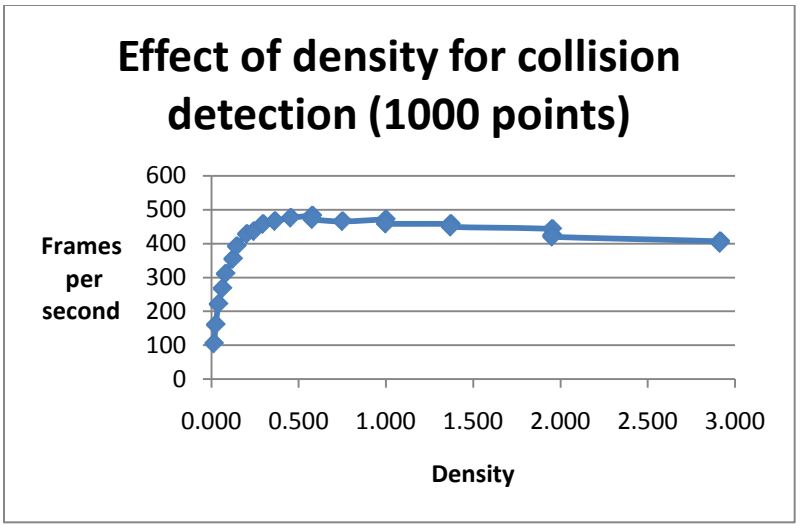

Figure 4—8: Effect of density of points and performance of collision detection. This experiment tests grid-based partitioning for collision detection between 1,000 points. Results show the best frame rate when the density is between 0.45 – 1.0.

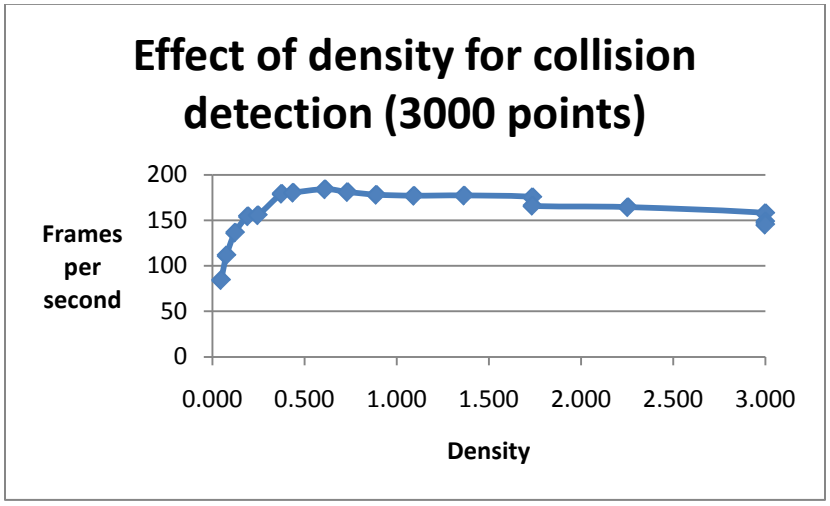

Figure 4—9: Effect of density of points and performance of collision detection. This experiment is set for the collision detection with grid-based partitioning between 3,000 points. The result shows the good frame rate when the density is between 0.4 – 1.7.

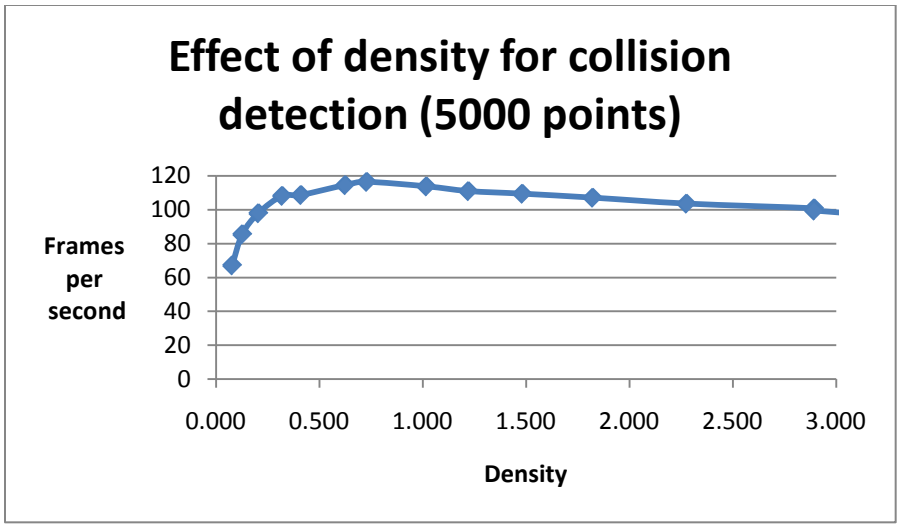

Figure 4—10: Effect of density of points and performance of collision detection. This experiment is set with grid-based partitioning for the collision detection between 5,000 points. The result shows the good frame rate when the density is between 0.6 – 1.2.

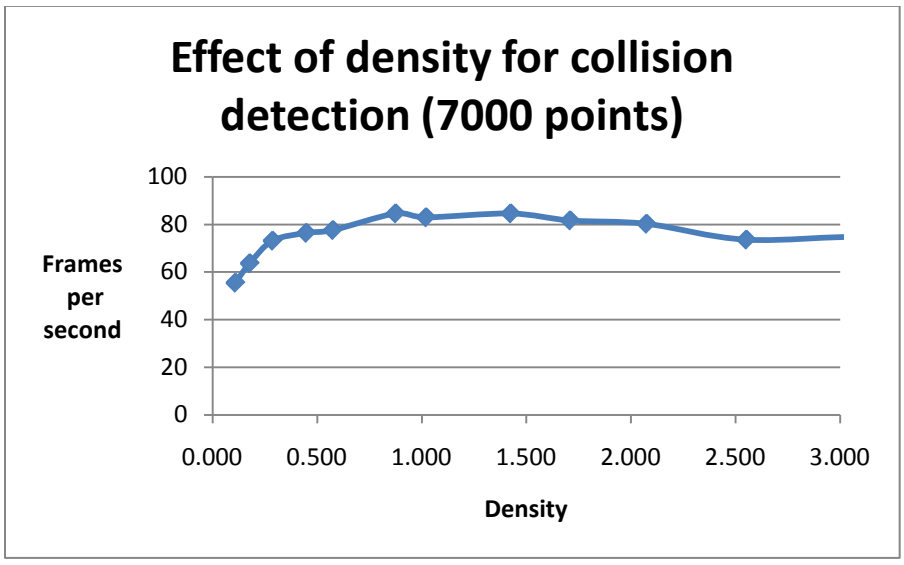

Figure 4—11: Effect of density of points and performance of collision detection. This experiment is set with grid-based partitioning for the collision detection between 7,000 points. The result shows the good frame rate when the density is between 0.8 – 2.1.

Figures 4—9 through 4—11 show that the best frame rates are obtained when the densities are between 0.45 - 1.0, 0.4 - 1.7, 0.6 - 1.2, and 0.78 - 2.1 for collision detection between 1000, 3000, 5000, and 7000 points, respectively. From these results we conclude that the approximated number of 1.0 should be the best density value for collision detection with grid-based partitioning because the good result in each scenario is in the range of 1.0 and our simulation has a number of points not larger than 7000. This approximated density from the experiment is used to calculate the grid cell size for collision detection, which is described below.

From the density in the experiments, we have:

$$D = \frac{Nn}{NC} \tag{4.5}$$

, where D is density, Nn is total number of points, and NC is the total number of grid cells partitioned in x, y, and z directions of AABB.

Since the number of partitions, NP, is equal in x, y, and z directions, NC is calculated as

$$NC = NP^3 \tag{4.6}$$

Substitute eq. 4.6 to eq. 4.5, eq. 4.5 becomes

$$D = \frac{Nn}{NP^3}$$

From the previous experiment, the good result of collision detection is when the density is approximated by one. Then, we have

$$1 = \frac{Nn}{NP^3}. \tag{4.7}$$

We simplify eq. 5.7 and we get $NP^3 = Nn$. In x, y, and z directions ,the number of partitions, NP, becomes $\sqrt[3]{Nn}$.

Thus, the grid cells size in x, y, and z directions (C_x, C_y, C_z) is

$$C_x = \frac{b_x}{NP},\ C_y = \frac{b_y}{NP},\ \text{and}\ C_z = \frac{b_z}{NP}, \tag{4.8}$$

where b_x, b_y, and b_z are the length of AABB in x, y, and z directions.

Then, the grid cell sizes (C_x, C_y, C_z) are used in the collision detection with partitioning and hashing scheme.

Collision detection between soft body models and the environment using partitioning and hashing scheme is performed in three steps. First, an AABB bounds all objects in the animated frame. Second, grid cell size is determined for partitioning and hashing, based on the current size of the AABB. Finally, collisions are detected by point-based contact surface. The details of the collision detection with partitioning and hashing are described next.

First, a single AABB bounds all soft body objects and environment in the frame, defined by the minimum and maximum points of all moving object positions including the points of the environment. Note that the length of the AABB in the x, y, and z dimensions changes over time since all objects may move between animation frames.

Second, to find the grid cell size for partitioning, the grid cell size is specified by the number of grid cells partitioned in x, y, and z directions. According to the previous experiments, the execution time of collision detection depends on the density of the points in the grid cells. After calculating the grid cell size (eq. 4.8), we find grid cell index and the hash table index for surface points as preceded in eq. 4.1 and 4.2 using the grid cell size of C_x, C_y, C_z in x, y, and z directions instead of the core radius of SPH, h_w. Correspondingly for the

neighboring grid cell indices and their hash indices, the eq. 4.3 and 4.4 are applied with the grid cell size of C_x, C_y, C_z instead of the core radius of SPH, h_w.

Finally, all surface points in grid cell of surface point p and the surface points in the neighboring grid cell of surface point p are tested for collision. If the distance between two surface points is less than collision tolerance, then collision occurs. After the collision is detected, the surface points are assigned to the new positions from collision resolution. All soft body models and the environment are rendered to the screen and then all steps of simulation are repeated during the simulation. The next section explains soft body simulation in detail.

4.4 Soft body simulation details

The main program repeats the simulation until the soft body becomes stable or user stops. At the beginning, the partitioning and hashing scheme for fluid modeling creates the 3D hash table (*hashTableCreation ()*) and assigns the 3D hash index to each surface point of the soft body. The function *accessToHashTable()* accesses all surface points p. Each surface point in the 3D hash index and the 3D hash indices of neighboring grid cells of the surface point p is tested for surface points in the SPH core radius, h_w, (*testforFluidParticleList()*). The result of these steps is the fluid particle list table in which each entry has a pointer to the surface points in the SPH core radius.

For all surface points of all soft body models in the scene, each force is calculated and all forces are aggregated. The velocity of each surface point is computed by the combination of all

forces and is approximated by the implicit Euler method. This new velocity provides the new position of each surface point.

At the end of the loop, by using partitioning and hashing scheme for collision detection, new positions of all surface points are placed into the 3D hash table (*hashTableCreation ()*). Then the positions of all surface points in the 3D hash index and the 3D hash indices of neighboring grid cells of the surface point *p* are accessed (*accessToHashTable()*) and tested for collision detection (*testforCollision()*). Collision detection assigns the new positions for the colliding points and then the display function renders the soft body.

The algorithm is organized as follows:

- Data structure used for both fluid modeling and collision detection :
    - *structure surfacePointList* – contains the surface point list used in both the 3D hash table and the fluid particle list table.
- Functions for both fluid modeling and collision detection :
    - *void hashTableCreation (int Gx, int Gy, int Gz)* – constructs the 3D hash table which contains a list of surface points in the 3D hash table index.
    - *void accessToHashTable (int Gx, int Gy, int Gz, string process)* – accesses each surface point in the 3D hash table and passes the surface points to either *testforFluidParticleList* or *testforCollision.*
- Functions for fluid modeling :

- *void fluidParticleListTableCreation()* – creates the fluid particle list table of size *n*.
- *void testforFluidParticleList(int i, int j)* – stores the surface point j in the SPH core radius, h_w, of surface point i into fluid particle list table.

- Functions for collision detection :
  - *void testforCollision(int i, int j)* – detects the collision between surface points *i* and j.
  - *void createOneAABB()* – creates an AABB for collision detection.
  - *int (Cx, Cy, Cz) findGridCellSizeForCollision()* – computes the grid cell size, C_x, C_y, C_z for collision detection.

The details of all functions are presented in figures 4—12 through 4—20:

```
structure surfacePointList
  {
    public int count;
    public int pointNumber[100];
  }
```

Figure 4—12: Structure of *surfacePointList*. This structure is for containing surface point list used in the 3D hash table and the fluid particle list table. The variable *count* tracks the number of surface points in the list. The variable *pointNumber* stores the surf ace points into the list.

```
void hashTableCreation (int Gx, int Gy, int Gz)
{
        if (hash table does not exist)
                create an empty hash table with size of HTxHTxHT and pointer for each index pointing to empty
surfacePointList
        for all indexed (x,y,z) = 0 to HT-1
                initialize count in each hashTable[Hx][Hy][Hz] to 0
        for all surface point i = 0 to n-1 {
                // find grid cell index for each surface point i
                cx = ⌊xi / Gx⌋, cy = ⌊yi / Gy⌋ cz = ⌊zi / Gz⌋, (xi , yi , zi) is position of surface point i
                // convert the grid cell index to hash table index
                Hx = cx mod HT, Hy = cy mod HT, Hz = cz mod HT
                add the surface point i to the surfacePointList indexed by count in hashTable[Hx][Hy][Hz]
                increase count in hashTable[Hx][Hy][Hz] by one
        }
}
```

Figure 4—13: The function *hashTableCreation(int Gx, int Gy, int Gz)*. This function creates the 3D hash table by taking grid cell size *(Gx, Gy, Gz)* as arguments. Both fluid modeling and collision detection use this function to construct the 3D hash table. When being called, it removes all surface points in the list in each entry of the 3D hash table by initializing the variable *count* to 0. Then, the function uses the grid cell size to compute the 3D grid cell index and converts the 3D grid cell index to the 3D hash table index. The result of the function is the 3D hash table containing the list of surface points.

```
void accessToHashTable (int Gx, int Gy, int Gz, string process )
{
    for all surface point i = 0 to n-1
        x = ⌊x_i / Gx⌋,  y = ⌊y_i / Gy⌋ z = ⌊z_i / Gz⌋, (x_i , y_i , z_i) is position of surface point i
         // repeats for all surface points in the hash table indexed by (x, y, z) and all surface points in neighboring grid
         // cells in the hash table indexed by (x±1, y±1, z±1) mod HT.
         for (k =  0 to k = count – 1 in hashTable [x' = (x±{0,1}) mod HT],[ y'=(y±{0,1}) mod HT],
              [z'=(z±{0,1}) mod HT])
            j is the surface point indexed by k in  hashtable[x'][y'][z']
            if (surface point i ≠ surface point j)
             {  if   (process == "fluid modeling")
                      testforFluidParticleList(i, j)
                 if (process == "collision")
                      testforCollision(i, j)
              }
```

Figure 4—14 : The function *void accessToHashTable (int Gx, int Gy, int Gz, string process).* This function accesses the 3D hash table in which the surface point *p* is placed. All surface points in the 3D hash index and all surface points in 3D hash indices of neighboring grid cells are tested for either fluid modeling or collision detection. If this function is used for fluid modeling, it passes surface points *i* and *j* to the function *testforFluidParticleList.* Similarly, if this function is used for collision detection, it passes those two points to the function *testforCollision.*

```
void fluidParticleListTableCreation()
{
        if (fluidParticleListTable does not exist)
                create an empty fluid particle list table with size of n and pointer for each index pointing to empty
surfacePointList
        for i = 0 to n-1
                initialize count in each fluidParticleListTable[i] to 0
}
```

Figure 4—15 : : The function *void fluidParticleListTableCreation().* This function is used for fluid modeling. It creates the fluid particle list table of size *n* if the table has not been created. When this function is called, it removes all surface points in the list for each entry of the fluid particle list by initializing the variable *count* to 0.

```
void testforFluidParticleList(int i, int j)     {
        distance is the distance between surface point i and j
        if (distance is less than or equal to the SPH core radius, h_w )
        {       add the surface point j to the surfacePointList indexed by count in fluidParticleListTable[i]
                increase count in fluidParticleListTable[i] by one
        }
}
```

Figure 4—16 : The function *void testforFluidParticleList(int p, int q).* This function is called by the function *accessToHashTable* and stores the surface points in the fluid particle list table for fluid modeling. It checks if the surface point *j* is in the SPH core radius, h_w, of surface point *i* or not. If so, the surface point *j* is placed into the surface point list indexed by *count* in the fluid particle list table indexed by *i*.

```
void testforCollision(int i, int j)          {
        distance is the distance between surface point i and j
        if (distance is less than collision threshold)
                compute the effect of collision
}
```

Figure 4—17 : The function *void testforCollision(int p, int q).* This function is called by the function *accessToHashTable* and detects collisions. It checks if the distance between surface point *i* and *j* is less than the collision threshold or not. If so, it resolves the effect of collision for those colliding points.

```
maxAABB(x,y,z) = (-1000, -1000, -1000)
minAABB(x,yz) = (1000, 1000, 1000)
void createOneAABB(){
        for (i = 0 to i = numberOfObjects)
                for (j = 0 to object[i].n-1) {
                  if  (position (x, y, z) of surface point j  of object i is bigger than maxAABB(x,y,z))
                        maxAABB(x,y,z) is equal to position (x, y, z) of surface point j of object i
                  if  (position (x, y, z) of surface point j  of object i is less than minAABB(x,y,z))
                        minAABB(x,y,z) is equal to position (x, y, z) of surface point j of object i
        }
}
```

Figure 4—18 : The function *void createOneAABB().* This function is called at the beginning of collision detection to create an AABB. All surface points of all objects in the frame are tested for the maximum and minimum points of the AABB. This AABB is used later to find the grid cell size for collision detection in the function *findGridCellSizeForCollision*.

```
int (Cx, Cy, Cz) findGridCellSizeForCollision(){
        // find the length of AABB in x, y, z directions
          b(x, y, z) is  (|maxAABB.x- minAABB.x|,|maxAABB.y- minAABB.y|, |maxAABB.z- minAABB.z|)
         // initialize the total number of points to 0
          initialize Nn to 0

           for (i = 0 to i = numberOfObjects)
              Nn is Nn plus the number of points of object[i].

           NP is ∛Nn
          gridCellSize(Cx, Cy, Cz) is (b.x/NP, b.y/NP, b.z/NP)
           return Cx, Cy, Cz
}
```

Figure 4—19 : The function *int (Cx, Cy, Cz) findGridCellSizeForCollision().* This function is called after an AABB has been created. It computes the grid cell size for collision detection by finding the length of AABB. Then the length in each direction is divided by the number of partitions, *NP*. The number of partitions, *NP*, is evaluated by $\sqrt[3]{Nn}$. The result of this function is grid cell size which is later passed as arguments to create the 3D hash table for collision detection.

```
void main() {
    Repeat{
    hashTableCreation( 2h_w , 2h_w , 2h_w )
    accessToHashTable( 2h_w , 2h_w , 2h_w ,  "fluid modeling")
    for all soft bodies
        for all surface points i = 0 to i = n-1     {
            compute spring force for surface point i, F^t_si
            compute spring damping force for surface point i,  F^t_di
            for  j = first surface point to j = last surface point in fluid particle list table indexed i  {
                compute fluid density for surface point i,  ρ^t_i , using all surface points  j
                compute fluid pressure force for surface point i,  F^t_fpi , using all surface points  j
                compute fluid viscosity for surface point i,  F^t_fvi  using all surface points  j
            }
            compute internal pressure force for surface point i,  F^t_vi
            compute gravitation force for surface point i,  F^t_gi
            combine all forces for surface point i,
                F^t_i = α(F^t_si + F^t_di) + β(F^t_fpi + F^t_fvi) + γF^t_vi + δF^t_gi
            compute velocity for surface point i,  v^{t+h}_i = v^t_i + F^{t+h}_i h / m_i
            compute position for surface point i,  p^{t+h}_i = p^t_i + v^{t+h}_i h
        }
    }
    createOneAABB()
    int Cx, Cy, Cz = findGridCellSizeForCollision()
    hashTableCreation(Cx, Cy, Cz)
    accessToHashTable(Cx, Cy, Cz, "collision")
    display()
    } until user stops
```

Figure 4—20 : The function *void main()***.** This main program repeats the simulation until the soft body until the user stops. All soft bodies are simulated with mass-spring force, fluid modeling force, internal pressure force, and gravitational force. Then all forces for each surface point are combined. The velocity is generated by the implicit Euler method to evaluate the new position of the surface point. Finally, collision detection is resolved for the colliding surface points.

4.5 Complexity analysis

Simulating realistic soft body models is computationally intensive because both fluid modeling and collision detection require comparison of each surface point to all other adjacent particles, the time complexity of which increases exponentially with the number of particles. This section examines in detail how the partitioning and hashing scheme presented greatly improves such calculations, making realistic soft body simulations possible on common consumer hardware.

Note that the size of the 3D hash table, HT, is used for the partitioning and hashing scheme. From the result presented in [103], HT in range 13-17 provides the good results for 2,500 – 4,000 surface points, which can be approximated by $HT < \sqrt[3]{n}$. Additionally, HT^3 presents the number of the 3D hash indices of the 3D hash table and is defined as the constant C in this analysis.

With the partitioning and hashing scheme, the function *hashTableCreation()* initializes the variable *count* to 0 in each 3D hash index and assigns the 3D hash index to each surface point, resulting in time complexity $(C + n)$.The function *accessToHashTable()* accesses all surface point p and all surface points in the 3D hash index including the 3D hash indices of the neighboring grid cells of surface point p, resulting in time complexity $(27nn_h)$, where n_h is number of surface points in the surface point list and it is greatly less than n and 27 is the total number of hash indices accessed by the function for each surface point p. The computation of mass-spring force, internal pressure force, gravitational force, combination of all forces, velocity,

and position is for each surface point, resulting in time complexity (n). However, in fluid modeling with the partitioning and hashing scheme, the fluid force of each surface point is calculated with only the surface points in the SPH core radius, h_w. Thus the calculation of all forces, velocity, and position becomes (nn_k), where n_k is the number of surface points in the SPH core radius, h_w and n_k is significantly less than n . For collision detection with the partitioning and hashing scheme, the function *createOneAABB()* processes in each surface point, resulting in time complexity (n). The functions *hashTableCreation()*and *accessToHashTable()* are called an additional time for collision detection, resulting in time complexity ($C + n$) and ($27nn_h$), respectively. This results in a total complexity of $2(C + n) + nn_k + 2(27nn_h)$ or $2(C + n) + nn_k + 54nn_h$. We are currently working on the detail of analysis in average case of this time complexity.

In the best case, we assume that the surface points are distributed equally in the 3D hash table ($n_h = n/C$) and all the surface points in the 3D hash index are in the SPH core radius, h_w, of surface point p ($n_k = n/C$). The time complexity of this case becomes $2(C + n) + n(n/C) + 54n(n/C)$ or $2(C + n) + 55n^2/C$. In the worse case, we assume that all surface points are in one 3D hash index and all surface points are in the SPH core radius, h_w, of each surface point p. The time complexity of this case becomes $2(C + n) + n^2 + 54n^2$ or $2(C + n) + 55n^2$.

With all details of the algorithm described, the next chapter demonstrates the proposed method in practice, as applied to the domains of fluid-like soft bodies, human tissue simulation, and blob-like video game monsters.

CHAPTER 5: SIMULATION OF FLUID-LIKE SOFT BODY, ORGANIC FACE, AND SOFT BODY IN GAMES

Two categories of experiments aim to test all aspects of the proposed algorithm: (1) impact of various force parameters and (2) implementation of fluid-like soft body creatures into the Galactic Arms Race video game [104]. The experiments are conducted on a mid-range laptop, 2.0 GHz Core-2 Duo processor, 1GB memory, and NVIDIA GeForce 6800 Mobile GPU. The scenarios measure algorithm efficiency in FPS.

This chapter is organized as follows. Section 5.1 shows deformation experiments on the simulation of fluid-like soft bodies and organic faces and section 5.2 demonstrates a integration of the soft body into game engine.

5.1 Deformation experiments

Two deformation scenarios are simulations of a soft object and a human face. Each simulation has six different combinations of parameters $\alpha, \beta, \gamma,$ and δ which are body control, fluidity control, volume control, and gravitational field, respectively, where $\alpha > 0$, and $\beta, \gamma, \delta \geq 0$. Since the selected mass-spring system for body control enables the model to deform while maintaining a relative configuration among the surface points without loss of generality, the body control is set to one in both scenarios ($\alpha = 1$).

Both the soft body and human face illustrate the main effects of fluid force and internal pressure force. The first scenario emphasizes fluid-like behavior, thus the fluid force is set to $\beta = 0, 1, 2$ while the second scenario focuses on deformation of the human face (fluidity has less

effect), thus the fluid force is set to $\beta = 0, 0.075$. In both scenarios, the internal pressure force is adjusted and based on the shape and the purpose of the model. For the shape of balloon-like soft body in first scenario, the internal pressure force is set to $\gamma = 0, 1, 5$, which provides more volume. Since the internal pressure force can destroy the shape of the model in the second scenario, the internal pressure force is set to smaller values of $\gamma = 0.001, 0.005, 0.010$.

We have experimented with various parameter values on both scenarios. The visual results of the model depend on the model structures such as the number of surface points and connection of those surface points. Because second scenario has many more surface points than the first scenario, small parameter values provide more suitable results. However, the experiments of parameter values are open for additional experimentation.

5.1.1 Fluid-like soft body simulation

In the first experiment, a soft body mass is dropped into a rigid-body wine glass. To demonstrate a variety of soft body behaviors, six versions of the experiment are performed where equation parameters are varied as shown in table 5—1. The soft body has 1,986 surface points, 3,968 faces, and 5,952 springs. Experiment A shows that a zero fluid parameter value results in a smooth surface of the soft body as shown in figure 5—1. Experiments B and C illustrate variable different fluid parameters that cause different levels of undulating surface waves, and the parameter of fluid force becomes the dominant factor on the soft body as shown in figures 5—2 and 5—3. Experiments D and E have high internal pressure value and behave like bubbles or rubber balls as shown in figures 5—4 and 5—5. Both show that internal pressure force becomes the dominant factor on the soft body. Thus, the fluid force parameter does not affect

visual result in figure 5—5. Finally, experiment F has a zero internal pressure, which results in a soft body without volume much like fabric or cloth as shown in figure 5—6. Overall, the experiment demonstrates the effectiveness of the proposed method to simulate a variety of fluid-like surfaces.

Table 5—1: Experiment parameters for fluid-like soft body deformation experiments.

Experiment	α	β	γ	δ	**FPS**
A	1	0	1	1	>20
B	1	1	1	1	>20
C	1	2	1	1	>20
D	1	0	5	1	>20
E	1	2	5	1	>20
F	1	1	0	1	>20

In each variation of the fluid-like soft body experiments, equation parameters are varied to achieve a variety of fluid-like characteristics The parameters affect the following soft body parameters: (α) net body control, (β) net fluidity control, (γ) net volume control, and (δ) gravitational field. Visual results for each experiment are detailed in figures 5—1 through 5—6.

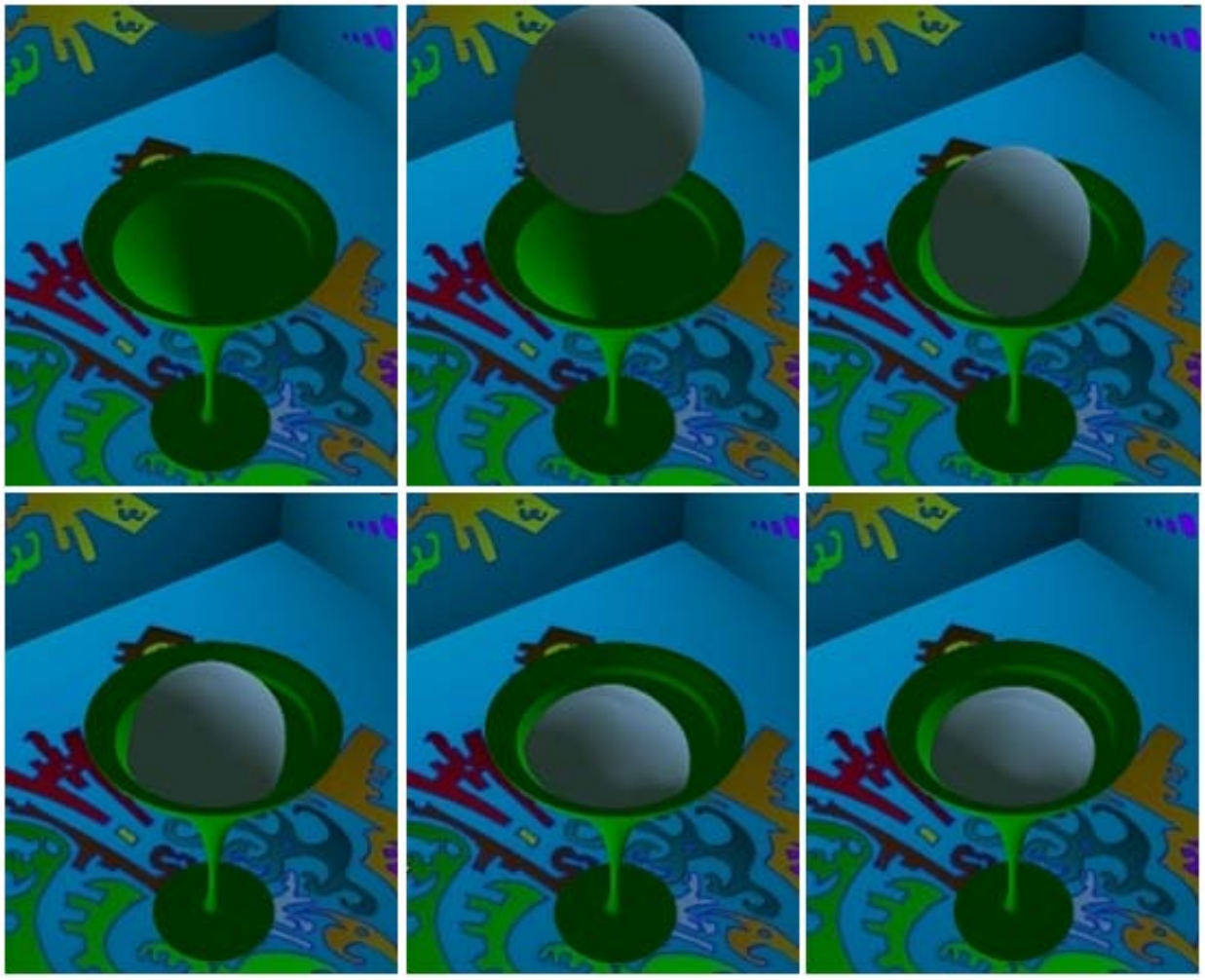

Figure 5—1: The visual result of soft body model of experiment A, where the parameters α =1, β =0, γ =1, and δ =1 have been set. The result is captured at animated frames 100th, 1000th, 2000th, 3000th, 4000th, and 6000th and it shows that these parameters generate the smooth surface of the soft body.

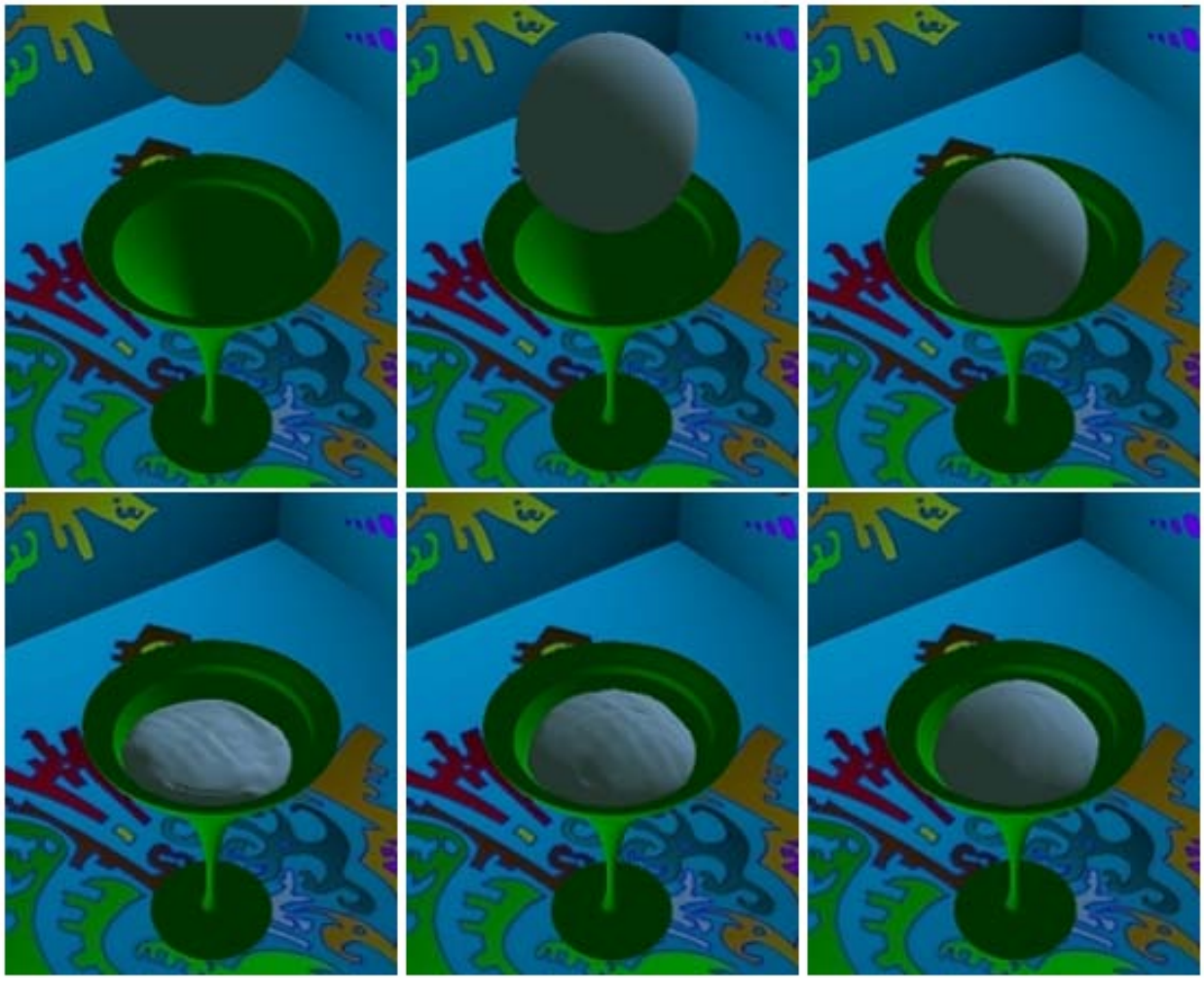

Figure 5—2: The visual result of soft body model of experiment B, where parameters α =1, β =1, γ =1, and δ =1 have been set. The result is captured at animated frames 100th, 1000th, 2000th, 3000th, 4000th, and 6000th and it shows that these parameters generate the undulating surface waves.

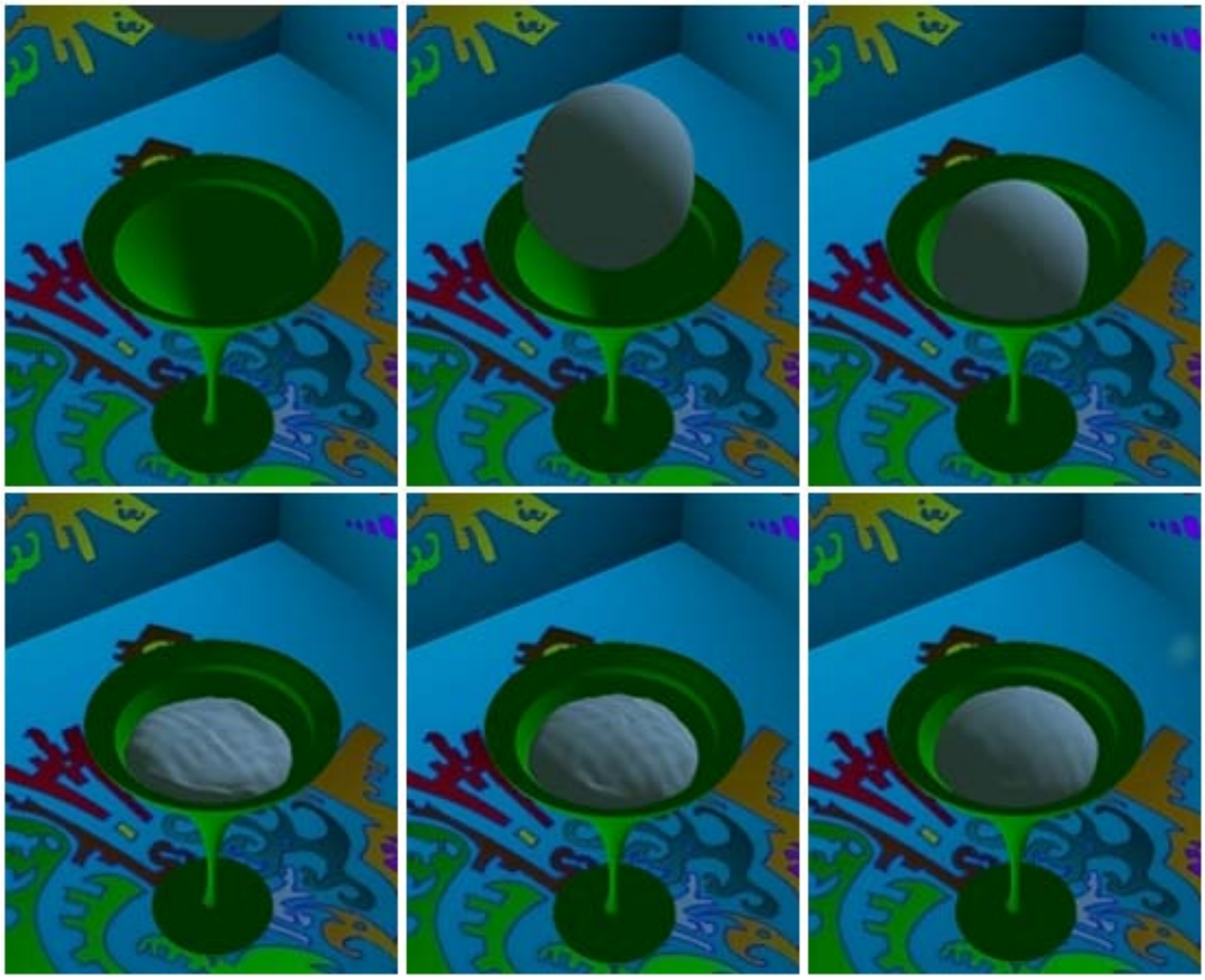

Figure 5—3: The visual result of soft body model of experiment C, where the parameters α =1, β =2, γ =1, and δ =1 have been set. The result is captured at animated frames 100th, 1000th, 2000 ^{h}, 3000th, 4000th, and 6000th and it shows that these parameters generate more undulating surface waves compared to experiment B.

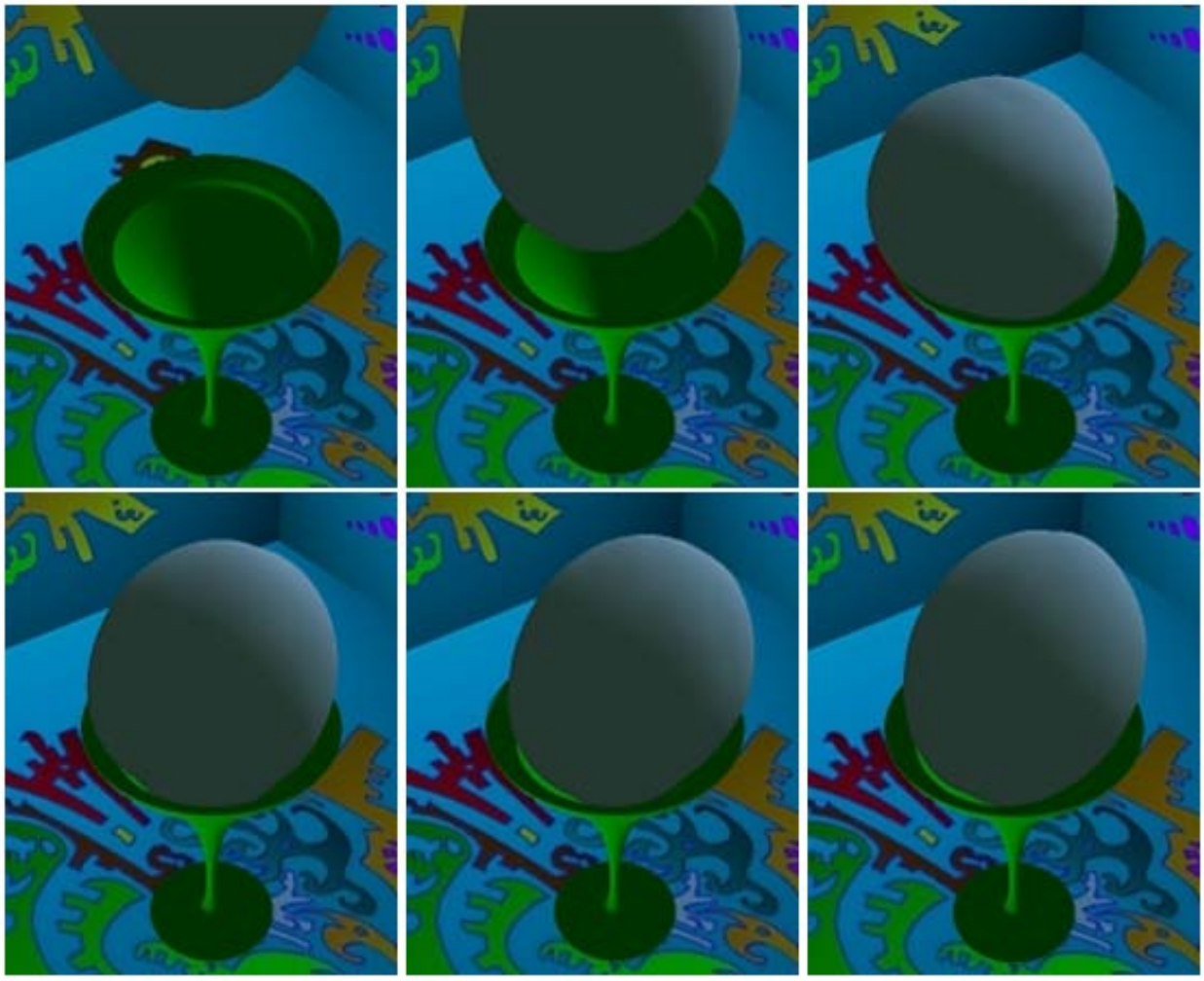

Figure 5—4: The visual result of soft body model of experiment D, where the parameters α =1, β =0, γ =5, and δ =1 have been set. The result is captured at animated frames 100th, 1000th, 2000^{h}, 3000th, 4000th, and 6000th and it shows that these parameters generate bubbles or rubber balls.

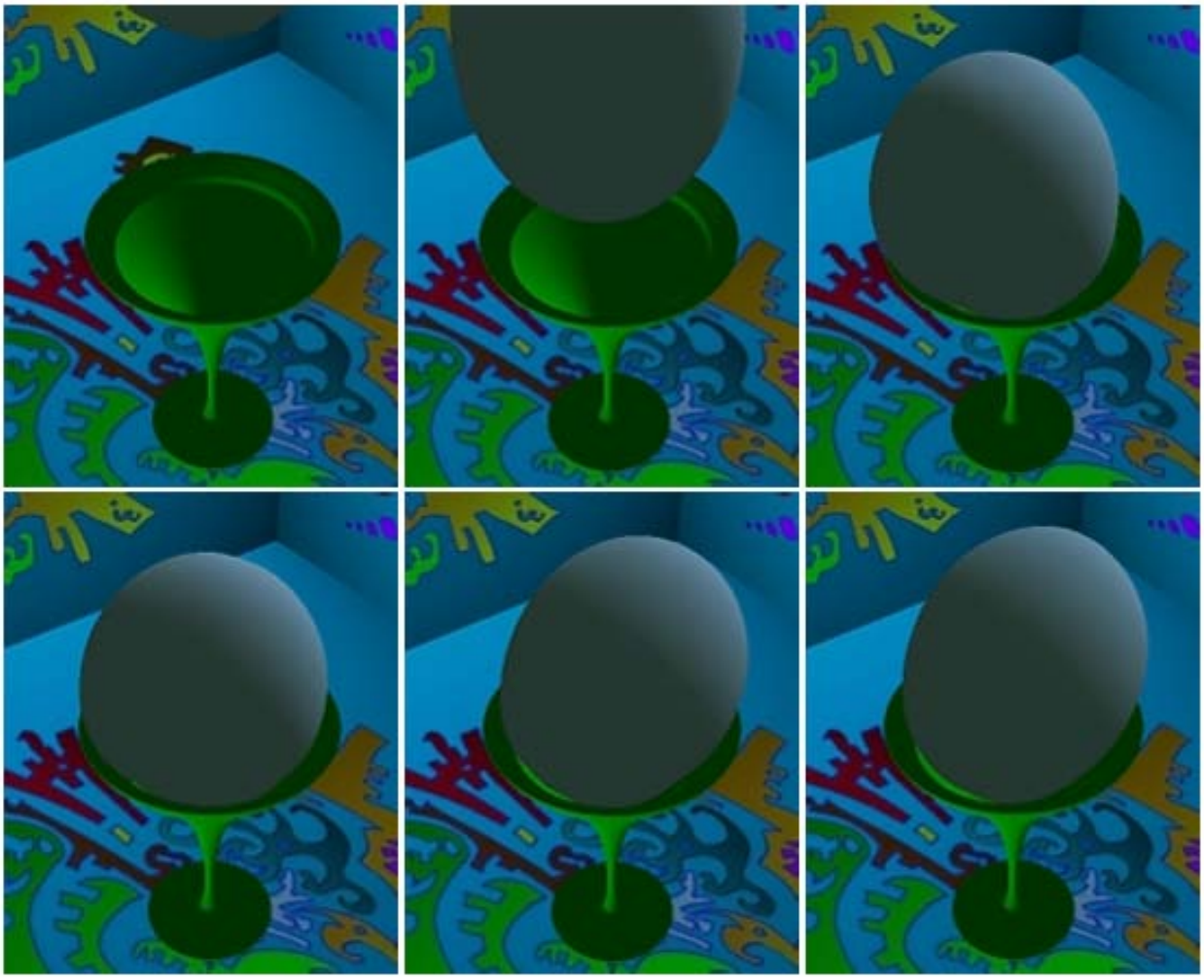

Figure 5—5: The visual result of soft body model of experiment E, where the parameters α =1, β =0, γ =2, and δ =1 have been set. The result is captured at animated frames 100th, 1000th, 2000^{h}, 3000th, 4000th, and 6000th and it shows that these parameters generate bubbles or rubber balls. Since internal pressure force becomes the dominant factor on the soft body, the fluid force parameter does not affect visual result in this experiment.

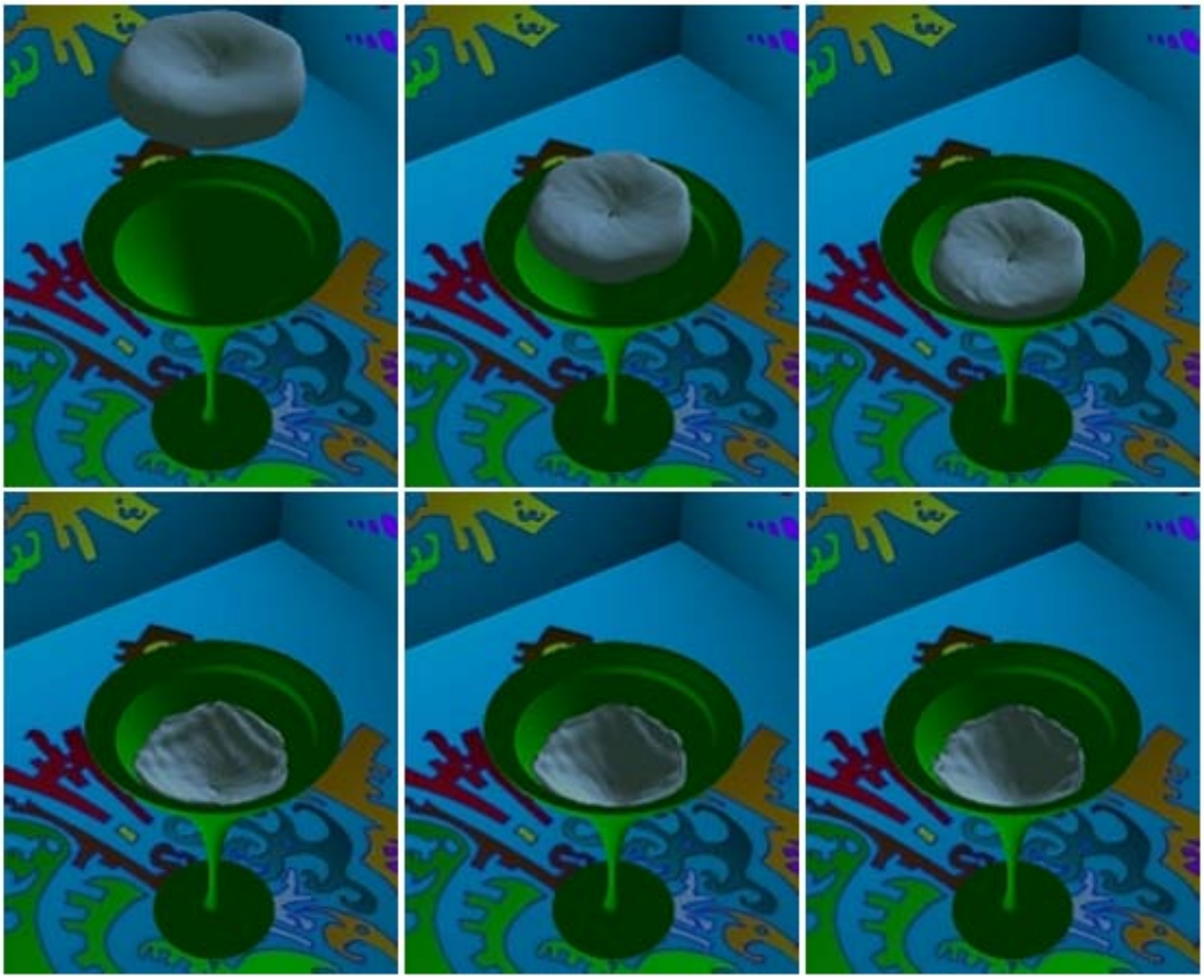

Figure 5—6 : The visual result of soft body model of experiment F, where the parameters α =1, β =1, γ =0, and δ =1 have been set. The result is captured at animated frames 100th, 1000th, 2000 ^{h}, 3000th, 4000th, and 6000th and it shows that these parameters generate the soft body without volume much like fabric or cloth.

5.1.2 Simulation of organic faces

In the second experiment, a human face is deformed in various ways by varying equation parameters as shown in table 5—2. The human face structure has 4,721 surface points, 9,213 faces, and 13,825 springs. Gravitational force is not applied in this experiment. The visual results of the experiments are depicted in figures 5—7 through 5—12 and demonstrate the effectiveness of the proposed algorithm at simulating deformed organic faces. In experiments A through C internal pressure coefficient is increased, resulting in a balloon effect with a smooth surface as shown in figures 5—7 through 5—9. In experiments D through F the internal pressure force is increased, however a higher fluid force results in a less bloated appearance and retains more of the original surface structure as shown in figures 5—10 through 5—12.

Comparing the results shown in figure 5—7 to figure 5—10 shows that in 5—10 the parameter of fluid force becomes dominant and holds the outline or shape of the model. The same relationship holds for the results shown in figures 5—8 and 5—11 and also between figures 5—9 and 5—12, even though higher internal pressure force is applied. Such deformations can be used for model morphing effects in graphics and games.

Table 5—2: Parameters for organic face deformation experiments.

Experiment	α	β	γ	δ	**FPS**
A	1	0	0.001	0	>15
B	1	0	0.005	0	>15
C	1	0	0.010	0	>15
D	1	0.075	0.001	0	>15
E	1	0.075	0.005	0	>15
F	1	0.075	0.010	0	>15

In each variation of the organic surface deformation experiments the equation parameters are varied to achieve different output characteristics. The parameters affect the following soft body parameters: (α) net body control, (β) net fluidity control, (γ) net volume control, and (δ) gravitational field. The visual results for each experiment are detailed in figures 5—7 through 5—12.

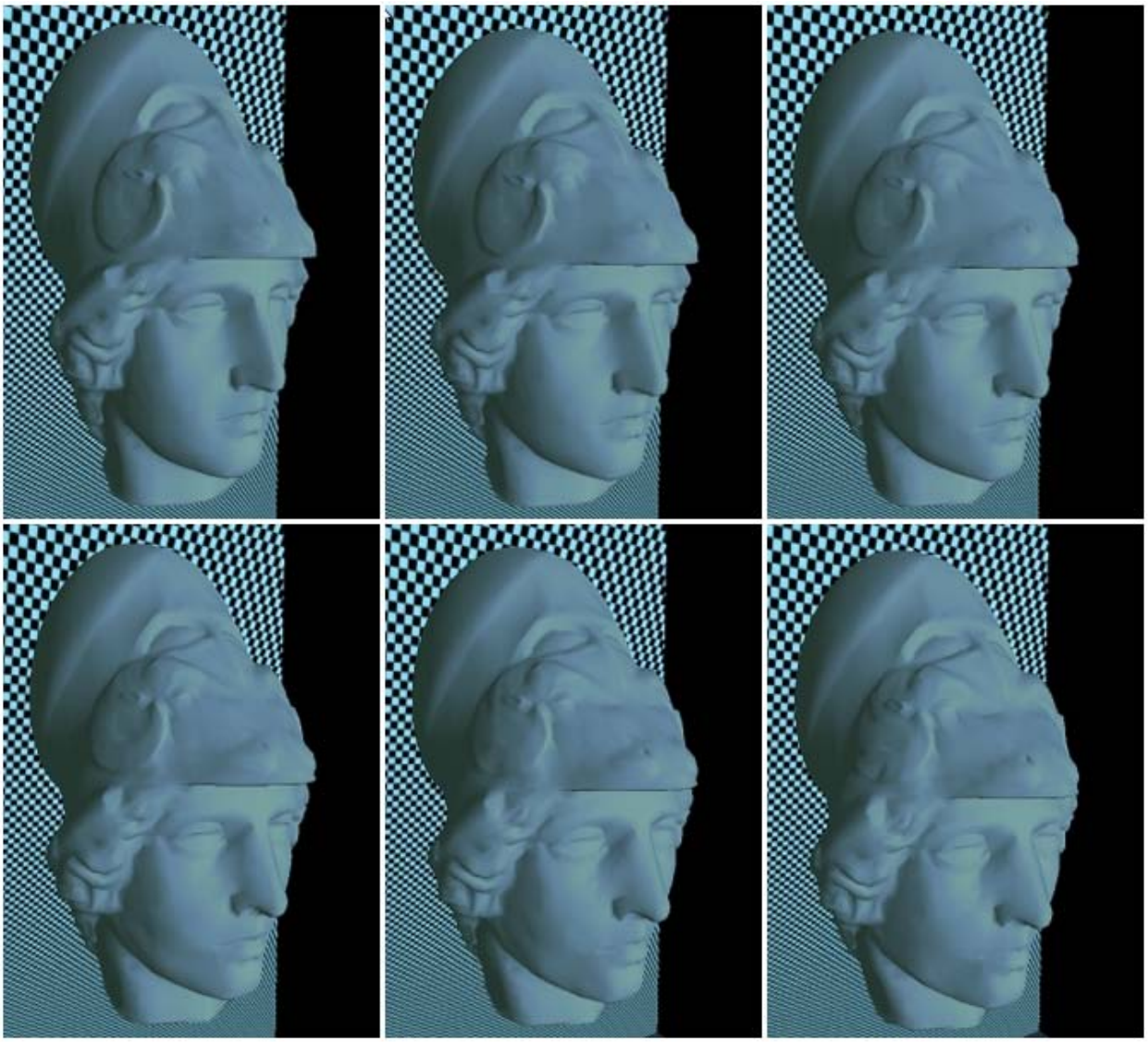

Figure 5—7 : The visual result of organic face of experiment A, where the parameters α =1, β =0, γ =0.001, and δ =0 have been set. The result is captured at animated frames 100th, 1000th, 2000^{h}, 3000th, 4000th, and 6000th and it shows that these parameters generate the organic surface with less balloon effect.

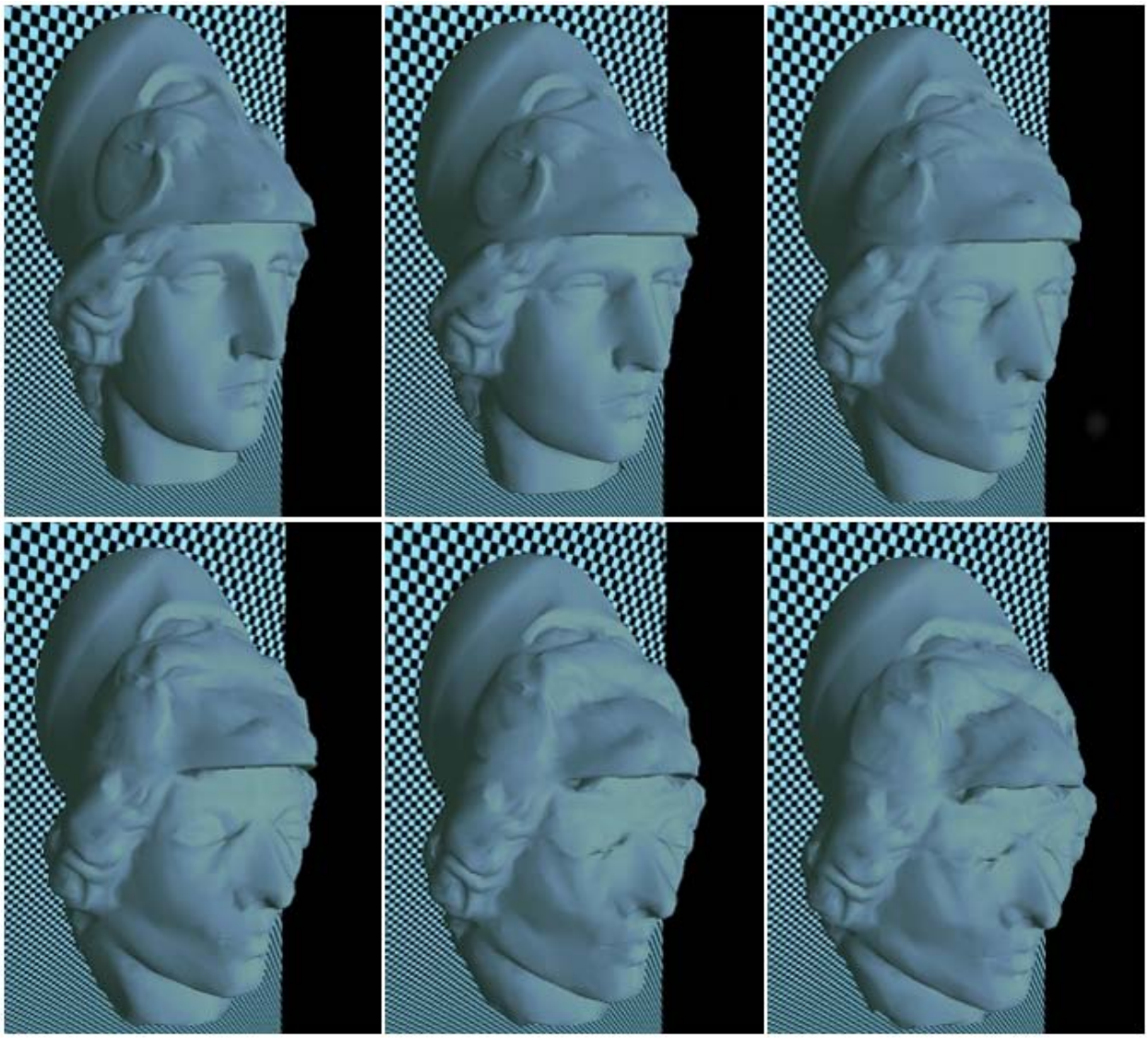

Figure 5—8 : The visual result of organic face of experiment B, where the parameters α =1, β =0, γ =0.005, and δ =0 have been set. The result is captured at animated frames 100th, 1000th, 2000^{h}, 3000th, 4000th, and 6000th and it shows that these parameters generate the organic surface with more balloon effect compared to experiment A but less balloon effect compared to experiment C.

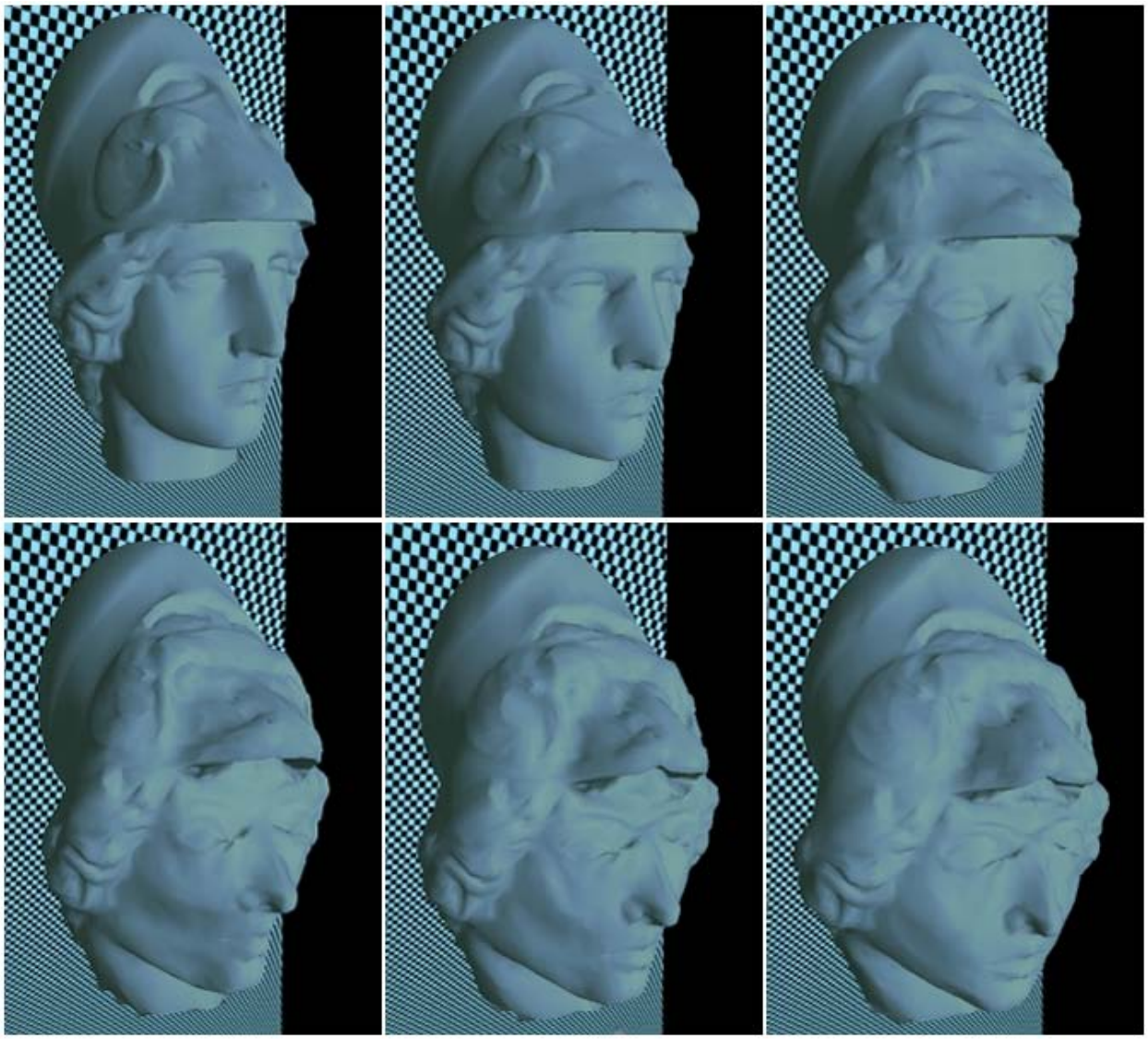

Figure 5—9 : The visual result of organic face of experiment C, where the parameters α =1, β =0, γ =0.010, and δ =0 have been set. The result is captured at animated frames 100th, 1000th, 2000^{h}, 3000th, 4000th, and 6000th and it shows that these parameters generate the organic surface with more balloon effect compared to experiments A and B.

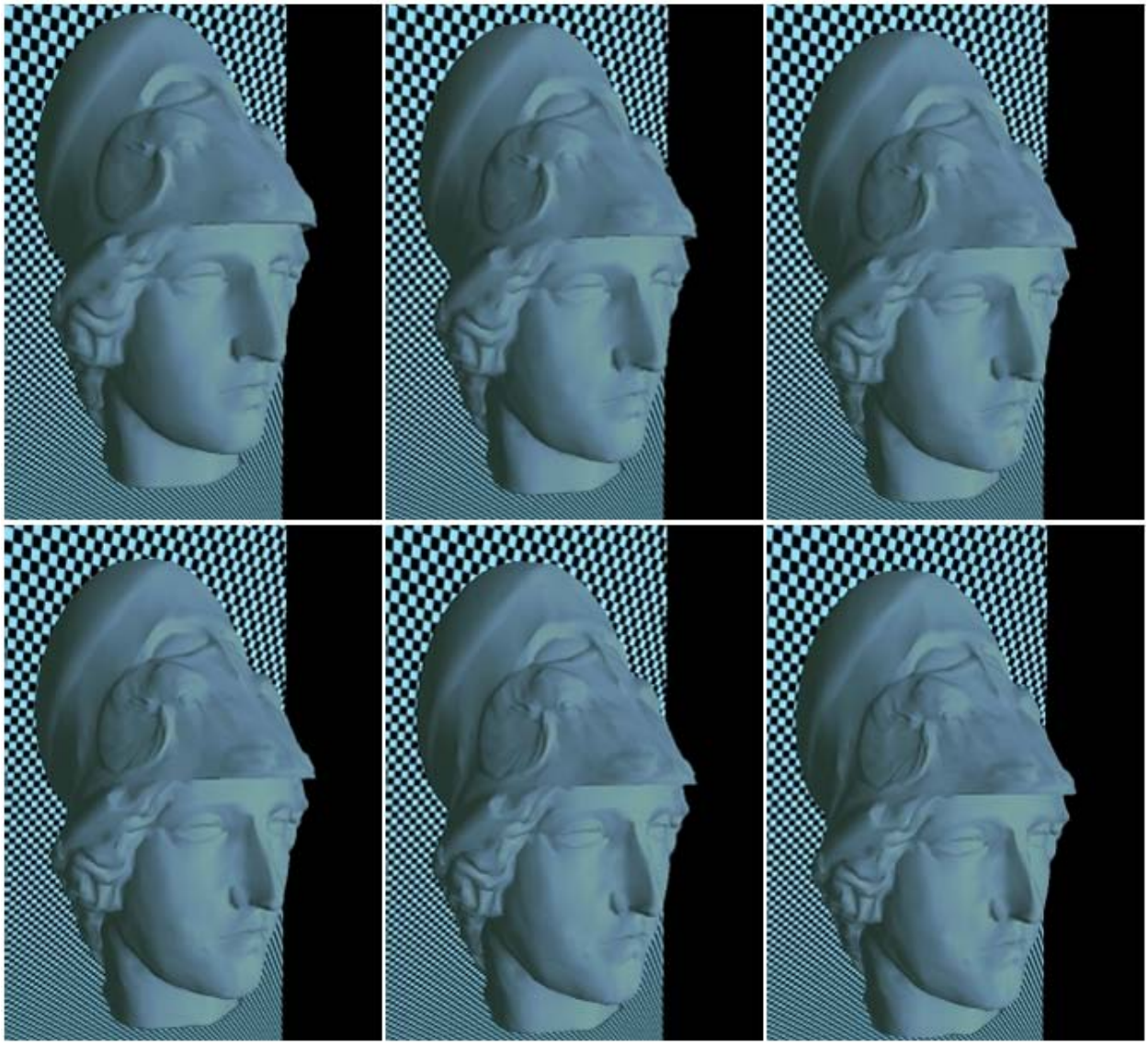

Figure 5—10 : The visual result of organic face of experiment D, where the parameters α =1, β =0.075, γ =0.001, and δ =0 have been set. The result is captured at animated frames 100th, 1000th, 2000 ^{h}, 3000th, 4000th, and 6000th and it shows that these parameters generate the organic surface with less bloated appearance compared to experiments E and F and retains more of the original surface structure compared to experiment A.

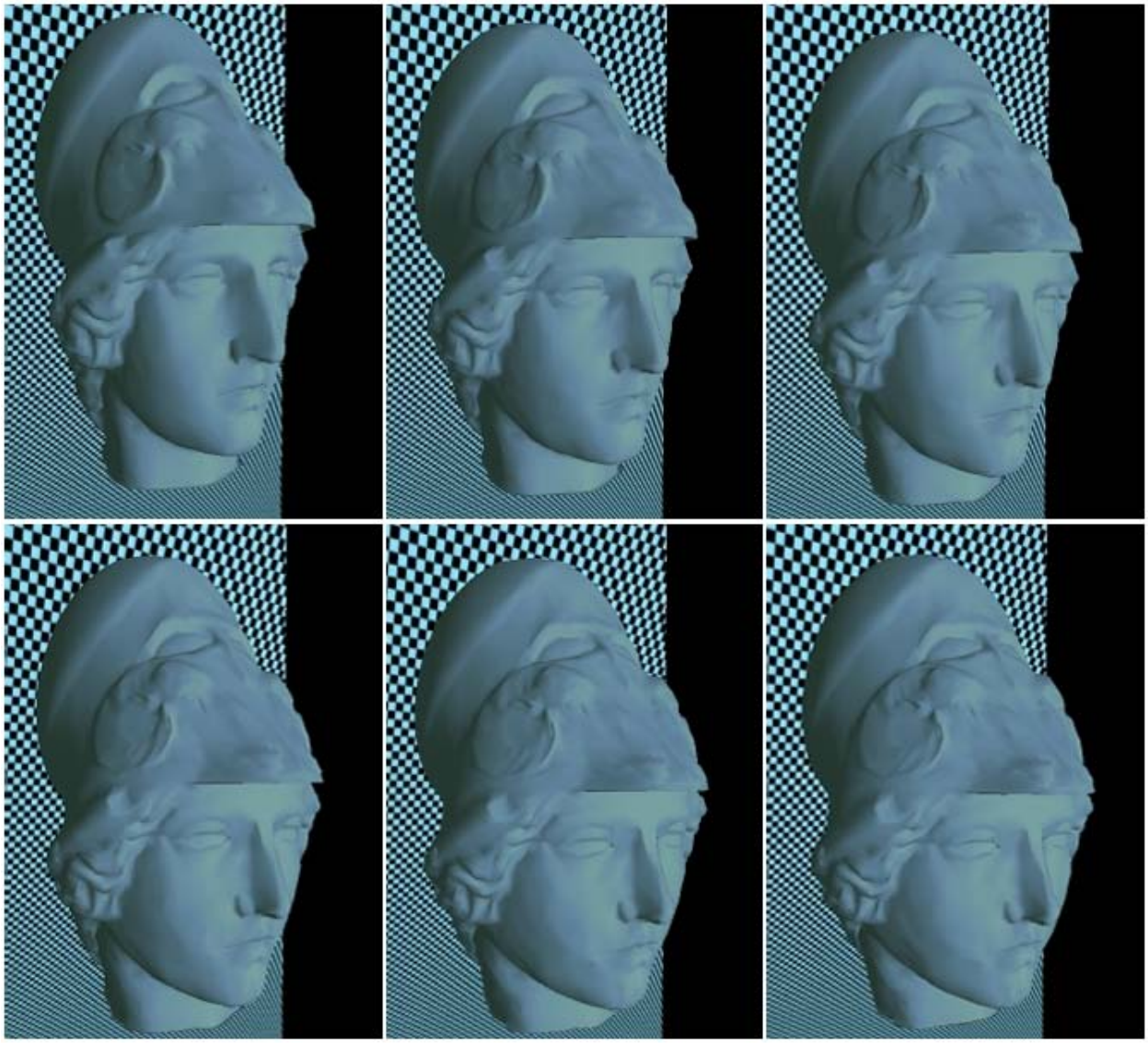

Figure 5—11 :The visual result of organic face of experiment E, where the parameters α =1, β =0.075, γ =0.005, and δ =0 have been set. The result is captured at animated frames 100th, 1000th, 2000 ^{h}, 3000th, 4000th, and 6000th and it shows that these parameters generate the organic surface with more bloated appearance compared to experiment D but less bloated appearance compared to experiment F and retains more of the original surface structure compared to B.

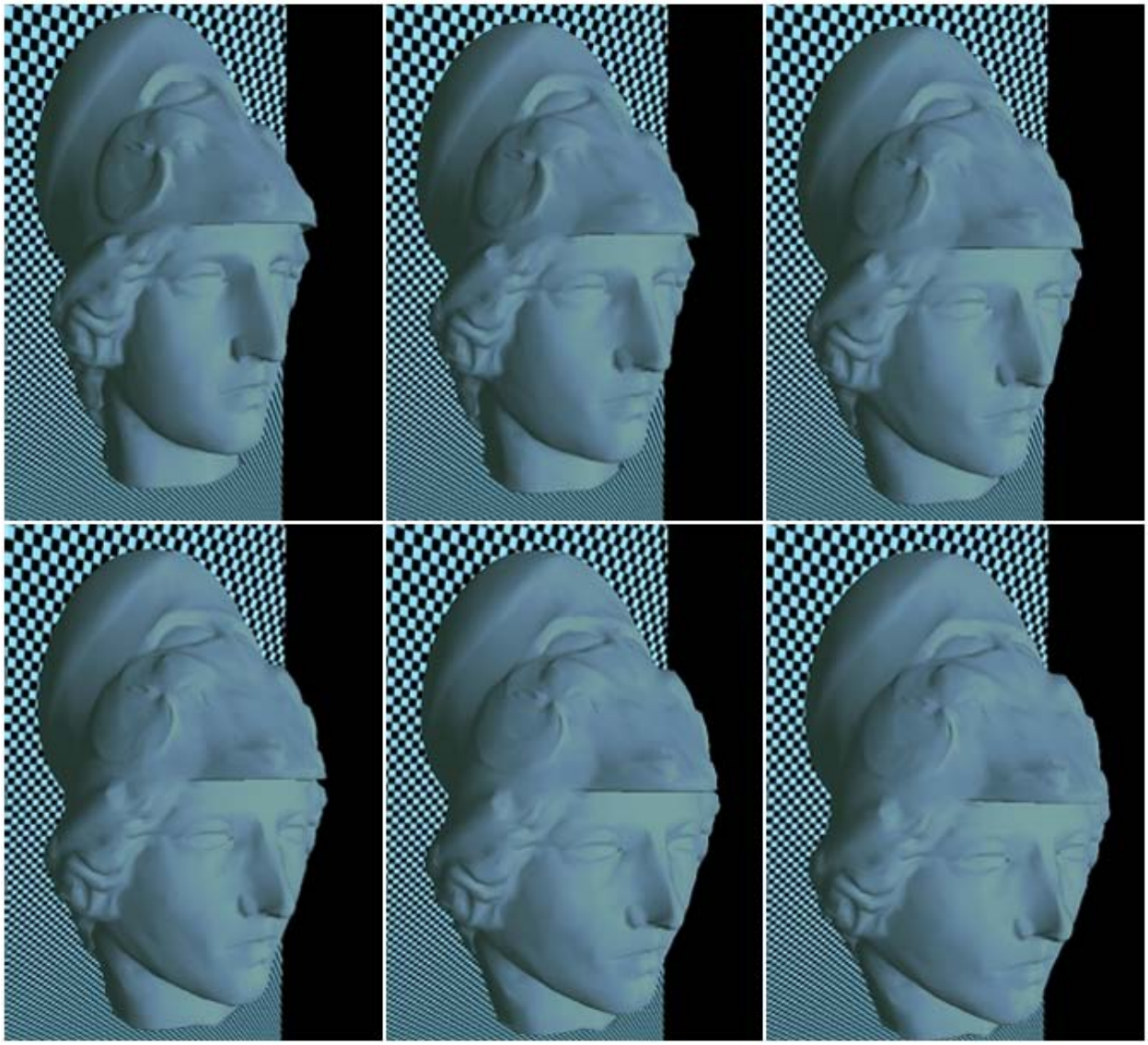

Figure 5—12 : :The visual result of organic face of experiment F, where the parameters α =1, β =0.075, γ =0.010, and δ =0 have been set. The result is captured at animated frames 100th, 1000th, 2000 ^{h}, 3000th, 4000th, and 6000th and it shows that these parameters generate the organic surface with more bloated appearance compared to experiments D and E and retains more of the original surface structure compared to C.

5.2 Integration into a game engine

While isolated simulations are useful for testing graphics and gaming technology, they do not accurately reflect the environment of video games which have strict time requirements and large additional processing burden from AI, sound, networking, physics, particles systems, and various other game components. Therefore, the proposed soft-body algorithm has been integrated into the Galactic Arms Race (GAR) game engine [104]. GAR is a multi-player space video game and a test bed for experimental game technology. Additional information and the GAR game demo is available at http://gar.eecs.ucf.edu.

In GAR, the proposed soft-body algorithm is used to animate and render "Space Blobs", which are large amorphous enemies (see figure 5—13). The 3D model for the Space Blob is a 162 vertex sphere. The sphere vertex count is chosen to be low enough for fast simulation of multiple creatures on low-end hardware, yet high enough so that the model does not look "blocky". With 162 vertices, an appropriate texture, and a lighting shader, the Space Blob creature looks quite convincing in game. To introduce a continual animated "breathing" effect in the blob, the soft body internal pressure is periodically randomized. The animated Space Blob enemies in the GAR game engine prove the effectiveness of the algorithm described in this research for gaming.

Figure 5—13 : Soft Body Monsters in Galactic Arms Race (GAR). The proposed soft-body algorithm animates and renders the large amorphous "Space Blob" enemies in the Galactic Arms Race video game (http://gar.eecs.ucf.edu). Varying internal pressure force creates a continual animated “breathing” effect.

The simulation of fluid-like and organic faces in this chapter demonstrates that a variety of soft body deformations can be produced by adjusting the parameter values. Additionally, implementation of the “Space Blobs” in the Galactic Arms Race shows the use of soft body models in video games. In the next chapter, the simulation of lung function shows that the proposed soft body model can be used in medical applications.

CHAPTER 6: SIMULATION OF LUNG RESPIRATION FUNCTION

Lung simulation has been used in medical imaging for training purposes. This lung simulation involves deformation methods simulated by geometrically-based or physically-based modeling [105].

Geometrically-based methods normally involve spherical harmonics (SP) where the angular portion of a set of solutions is used to transform the points of the model [106,107] during the respiration. These methods have been used not only in artificial approximation for model recognitions but also in predicting soft tissue deformation. However, the physiological basis cannot be achieved by this method especially for the simulation of a breathing lung.

Physically-based methods model the physiology of lungs. The lung functions, inhalation and exhalation, are simulated according to the functional representation such as mass-spring system or FEM [108,109]. In FEM, the airflow inside bronchioles is simulated through fluid dynamics to generate the forces that are applied to each element of FEM. The lung model is divided into the individual elements of the FEM, thus the computational complexity depends on the number of FEM elements. In order to reduce computational complexity, a mass-spring system is used to model lungs [105,110,111,112,113].

By using our general soft body model, an internal pressure model controls respiratory rhythm by using a constraint of Pressure-Volume (P-V) relation while the mass-spring system maintains the elastic behavior of the model and the fluid model generates surface deformation of lungs. Experiments of the parametric values of body control, volume control, and fluidity control are conducted to determine the suitable values of these parameters for realistic simulation. The

interaction between lungs and rib cage also illustrate the realism of our simulation as the lung volume changes due to this.

This chapter is structured as follows. Section 6.1 briefly describes the respiratory and lung functions. Section 6.2 discusses internal pressure in our lung respiration model. Section 6.3 presents how our soft body model is adapted for simulation of lung respiration. Section 6.4 shows the results of the lung simulation. Finally section 6.5 summarizes our soft body for lung simulation.

6.1 Respiration and lung functions

In this section we present the basic functions of the lungs and respiration. A brief explanation of the respiratory system, lung volume, and Pressure Volume (P-V) curve relation are described here.

6.1.1 Respiratory system

There are two alternative means of human respiration [114]. The first mean is breathing through the nose called nasal breathing and the second is breathing through mouth called oral breathing. Nasal breathing provides two advantages: filtration of particulate and humidification of inspired gas. Since the nasal septum and turbinate expand the surface area of mucosa, they increase the efficiency of evaporation and produce turbulent flow. Humidification by nasal breathing is more efficient than by mouth breathing. Unfortunately, nasal breathing experiences more airflow resistance than mouth breathing. This can be exacerbated when the respiration is obstructed by polyps, adenoids, or congestion of the nasal mucosa. The resistance of nasal

breathing may necessitate oral breathing. For example during exercise, oral breathing occurs because the required respiratory volume increases as the body needs more air flow into lungs.

6.1.2 Lung volume

Lung volume or lung capacity is related to the state of the respiratory cycle. During normal breathing, the average of total lung capacity is about 6 liters of air for human male and 4.7 liters of air for a human female [115]. Lung capacity is related to several other lung parameters. During respiration, it depends on Total lung capacity (TLC), Vital capacity (VC), Forced vital capacity (FVC), Tidal volume (Vt), Residual volume (RV), Expiratory reserve volume (ERV), Inspiratory reserve volume (IRV)**,** Functional residual capacity (FRC), and Inspiratory capacity (IC). *TLC* is the volume of air at the end of maximal inspiration. *VC* is the amount of air that can be forced out after the maximal inspiration and represents the completeness of expiration. *FVC* is similar to *VC* but represents a maximal value. *Vt* is the volume of air during inhalation and exhalation during normal respiratory. *RV* is the amount of air left in the lungs after a maximal exhalation. *ERV* is the amount of air that can be pushed out after the end of inhalation for normal respiratory. *IRV* is the additional air that can be inhaled after a normal tidal inhalation. *FRC* is the amount of air left in the lungs after a tidal exhalation during normal breathing. Finally, *IC* is the maximal volume that can be inspired following a normal expiration.

6.1.3 Pressure Volume (P-V) curve relation

A single inhalation and exhalation involves the movement of thoracic muscles including the muscles of the abdominal region, the diaphragm, and the intercostal muscles. The movement of diaphragm and chest wall forms a mechanical pump that drives air flow into and out of the lungs [112,116,117]. The change of pressure and volume is known as the Pressure-Volume (P-V) relation. The relation of P-V is shown in figure 6—1. This relationship indicates changes in pulmonary mechanics and reflects the respiratory parameters such as lung tissue properties [112].

In our lung simulation, we use this P-V curve to control the volume of lungs that generate the internal pressure force during inhalation and exhalation. In the next section we propose a adapting our soft body model using this P-V relation determined from experiment results [118,119] .

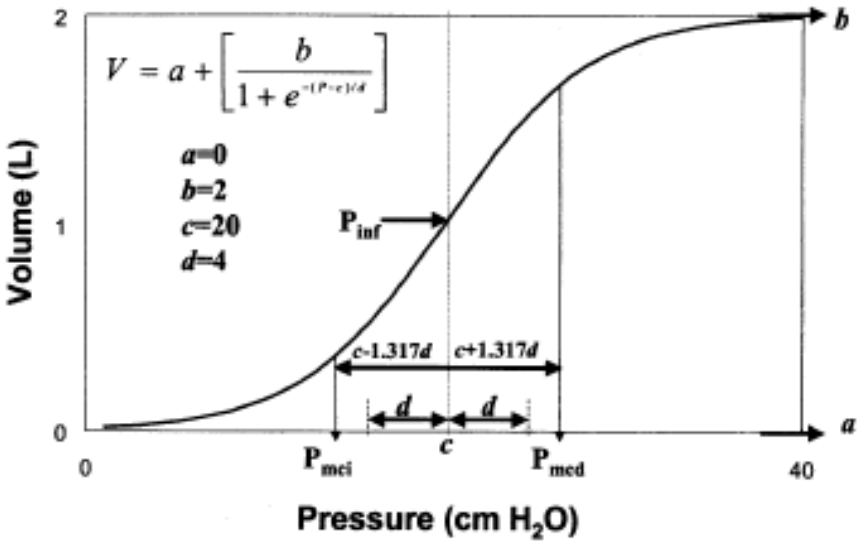

Figure 6—1: A sigmoidal for curve-fitting pressure-volume data in [118,119] (Sigmoidal P-V curve is discussed in 6.2).

6.2 Adapting internal pressure in lung respiration model

The lung model is simulated by adapting the internal pressure and volume of lung function as a constraint for our soft body model. We presented a soft body model by using mass-spring system to generate the elastic behavior of the model, the fluid modeling to present the surface deformation of the model tissue, the internal pressure to maintain the volume the soft body, the constraint to control the relation between pressure and volume, and finally gravitation to present the realism of physics.

Respiration is a function of the lung's internal pressure force, which controls the lung volume within certain limits. For an appropriately functioning lung model, we adapt this volume control function which depends on internal pressure of the lung. This volume control function specifies the internal pressure inside the lung which leads the air to flow in and out of the lung. The flow of air during inhalation and exhalation is simulated by the number of molecules in the internal pressure model in our simulation.

The internal pressure, $\boldsymbol{F}_v$, in lung simulation is presented as:

$$\boldsymbol{F}_v = \{N,V\},$$

where N is number of molecule and V is bounding volume of soft body.

In this simulation, the constraint is the function that controls the relation between pressure and volume (P-V). We adopted the P-V curve using the sigmoidal equation modeled from the experimental pressure volume data. The sigmoidal form of a P-V model equation is presented in [120] and based on observed performance of both healthy humans, dogs and, humans who have had lung surgery [121]. The P-V data has further been analyzed in acute

respiratory distress syndrome in [119]. We use this sigmoidal equation shown in figure 6—1 to generate lung volumes during inhalation and exhalation. The sigmoidal model in P-V curve is presented by:

$$V = a + \left[\frac{b}{1 + e^{-(P-c)/d}} \right],$$

where P is the respective pressure of the respiratory system, a is the lower asymptote volume, b is the vital capacity, c is the point of maximal compliance (true inflection point), and d is the pressure range that includes most of the volume change.

During inhalation or exhalation, we compute the volume of the lung model by increasing or decreasing the pressure of the respiratory system. The volume function of lung model during inhalation or exhalation is given by the constraints, $C_i(P,V)$ for inhalation and $C_e(P,V)$ for exhalation.

$C_i(P,V)$ is given by:

$$V_t = a + \left[\frac{b}{1 + e^{(P_t + \Delta P - c)/d}} \right], \text{ and}$$

$C_e(P,V)$ is given by:

$$V_t = a + \left[\frac{b}{1 + e^{(P_t - \Delta P - c)/d}} \right],$$

where P_t is the pressure of the respiratory system at the current time, ΔP is the pressure difference in the current frame and the next frame, and $c = \{c_{inh}, c_{exh}\}$ are the points of maximal compliance; c_{inh} is the point for inhalation and c_{exh} is the point for exhalation.

In the next section, we discuss how we use this volume control function in our soft body model to simulate inhalation and exhalation.

6.3 Lung respiratory simulation

In this simulation the visualization of diaphragm is does not move the position of the rib cage; however, we approximate the movement and position of the rib cages from the number of animated frames during inhalation and exhalation. The interaction between lungs and rib cages is achieved by point-based collision.

We approximate the fixed number of frames for the state of inhalation or exhalation at each respiratory cycle. However, it can be changed according to the inhalation or exhalation of a specific individual. The internal pressure force models the inhalation and exhalation of the lung volume according to $\{ +\Delta P, c_{inh}, N_{inh} \}$ and $\{ -\Delta P, c_{exh}, N_{exh} \}$, respectively, where ΔP = 40/(number of frames for one inhalation or exhalation), c_{inh} = 20, c_{exh} = 10, N_{inh} = 70k, N_{exh} =100 in this simulation. The gravitational force is exerted on all surface points of the lung for the real-world physics effects. Finally, all the forces are combined and velocities are evaluated for the new surface positions.

After updating new positions of all lung surface points, we perform the collision detection between the lung model and the rib cage. If a collision occurs, the effects of collision on the point positions are calculated. After surface positions have been checked for collision and have been assigned to new positions from the effect of collision, the display function displays all

surface points of both lung models and rib cage. These steps repeat until the user stops the simulation.

6.4 Experiments

We applied the soft body model, where the mass spring system controls the body of the model, the internal pressure produces the inhalation and exhalation activities of lung functions, and the fluid modeling generates the surface deformation. To maintain the realistic physics, the gravitational force is applied at each surface point. The experiments on body control, volume control, and fluid modeling show the effects of the soft body parameters. The parameter values, body control by mass spring (α), fluid modeling (β), volume control by internal pressure (γ), and gravitation (δ), are set in table 6—1 to present the effect of body control, table 6—2 to demonstrate the effect of volume control, and table 6—3 to show the effect of fluidity control.

6.4.1 Determining the dependency of the body control on lung model

This experiment observes the dependency of body control on the lung model to control the volume of lung within the limit. The parameter of body control by mass spring (α), is set to 0.25, 1, 3, and 4 while the fluid modeling (β), and volume control by internal pressure (γ) are fixed to 1. For the realism of physics we set the gravitational parameter (δ) to 0.1 because we need the gravitation force to maintain the center of the lungs. The parameter value sets are shown in table 6—1.

Table 6—1: Parameter values of the experiment on effect of the mass-spring

Experiment	α	β	γ	δ
A	0.25	1	1	0.1
B	1	1	1	0.1
C	3	1	1	0.1
D	4	1	1	0.1

The lung volumes of five respiratory cycles in each mass spring parameter are shown in figure 6—2. This experiment shows that even though the body control determine the structure of the model, it can affect the volume of the lung model. Thus, a good parameter value needs be selected to provide the realistic lung simulation. In this experiment, α =3 provides a lung volume under control during inhalation and exhalation. The visualizations of lung simulation during inhalation and exhalation for the parameter sets, A, B, C, and D are shown in figures (6—3 and 6—4), (6—5 and 6—6), (6—7 and 6—8), and (6—9 and 6—10), respectively. Figures 6—3, 6—5, 6—7, and 6—9 show the sequences of images from the beginning to the end of inhalation while figures 6—4, 6—6, 6—8, and 6—10 present the sequences of images from the beginning to the end of exhalation. Figures 6—3 and 6—4 show that with the body control parameter of 0.25, the lung model at the end of exhalation (the rightmost in figure 6—4) does not deform back to the volume at the beginning of inhalation (the leftmost in figure 6—3). It maintains some volume even though it reaches the end of exhalation and should be in an initial state to start a new cycle of inhalation and exhalation. Figures 6—5 and 6—6 show that with the body control

parameter of 1.0, the lung model at the end of exhalation (the rightmost in figure 6—6) deforms back to the original volume (the leftmost in figure 6—5) better than the result presented in figures 6—3 and 6—4. However it still maintains some volume at the end of exhalation. Figures 6—7 and 6—8 illustrate that with the body control parameter of 3.0, the lung model at the end of exhalation (the rightmost in figure 6—8) deforms back to the original volume (the leftmost in figure 6—7), which provides the best result compared to two previous parameter values. Figures 6—9 and 6—10 demonstrate that with the body control parameter of 4.0, the lung model at the end of exhalation (the rightmost in figure 6—10) deforms smaller than original volume (the leftmost in figure 6—9) compared to three previous parameter values. Thus, the best parameter value for body control is α =3.

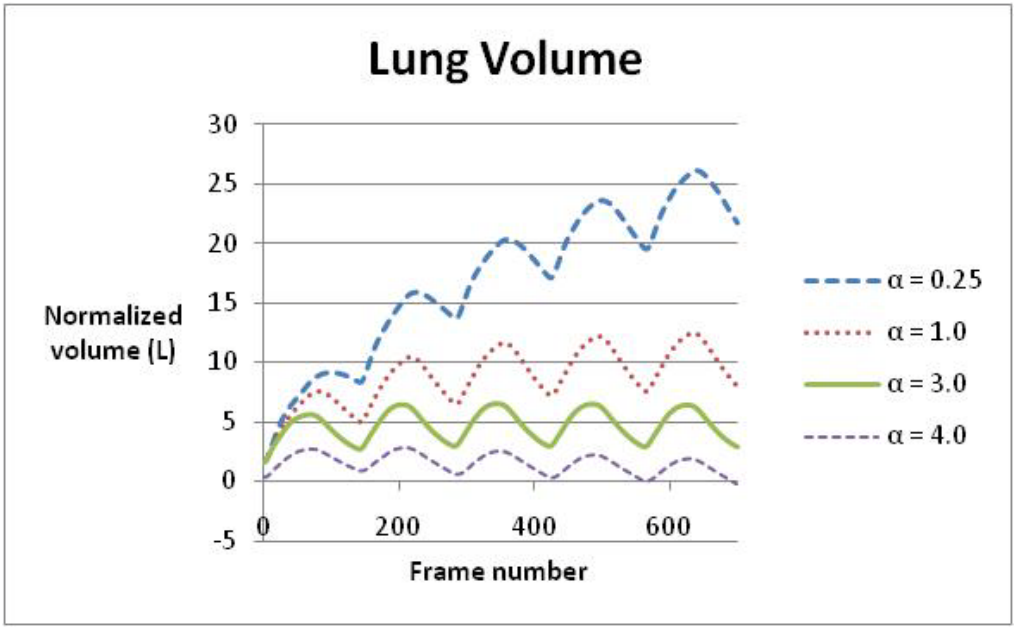

Figure 6—2: Lung volumes resulted from the experiment parameter sets in table 6—1.

.

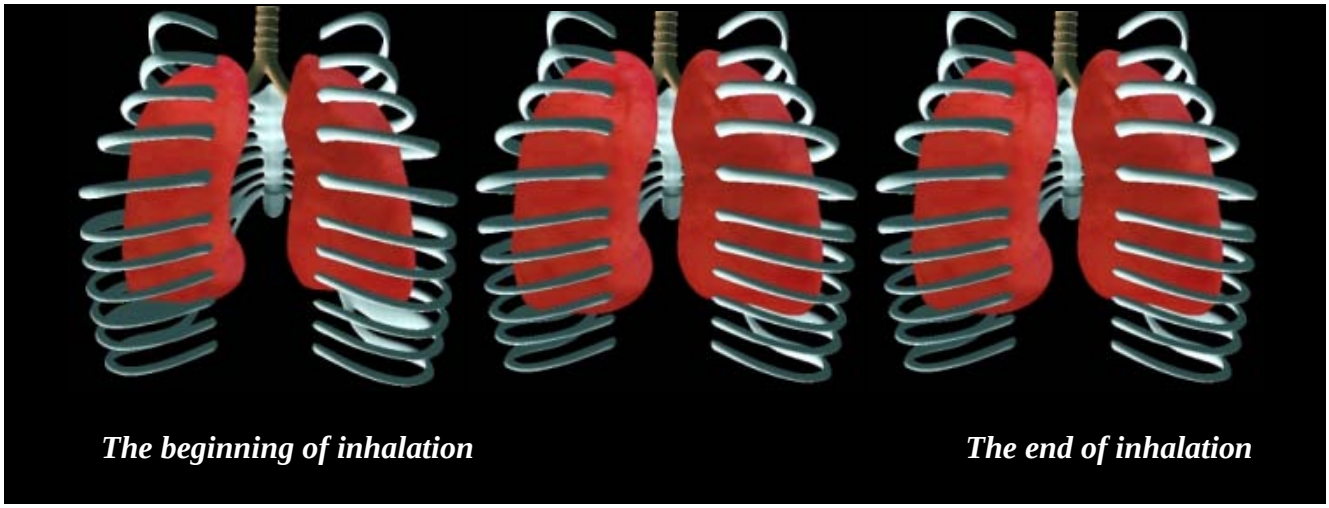

Figure 6—3: The visual result of lungs from the beginning to the end of inhalation (left to right) when parameters of body control, fluidity control, volume control, and gravity are set to 0.25, 1, 1, and 0.1, respectively.

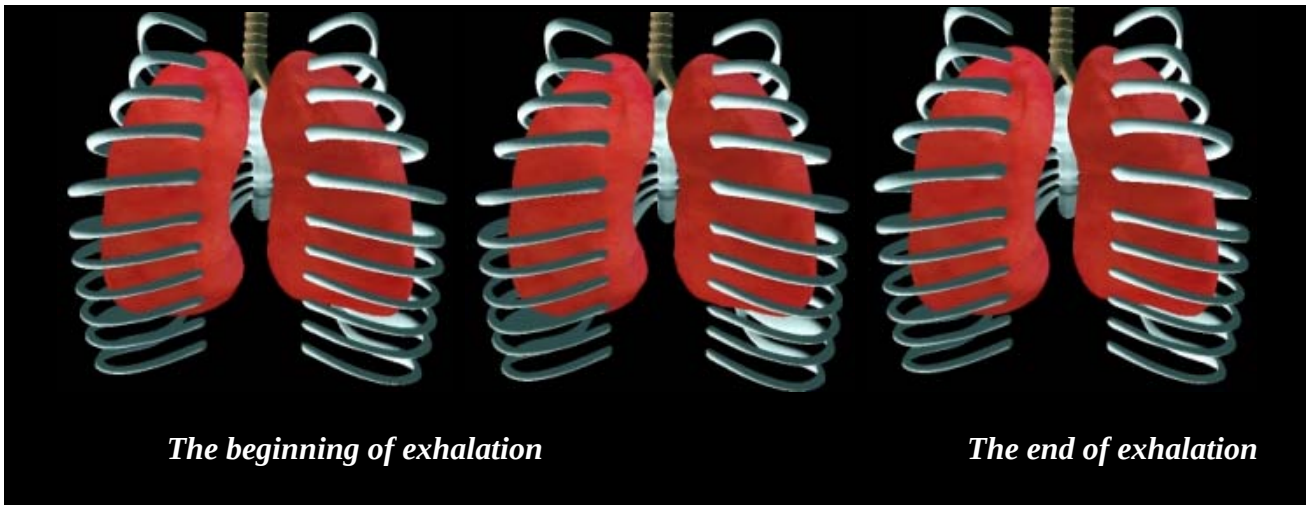

Figure 6—4: The visual result of lungs from the beginning to the end of exhalation (left to right) when parameters of body control, fluidity control, volume control, and gravity are set to 0.25, 1, 1, and 0.1, respectively.

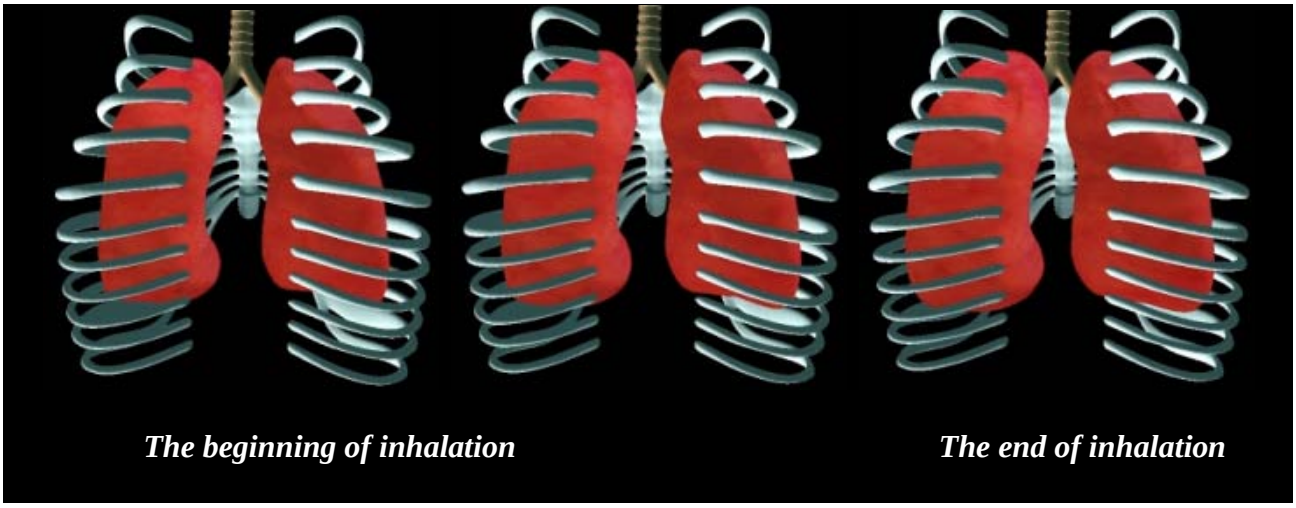

Figure 6—5: The visual result of lungs from the beginning to the end of inhalation (left to right) when parameters of body control, fluidity control, volume control, and gravity are set to 1, 1, 1, and 0.1, respectively.

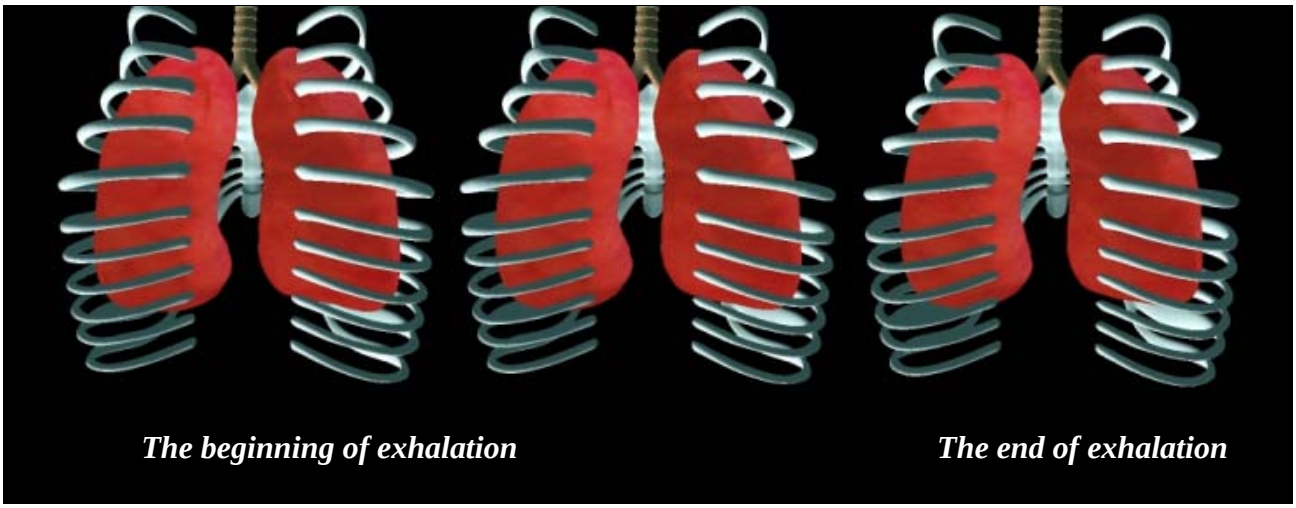

Figure 6—6: The visual result of lungs from the beginning to the end of exhalation (left to right) when parameters of body control, fluidity control, volume control, and gravity are set to 1, 1, 1, and 0.1, respectively.

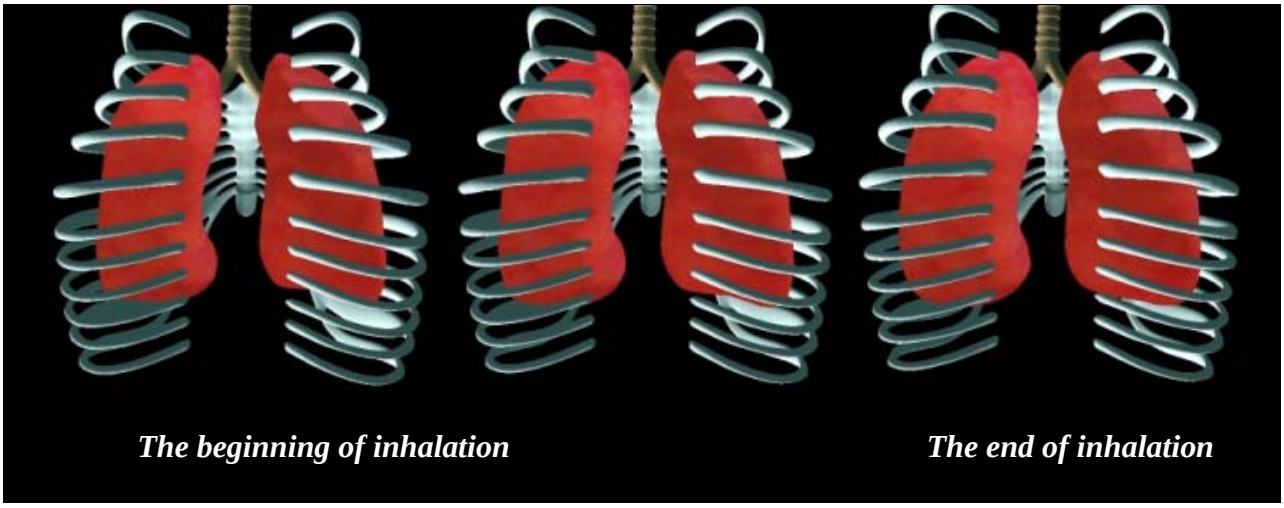

Figure 6—7: The visual result of lungs from the beginning to the end of inhalation (left to right) when parameters of body control, fluidity control, volume control, and gravity are set to 3, 1, 1, and 0.1, respectively.

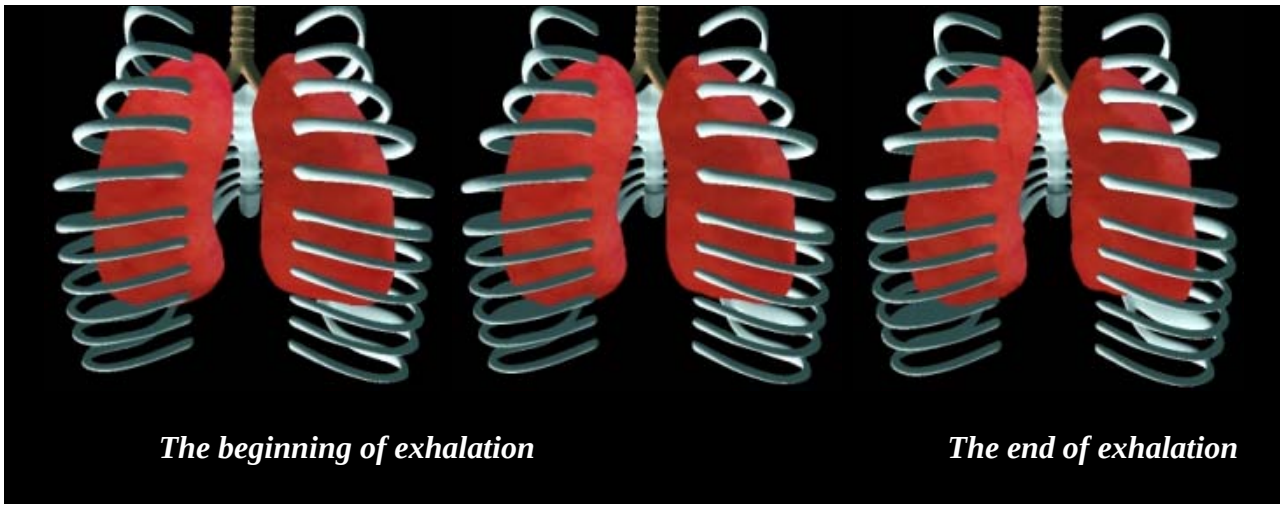

Figure 6—8: The visual result of lungs from the beginning to the end of exhalation (left to right) when parameters of body control, fluidity control, volume control, and gravity are set to 3, 1, 1, and 0.1, respectively.

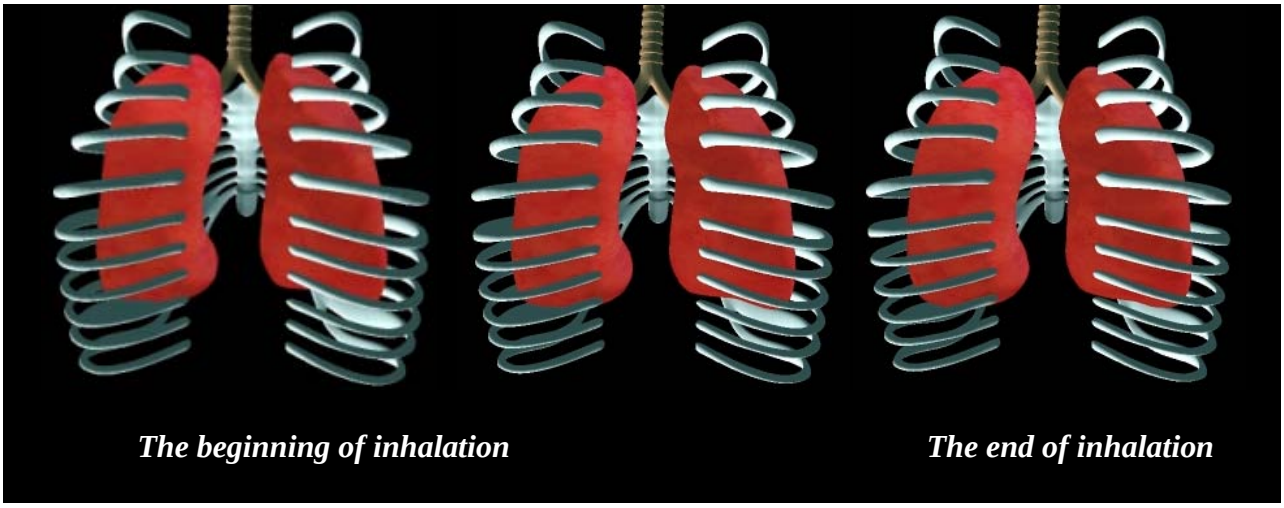

Figure 6—9: The visual result of lungs from the beginning to the end of inhalation (left to right) when parameters of body control, fluidity control, volume control, and gravity are set to 4, 1, 1, and 0.1, respectively.

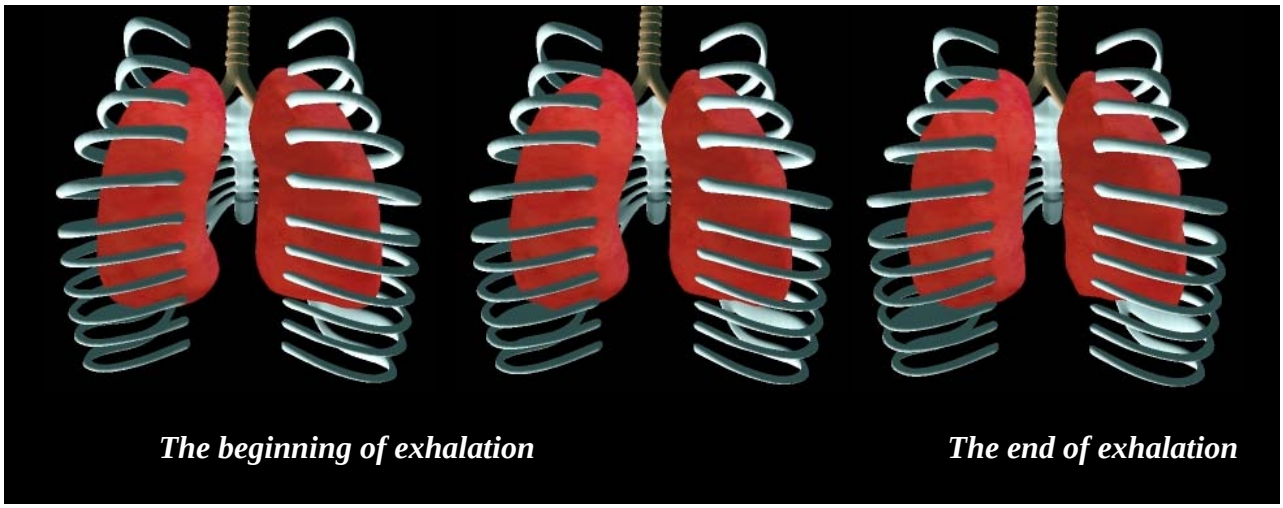

Figure 6—10: The visual result of lungs from the beginning to the end of exhalation (left to right) when parameters of body control, fluidity control, volume control, and gravity are set to 4, 1, 1, and 0.1, respectively.

6.4.2 Determining the effect of the volume control

This experiment observes the dependency of volume control on the lung model to control the volume of lung beyond the effect of body control. Since the best result is given when the body control parameter is set to 3.0, the parameter of body control by mass-spring, (α), is fixed to 3 in this experiment. Similar to the previous experiment, fluid modeling, (β), and gravitational parameter, (δ**),** are set to 1 and 0.1, respectively. Volume control by internal pressure parameter, (γ), is varied from 1.0, 1.5, and 2.0 to investigate the effect of the internal pressure. The parameter value sets are shown in table 6—2.

Table 6—2: Parameter values of the experiment on effect of the internal pressure.

Experiment	α	β	γ	δ
A	3	1	1	0.1
B	3	1	1.5	0.1
C	3	1	2	0.1

The result of this experiment is shown on the graph of the lung volumes in figure 6—11. In this simulation, we ignore the transient values of first cycle of respiration which are due to stabilization of the model as it interacts with the forces of the simulation. We consider the remaining cycles of the simulation. When internal pressure parameter is set to 1.0, the volume of inhalation and exhalation is the same as the result in the body control parameter set in C that provides the best result among those parameter sets. When the internal pressure parameter is set

to 1.5, the volume of the lung is increased for both inhalation and exhalation compared to the volume when the internal pressure is set to 1.0. However, at the end of exhalation the volume does not reduce back to the volume at the beginning of inhalation. Similarly, when the internal pressure is set to 2.0, the volume does not decrease back to the beginning of inhalation. Thus, in this experiment the best parameter for the volume control is γ =1.0.

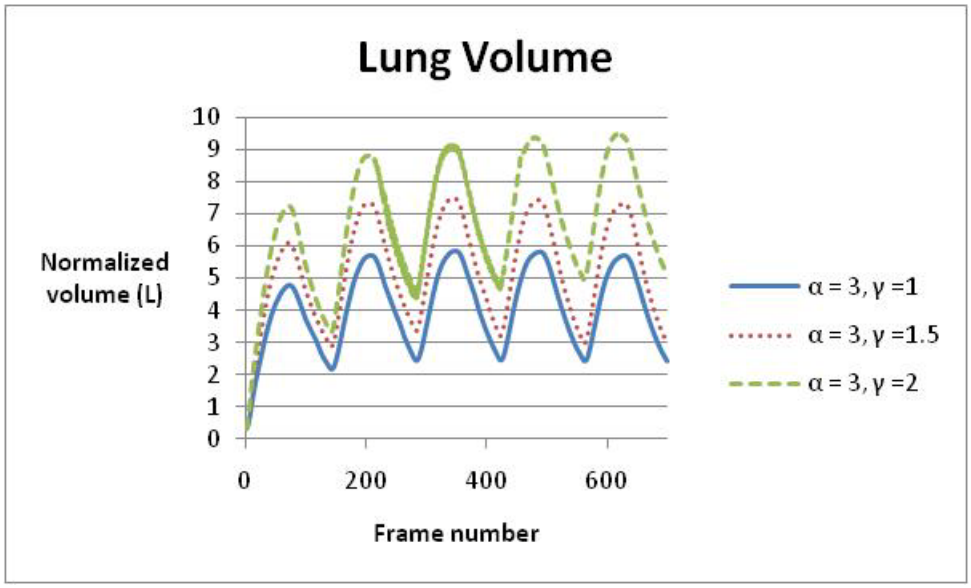

Figure 6—11: Lung volumes resulted from the experiment parameter sets in table 6—2

6.4.3 Determining the effect of the fluidity control

This experiment measures the effect of fluidity control to create the bumpiness of the lung surface. The parameter of body control by mass-spring, (α) and volume control by internal pressure, (γ) are fixed to 3 and 1, respectively, while the fluid control, (β), is varied from 0.25,

0.5, 0.75, 1, and 2. Similar to the previous experiment, the gravitational parameter (δ) is set to 0.1. The parameter value sets are shown in table 6—3.

Table 6—3: Parameter values of the experiment on effect of the fluid modeling

Experiment	α	β	γ	δ
A	3	0.25	1	0.1
B	3	0.5	1	0.1
C	3	0.75	1	0.1
D	3	1	1	0.1
E	3	2	1	0.1

The visual results of each fluid parameter value are shown in figures 6—12. Figure 6—12 is created using a gradient map tool to present the bumpiness of the lung surface. This figure shows that the bumpiness of the surface is increased when the fluid modeling value is increased. Thus, the fluidity control parameter effects the surface deformation of the lung. This parametric value can be selected to vary with the condition of lung's surface.

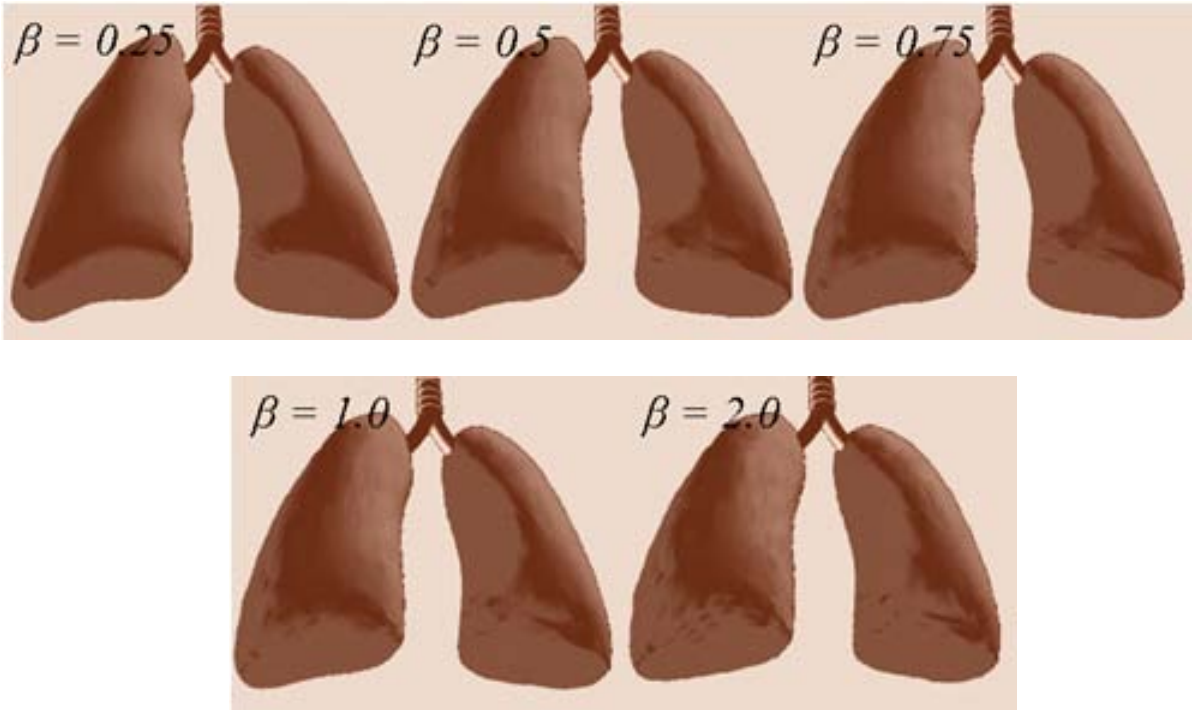

Figure 6—12: The visual result with gradient map of lung simulation for each fluidity control parameter value.

6.5 Lung simulation conclusion

We have utilized the adapting soft body model based on the experimental results of pressure-volume curve of lung function. The parameters of body control and volume control have been examined to generate the realistic simulation of lung's deformation during respiration. The result shows that our soft body model can simulate the lung function appropriately with certain parametric values of lung model. Thus, the best parameter values can be set for certain condition of lungs. In this simulation, the detection of the rib cages is also considered using point-based collision detection. By using this collision detection, the lung simulation becomes

more realistic. In future works, more lung simulations under various disease conditions should be simulated by using our soft body model.

CHAPTER 7: CONCLUSION AND FUTURE WORK

The soft body model presented in this dissertation is a generalized soft body simulation method that integrates and extends specific cases of other existing models in order to mimic a wide variety of real-world non-solid objects. User defined characteristics such as body control, surface deformation, volume control, constraints, gravitation, and combined forces can be customized by users to realistically simulate an intended model. In this technique, body control enables the surface deformation while maintaining a relative configuration among surface points, fluid modeling helps create realistic fluid-like motion on the soft body surface, and volume within the 3D model is maintained by simulated gas molecule pressure. User-defined constraints can influence the amount of deformation, while gravitational force provides natural free fall motion. Additionally, a custom partitioning and hashing scheme for both fluid modeling and collision detection significantly reduces computation time necessary for interactive applications.

The experiments presented demonstrate simulation of fluid-like and organic faces show that the algorithm can produce a variety of soft body surfaces. Implementation of the "Space Blobs" in the Galactic Arms Race video game shows not only the algorithm's effectiveness in real-time game engines, but also that creative use of soft body models can help to increase the realism of video games. The human lung simulation demonstrates that the algorithm is suitable for medical applications.

There are several distinct possibilities for future work including, simulation of other internal organs for medical applications, splitting models in real-time for surgical simulations,

and further exploration of soft-body models in interactive entertainment, in both standard video game genres or perhaps in an experimental game consisting entirely of soft-body objects.

LIST OF REFERENCES

[1] Y. Kang, J. Choi, and H. Cho, "Fast and stable animation of cloth with an approximated implicit method," *In Proceedings Computer Graphics International*, 2000.

[2] D. Baraff and A. Witkin, *Large steps in cloth simulation*, New York, New York, USA: ACM Press, 1998.

[3] D.E. Breen, D.H. House, and M.J. Wozny, "Predicting the drape of woven cloth using interacting particles," *Proceedings of the 21st annual conference on Computer graphics and interactive techniques - SIGGRAPH '94*, New York, New York, USA: ACM Press, 1994, pp. 365-372.

[4] M. Carignan, Y. Yang, N.M. Thalmann, and D. Thalmann, "Dressing animated synthetic actors with complex deformable clothes," *Proceedings of the 19th annual conference on Computer graphics and interactive techniques - SIGGRAPH '92*, New York, New York, USA: ACM Press, 1992, pp. 99-104.

[5] K. Ward, N. Galoppo, and M. Lin, "A Simulation-based VR System for Interactive Hairstyling," *IEEE Virtual Reality Conference (VR 2006)*, IEEE, 2006, pp. 257-260.

[6] K. Ward, F. Bertails, T. Kim, S.R. Marschner, M. Cani, and M.C. Lin, "A survey on hair modeling: styling, simulation, and rendering," *IEEE transactions on visualization and computer graphics*, vol. 13, 2007, pp. 213-34.

[7] K. Ward, N. Galoppo, and M. Lin, "Interactive Virtual Hair Salon," *Presence: Teleoperators & Virtual Environments*, vol. 16, 2007, pp. 237-251.

[8] D. Terzopoulos and K. Fleischer, "Modeling Inelastic Deformation: Viscoelasticity, Plasticity, Fracture," *Computer Graphics*, vol. 22, 1988, pp. 269-278.

[9] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer, *Elastically deformable models*, New York, New York, USA: ACM Press, 1987.

[10] I. Costa and R. Balaniuk, "LEM-an approach for real time physically based soft tissue simulation," *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, IEEE, 2001, pp. 2337-2343.

[11] S. Cotin, H. Delingette, and N. Ayache, "Real-time elastic deformations of soft tissues for surgery simulation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, 1999, pp. 62-73.

[12] N. Foster and R. Fedkiw, "Practical animation of liquids," *Proceedings of the 28th annual conference on Computer graphics and interactive techniques - SIGGRAPH '01*, New York, New York, USA: ACM Press, 2001, pp. 23-30.

[13] D. Hinsinger, F. Neyret, and M. Cani, "Interactive animation of ocean waves," *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation - SCA '02*, New York, New York, USA: ACM Press, 2002, p. 161.

[14] J. Kim, D. Cha, B. Chang, B. Koo, and I. Ihm, "Practical animation of turbulent splashing water," *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2006, pp. 335-344.

[15] F. Losasso, T. Shinar, A. Selle, and R. Fedkiw, "Multiple interacting liquids," *ACM Transactions on Graphics*, vol. 25, 2006, p. 812.

[16] W.E. Lorensen and H.E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *ACM SIGGRAPH Computer Graphics*, vol. 21, 1987, pp. 163-169.

[17] H. Mao and Y. Yang, "Particle-based immiscible fluid-fluid collision," *GI '06: Proceedings of Graphics Interface 2006*, Toronto, Ont., Canada, Canada: Canadian Information Processing Society, 2006, pp. 49-55.

[18] J. Teran, S. Blemker, V.N. Hing, and R. Fedkiw, "Finite volume methods for the simulation of skeletal muscle," *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2003, pp. 68-74.

[19] P. Volino, N. Thalmann, and D. Thalmann, "An evolving system for simulating clothes on virtual actors," *IEEE Computer Graphics and Applications*, vol. 16, 1996, pp. 42-51.

[20] P. Volino, M. Courchesne, and N. Magnenat Thalmann, "Versatile and efficient techniques for simulating cloth and other deformable objects," *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques - SIGGRAPH '95*, New York, New York, USA: ACM Press, 1995, pp. 137-144.

[21] G. Debunne, M. Cani, M. Desbrun, and A. Barr, "Adaptive Simulation of Soft Bodies in Real-Time," *CA '00: Proceedings of the Computer Animation*, Washington, DC, USA: IEEE Computer Society, 2000, p. 15.

[22] E. Keeve, S. Girod, and B. Girod, "Craniofacial Surgery Simulation," *VBC '96: Proceedings of the 4th International Conference on Visualization in Biomedical Computing*, London, UK: Springer-Verlag, 1996, pp. 541-546.

[23] A. Santhanam, S. Pattanaik, J.P. Roll, C. Imielinska, and B. Informatics, "Physiologically-based Modeling and Visualization of Deformable Lungs," *In Proceedings of the 11th Pacific Conference on Computer Graphics and Applications (PG03*, 2003.

[24] M. Carignan, Y. Yang, N.M. Thalmann, and D. Thalmann, "Dressing animated synthetic actors with complex deformable clothes," *Proceedings of the 19th annual conference on Computer graphics and interactive techniques - SIGGRAPH '92*, New York, New York, USA: ACM Press, 1992, pp. 99-104.

[25] X. Provot, "Deformation constraints in a mass-spring model to describe rigid cloth behavior," 1995.

[26] B. Eberhardt, A. Weber, and W. Strasser, "A fast, flexible, particle-system model for cloth draping," *IEEE Computer Graphics and Applications*, vol. 16, 1996, pp. 52-59.

[27] D. Breen, M.L. (editors, K.S. Bhat, C.D. Twigg, J.K. Hodgins, S. M., P.K. Khosla, Z. Popovic, and S.M. Seitz, "Estimating Cloth Simulation Parameters from Video," *Eurographics/SIGGRAPH symposium on Computer Animation 2003*, 2003, pp. 37-51.

[28] R. Bridson, S. Marino, and R. Fedkiw, "Simulation of clothing with folds and wrinkles," *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2003, pp. 28-36.

[29] R. Bridson, R. Fedkiw, and J. Anderson, "Robust treatment of collisions, contact and friction for cloth animation," *ACM SIGGRAPH 2005 Courses on - SIGGRAPH '05*, New York, New York, USA: ACM Press, 2005, p. 2.

[30] P. Decaudin, D. Julius, J. Wither, L. Boissieux, A. Sheffer, and M. Cani, "Virtual Garments: A Fully Geometric Approach for Clothing Design," *Computer Graphics Forum*, vol. 25, 2006, pp. 625-634.

[31] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff, "Position based dynamics," *Journal of Visual Communication and Image Representation*, vol. 18, 2007, pp. 109-118.

[32] O. Ozgen, M. Kallmann, L.E. Ramirez, and C.F. Coimbra, "Underwater cloth simulation with fractional derivatives," *ACM Transactions on Graphics*, vol. 29, 2010, pp. 1-9.

[33] J. McCartney, "Dedicated 3D CAD for garment modelling," *Journal of Materials Processing Technology*, vol. 107, 2000, pp. 31-36.

[34] Z. LUO and M. YUEN, "Reactive 2D/3D garment pattern design modification," *Computer-Aided Design*, vol. 37, 2005, pp. 623-630.

[35] N. Metaaphanon and P. Kanongchaiyos, "Real-time cloth simulation for garment CAD," *Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia - GRAPHITE '05*, New York, New York, USA: ACM Press, 2005, p. 83.

[36] T.W. Sederberg and S.R. Parry, "Free-form deformation of solid geometric models," *ACM SIGGRAPH Computer Graphics*, vol. 20, 1986, pp. 151-160.

[37] J.E. Chadwick, D.R. Haumann, and R.E. Parent, "Layered construction for deformable animated characters," *Proceedings of the 16th annual conference on Computer graphics and interactive techniques - SIGGRAPH '89*, New York, New York, USA: ACM Press, 1989, pp. 243-252.

[38] M. Müller, J. Dorsey, L. McMillan, R. Jagnow, and B. Cutler, *Stable real-time deformations*, New York, New York, USA: ACM Press, 2002.

[39] D.L. James and D.K. Pai, *DyRT: Dynamic Response Textures for Real Time Deformation Simulation with Graphics Hardware*, 2002.

[40] K.K. Hauser, C. Shen, and J.F. Brien, "Interactive Deformation Using Modal Analysis with Constraints," *GRAPHICS INTERFACE*, 2003.

[41] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross, and M. Alexa, "Point Based Animation of Elastic, Plastic and Melting Objects," *Most*, 2004, pp. 141-151.

[42] M. Teschner, B. Heidelberger, M. Muller, and M. Gross, "A versatile and robust model for geometrically complex deformable solids," *Proceedings Computer Graphics International, 2004.*, IEEE, 2004, pp. 312-319.

[43] L. Verlet, "Computer "Experiments" on Classical Fluids. II. Equilibrium Correlation Functions," *Physical Review*, vol. 165, 1968, pp. 201-214.

[44] M. Müller and M. Gross, "Interactive virtual materials," *GI '04: Proceedings of Graphics Interface 2004*, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada: Canadian Human-Computer Communications Society, 2004, pp. 239-246.

[45] M. Müller, B. Heidelberger, M. Teschner, and M. Gross, "Meshless deformations based on shape matching," *ACM Trans. Graph.*, vol. 24, 2005, pp. 471-478.

[46] M. Botsch, M. Pauly, M. Gross, and L. Kobbelt, "PriMo: coupled prisms for intuitive surface modeling," *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing*, Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2006, pp. 11-20.

[47] N. Galoppo, M.A. Otaduy, S. Tekin, M. Gross, and M.C. Lin, "Soft Articulated Characters with Fast Contact Handling," *Computer Graphics Forum*, vol. 26, 2007, pp. 243-253.

[48] J. Gourret, N.M. Thalmann, and D. Thalmann, "Simulation of object and human skin formations in a grasping task," *ACM SIGGRAPH Computer Graphics*, vol. 23, 1989, pp. 21-30.

[49] Y. Lee, D. Terzopoulos, and K. Walters, "Realistic modeling for facial animation," *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques - SIGGRAPH '95*, New York, New York, USA: ACM Press, 1995, pp. 55-62.

[50] F. Scheepers, R.E. Parent, W.E. Carlson, and S.F. May, "Anatomy-based modeling of the human musculature," *Proceedings of the 24th annual conference on Computer graphics and interactive techniques - SIGGRAPH '97*, New York, New York, USA: ACM Press, 1997, pp. 163-172.

[51] S. Capell, M. Burkhart, B. Curless, T. Duchamp, and Z. Popović, "Physically based rigging for deformable characters," *Graphical Models*, vol. 69, 2007, pp. 71-87.

[52] N. Stoiber, R. Seguier, and G. Breton, "Facial animation retargeting and control based on a human appearance space," *Computer Animation and Virtual Worlds*, vol. 21, 2010, pp. 39-54.

[53] M. Desbrun, P. Schröder, and A. Barr, "Interactive animation of structured deformable objects," *Proceedings of the 1999 conference on Graphics interface '99*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 1-8.

[54] X. Provot, "Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior," *In Graphics Interface*, 1996, pp. 147-154.

[55] A. Fuhrmann, C. Groß, and V. Luckas, "Interactive Animation of Cloth including Self Collision Detection," *Journal of WSCG*, vol. 11, 2003, pp. 203-208.

[56] M. Meyer, G. Debunne, M. Desbrun, and A.H. Barr, "Interactive Animation of Cloth-like Objects in Virtual Reality," *The Journal of Visualization and Computer Animation*, vol. 12, 2000, pp. 1-12.

[57] K. Waters, "Physical model of facial tissue and muscle articulation derived from computer tomography data," *Proceedings of SPIE*, SPIE, 1992, pp. 574-583.

[58] M. Teschner, S. Girod, and B. Girod, "Direct Computation of Nonlinear Soft-Tissue Deformation," *Vision, Modeling, and Visualization VMV'00*, 2000, pp. 383-390.

[59] M. Castaneda and F. Cosio, "Computer simulation of prostate resection for surgery training," *Proceedings of the 25th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, IEEE, 2003, pp. 1152-1155.

[60] D. Nixon and R. Lobb, "A fluid-based soft-object model," *IEEE Computer Graphics and Applications*, vol. 22, 2002, pp. 68-75.

[61] M. Müller, L. McMillan, J. Dorsey, and R. Jagnow, "Real-time simulation of deformation and fracture of stiff materials," *Proceedings of the Eurographic workshop on Computer animation and simulation*, New York, NY, USA: Springer-Verlag New York, Inc., 2001, pp. 113-124.

[62] I. Oguz, W. Baxter, M.C. Lin, and J.D. Wendt, "Finite volume flow simulations on arbitrary domains," *Graph. Models*, vol. 69, 2005, p. 2007.

[63] M. Müller, D. Charypar, and M. Gross, "Particle-based fluid simulation for interactive applications," *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2003, pp. 154-159.

[64] M. Müller, B. Solenthaler, R. Keiser, and M. Gross, "Particle-based fluid-fluid interaction," *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, New York, NY, USA: ACM, 2005, pp. 237-244.

[65] J.J. Monaghan, "Smoothed Particle Hydrodynamics," *Annual Review of Astronomy and Astrophysics*, vol. 30, 1992, pp. 543-574.

[66] R.A. Gingold and J.J. Monaghan, "Smoothed particle hydrodynamics - Theory and application to non-spherical stars," *Royal Astronomical Society*, vol. 181, 1977, pp. 375-389.

[67] J.P. Morris, "Simulating surface tension with smoothed particle hydrodynamics," *International Journal for Numerical Methods in Fluids*, vol. 33, 2000, pp. 333-353.

[68] D. Nixon, "A fluid-based soft object model," 1999.

[69] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipes in C*, The Art of Scientific Computing, 2007.

[70] S. Nakamura, *In Applied Numerical Methods with Software*, Prentice-Hall, 1991.

[71] P. Hubbard, "Collision detection for interactive graphics applications," *IEEE Transactions on Visualization and Computer Graphics*, vol. 1, 1995, pp. 218-230.

[72] D.L. James and D.K. Pai, "BD-tree: output-sensitive collision detection for reduced deformable models," *ACM SIGGRAPH 2004 Papers on - SIGGRAPH '04*, New York, New York, USA: ACM Press, 2004, p. 393.

[73] L. Kavan, C. O'Sullivan, and J. Žára, "Efficient collision detection for spherical blend skinning," *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia - GRAPHITE '06*, New York, New York, USA: ACM Press, 2006, p. 147.

[74] C. MENDOZA and C. OSULLIVAN, "Interruptible collision detection for deformable objects," *Computers & Graphics*, vol. 30, 2006, pp. 432-438.

[75] G. van den Bergen, "Efficient collision detection of complex deformable models using AABB trees," *J. Graph. Tools*, vol. 2, 1997, pp. 1-13.

[76] C. Mendoza, I.N. (editors, R. Weller, G. Zachmann, and T. Clausthal, "Kinetic Separation Lists for Continuous Collision Detection of Deformable Objects," *In 3rd Workshop in Virtual Reality Interactions and Physical Simulation VRIPHYS 2006*, 2006.

[77] S. Gottschalk, M.C. Lin, and D. Manocha, "OBBTree: A Hierarchical Structure for Rapid Interference Detection," *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques - SIGGRAPH '96*, New York, New York, USA: ACM Press, 1996, pp. 171-180.

[78] T. He, "Fast collision detection using QuOSPO trees," *Proceedings of the 1999 symposium on Interactive 3D graphics - SI3D '99*, New York, New York, USA: ACM Press, 1999, pp. 55-62.

[79] J. Klosowski, M. Held, J. Mitchell, H. Sowizral, and K. Zikan, "Efficient collision detection using bounding volume hierarchies of k-DOPs," *IEEE Transactions on Visualization and Computer Graphics*, vol. 4, 1998, pp. 21-36.

[80] M.A. Otaduy and M.C. Lin, "CLODs: dual hierarchies for multiresolution collision detection," *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2003, pp. 94-101.

[81] J. Erickson, L.J. Guibas, J. Stolfi, and L. Zhang, "Separation-sensitive collision detection for convex objects," *SODA '99: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1999, pp. 327-336.

[82] M. Moore and J. Wilhelms, "Collision detection and response for computer animation," *Proceedings of the 15th annual conference on Computer graphics and interactive*

techniques - SIGGRAPH '88, New York, New York, USA: ACM Press, 1988, pp. 289-298.

[83] B. Naylor, J. Amanatides, and W. Thibault, "Merging BSP trees yields polyhedral set operations," *Proceedings of the 17th annual conference on Computer graphics and interactive techniques - SIGGRAPH '90*, New York, New York, USA: ACM Press, 1990, pp. 115-124.

[84] R. Baumann and D. Glauser, "Force feedback for virtual reality based minimally invasive surgery simulator," *Medecine Meets Virtual Reality*, 1996.

[85] M. Held, J.T. Klosowski, and J.S. Mitchell, "Evaluation of Collision Detection Methods for Virtual Reality Fly-Throughs," *In Canadian Conference on Computational Geometry*, 1995, pp. 205-210.

[86] F. Ganovelli, J. Dingliana, and C. O'Sullivan, "BucketTree: Improving collision detection between deformable objects," *Proceedings Spring Conference on Computer graphics SCCG'00*, 2000.

[87] J. HUANG, X. LIU, H. BAO, B. GUO, and H. SHUM, "An efficient large deformation method using domain decomposition," *Computers & Graphics*, vol. 30, 2006, pp. 927-935.

[88] N. GOVINDARAJU, I. KABUL, M. LIN, and D. MANOCHA, "Fast continuous collision detection among deformable models using graphics processors," *Computers & Graphics*, vol. 31, 2007, pp. 5-14.

[89] N. BANDI, C. SUN, D. AGRAWAL, and A. ELABBADI, "Fast computation of spatial selections and joins using graphics hardware☆," *Information Systems*, vol. 32, 2007, pp. 1073-1100.

[90] A. Greß, M. Guthe, and R. Klein, "GPU-based Collision Detection for Deformable Parameterized Surfaces," *Computer Graphics Forum*, vol. 25, 2006, pp. 497-506.

[91] S. Cotin, H. Delingette, and N. Ayache, "Real-time elastic deformations of soft tissues for surgery simulation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, 1999, pp. 62-73.

[92] M. Teschner, B. Heidelberger, M. Mueller, D. Pomeranets, and M. Gross, "Optimized Spatial Hashing for Collision Detection of Deformable Objects," 2003, pp. 47-54.

[93] N. Galoppo, S. Tekin, M.A. Otaduy, M. Gross, and M.C. Lin, "Interactive haptic rendering of high-resolution deformable objects," *ICVR'07: Proceedings of the 2nd*

international conference on Virtual reality, Berlin, Heidelberg: Springer-Verlag, 2007, pp. 215-233.

[94] S. Rodriguez and N. Amato, "Planning motion in completely deformable environments," *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, IEEE, , pp. 2466-2471.

[95] R. Alterovitz and K. Goldberg, "Comparing Algorithms for Soft Tissue Deformation: Accuracy Metrics and Benchmarks," 2003.

[96] K.K. Hauser, C. Shen, and J.F. OBrien, "Interactive Deformation Using Modal Analysis with Constraints," *IN GRAPHICS INTERFACE*, 2003, pp. 247-256.

[97] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross, and M. Alexa, "Point based animation of elastic, plastic and melting objects," *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2004, pp. 141-151.

[98] M. Matyka and M. Ollila, "Pressure Model of Soft Body Simulation," 2003.

[99] D.M. Bourg, *Physics for game developers*, 2001.

[100] J. Mesit, R. Guha, and S. Chaudhry, "3D Soft Body Simulation Using Mass-spring System with Internal Pressure Force and Simplified Implicit Integration," *Journal of Computers*, vol. 2, 2007.

[101] J. Mahovsky and B. Wyvill, "Fast Ray-Axis Aligned Bounding Box Overlap Tests with Plücker Coordinates," *journal of graphics, gpu, and game tools*, vol. 9, 2004, pp. 35-46.

[102] C. Tzafestas and P. Coiffet, "Real-time collision detection using spherical octrees: virtual reality application," *Proceedings 5th IEEE International Workshop on Robot and Human Communication. RO-MAN'96 TSUKUBA*, IEEE, , pp. 500-506.

[103] J. Mesit and R. Guha, "Experimenting with Real Time Simulation Parameters for Fluid Model of Soft Bodies," *ANSS2010*, 2010.

[104] E. Hastings, R. Guha, and K. Stanley, "Automatic Content Generation in the Galactic Arms Race Video Game," *IEEE Transactions on Computational Intelligence and AI in Games 2009*, vol. 1, 2009.

[105] A.P. Santhanam, F.G. Hamza-Lup, and J.P. Rolland, "Simulating 3-D Lung Dynamics Using a Programmable Graphics Processing Unit," *IEEE Transactions on Information Technology in Biomedicine*, vol. 11, 2007, pp. 497-506.

[106] W. Segars, D. Lalush, and B. Tsui, "Modeling respiratory mechanics in the MCAT and spline-based MCAT phantoms," *IEEE Transactions on Nuclear Science*, vol. 48, 2001, pp. 89-97.

[107] J. Lötjönen, I.E. Magnin, L. Reinhardt, J. Nenonen, and T. Katila, "Model, Automatic Reconstruction Of 3D Geometry Using Projections And A Geometric Prior," *In IEEE Transactions on Medical Imaging*, vol. 18, 1999, pp. 992-1002.

[108] M. Bro-Nielsen and S. Cotin, "Real-time Volumetric Deformable Models for Surgery Simulation using Finite Elements and Condensation," *Computer Graphics Forum*, vol. 15, 1996, pp. 57-66.

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

[110] A. Santhanam, C. Fidopiastis, P. Davenport, K. Langen, S. Meeks, and J. Rolland, "Real-Time Simulation and Visualization of Subject-Specific 3D Lung Dynamics," *19th IEEE Symposium on Computer-Based Medical Systems (CBMS'06)*, IEEE, 2006, pp. 629-634.

[111] A. Santhanam, S. Pattanaik, J. Rolland, C. Imielinska, and J. Norfleet, "Physiologically-based Modeling and visualization of deformable lungs," *11th Pacific Conference onComputer Graphics and Applications*, IEEE Comput. Soc, 2003, pp. 507-511.

[112] A. Santhanam, "Modeling, simulation, and visualization of three-dimensional lung dynamics," *Ph.D. Thesis, University of Central Florida*, 2003.

[113] V.B. Zordan, B. Celly, B. Chiu, and P.C. DiLorenzo, "Breathe easy: Model and control of human respiration for computer animation," *Graphical Models*, vol. 68, 2006, pp. 113-132.

[114] A.B. Lumb, *Nunn's Applied Respiratory Physiology*, Butterworth-Heinemann, 2005.

[115] A.J. B. Palsson, J.A. Hubbell, R. Plonsey, *Tissue Engineering (Principles and Applications in Engineering)*, CRC Press, 2003.

[116] J. Cotes, *Lung function assessment and application in medicine*, Blackwell Scientific Publication, 1993.

[117] R. Harris, "Pressure-Volume Curves of the Respiratory System," *RESPIRATORY CARE*, vol. 50, 2005.

[118] R. Harris, "Pressure-volume curves of the respiratory system," *RESPIRATORY CARE*, vol. 50, 2005, pp. 78-98.

[119] R. Harris, D. Hess, and J. Venegas, "An objective analysis of the pressure-volume curve in the acute respiratory distress syndrome," *Am J Respir Crit Care Med*, vol. 161, 2000, pp. 432-439.

[120] J.G. Venegas, S. Harris, and B.A. Simon, "A comprehensive equation for the pulmonary pressure-volume curve," *J Apply Physiol*, vol. 84, 1998, pp. 389-395.

[121] U. Narusawa, "General characteristics of the sigmoidal model equation representing quasi-static pulmonary P-V curves," *J Appl Physiol*, vol. 91, 2001, pp. 201-210.