DYNAMIC DEFORMATION OF UNIFORM ELASTIC TWO-LAYER OBJECTS

Miao Song

A thesis
in
The Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science
Concordia University
Montréal, Québec, Canada

August 2007

Your file Votre référence
ISBN: 978-0-494-34780-5
Our file Notre référence
ISBN: 978-0-494-34780-5

Abstract

Dynamic Deformation of Uniform Elastic Two-Layer Objects

Miao Song

This thesis presents a two-layer uniform facet elastic object for real-time simulation based on physics modeling method. It describes the elastic object procedural modeling algorithm with particle system from the simplest one-dimensional object, to more complex two-dimensional and three-dimensional objects.

The double-layered elastic object consists of inner and outer elastic mass spring surfaces and compressible internal pressure. The density of the inner layer can be set different from the density of the outer layer; the motion of the inner layer can be opposite to the motion of the outer layer. These special features, which cannot be achieved by a single layered object, result in improved imitation of a soft body, such as tissue's liquidity non-uniform deformation. The construction of the double-layered elastic object is closer to the real tissue's physical structure.

The inertial behavior of the elastic object is well illustrated in environments with gravity and collisions with walls, ceiling, and floor. The collision detection is defined by elastic collision penalty method and the motion of the object is guided by the Ordinary Differential Equation computation.

Users can interact with the modeled objects, deform them, and observe the response to their action in real time.

Acknowledgments

This thesis is made possible by these important people in my life:

I would like to thank Dr. Peter Grogono, my supervisor, for his sure guidance, careful and knowledgeable support, and his patience. I also want to thank Prof. Jason Lewis for his constant encouragement. Special thanks to Ms. Catherine LeBel, my immediate superior, and the CSLP (Center for the Study of Learning and Performance) software development team, who I work with, for their able support. Many thanks to Serguei Mokhov for his sturdy belief in the completion of this thesis and for introducing me to many application software tools. I would also like to thank my cherished little treasure, my daughter Deschanel, for her faith in me. Finally, I must thank my parents, brother, and all of my loved ones for their care and support throughout my studies.

Contents

List of Figures

Chapter 1

Introduction

In our real physical world there exist not only rigid bodies but also soft bodies, such as human and animal's soft parts and tissue, and other non-living soft objects, such as cloth, gel, liquid, and gas.

Soft body simulation, which is also known as deformable object simulation, is a vast research topic and has a long history in computer graphics. It has been used increasingly nowadays to improve the quality and efficiency in the new generation of computer graphics for character animation, computer games, and surgical training. So far, various elastically deformable models have been developed and used for this purpose.

In this chapter, we will introduce the concepts about deformable as well as elastic objects. Moreover, we will explain how important this research is and its present applications.

1.1 Definitions

1.1.1 Deformable Object

In engineering mechanics, "deformable object" refers to an object whose shape can be changed due to an applied force, such as tensile (pulling), compressive (pushing), bending, or tearing forces. The deformation can be categorized as the following, depending on the types of material and the forces applied:

- Elastic deformation (small deformation) is reversible. The object shape is temporarily deformed when tension is applied and it returns to its original shape

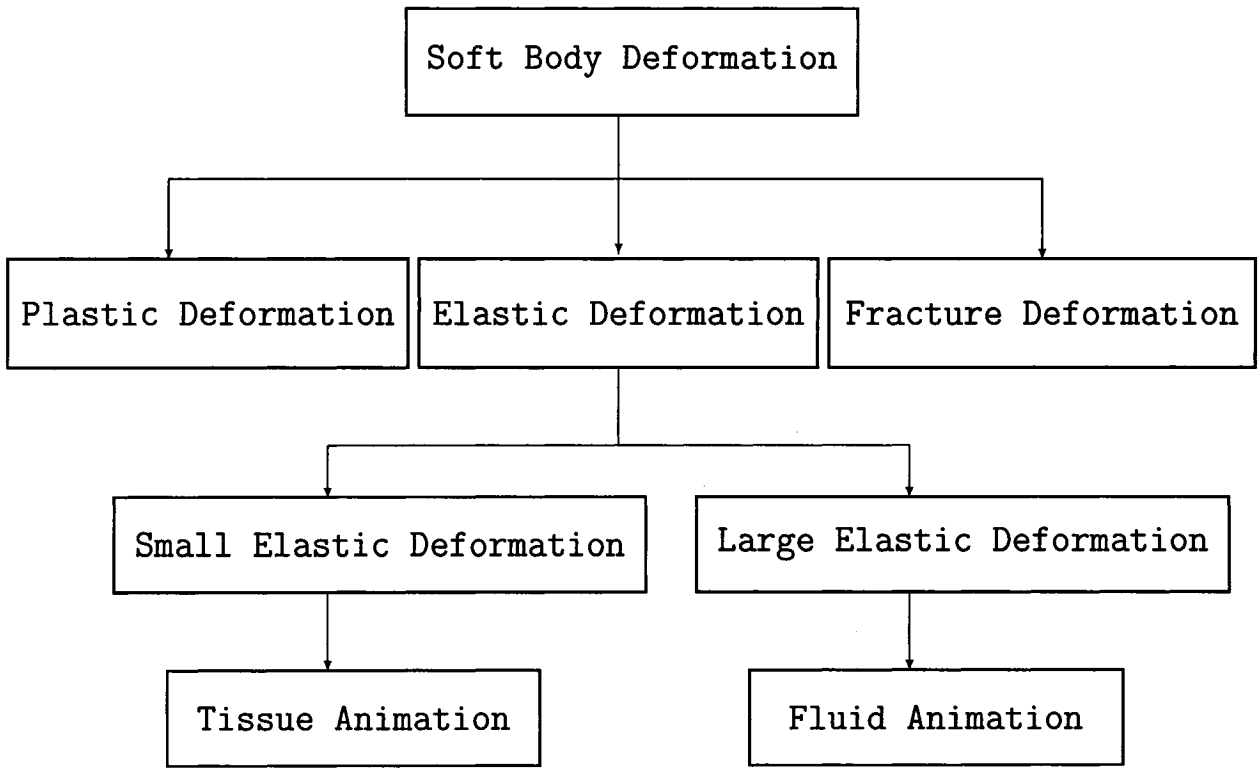


Figure 1: Soft Body Deformation

when force is removed. An object made of rubber has a large elastic deformation range; silk cloth material has a moderate elastic deformation range; crystal has almost no elastic deformation range.

- Plastic deformation (moderate deformation) is not reversible. The object shape is deformed when tension is applied and its shape is partially returned to its original form when the force is removed. Objects such as silver and gum, which can be stretched at their original length, cannot completely restore their original shapes after deformation.

- Fracture deformation (large deformation) is not reversible, but is different from the plastic deformation. The object is permanently deformed when it is irreversibly bent, torn, or broken apart after the material has reached the end of the elastic deformation ranges. All materials will experience fracture deformation when sufficient force is applied.

1.1.2 Elastic Object

Elastic objects belong to a subset of soft body deformable objects. They are dynamic objects that change shape significantly and keep constant volume in response to collision. They can be bent, stretched, and squeezed. Moreover, they restore their previous shape after deformation. Elastic objects can be divided into two domains:

- Large elastic deformation, such as fluid deformation, which focuses on flows through space. It tracks velocity and material properties at fixed points in space.
- Small elastic deformation, such as tissue deformation, which uses particle systems to identify chunks of matter and track their position, acceleration, and velocity.

Within this wide research range of soft body simulation, this work has focused on small elastic deformable object simulation, such as tissue animation. Even though there has been many valuable contribution related to this field, there are still many difficulties in accomplishing to realistic and efficient deformable simulation.

1.2 Animation Techniques

This section introduces some basic concepts related to the elastic simulation, such as the subject animation method. Animation relies on persistence of vision and refers to a series motion illusions resulting from the display of static images in rapid-shown succession. In traditional animation, squash and stretch are exaggerated for elastic objects. In order to be efficient when working with many of single frame images (or simply frames), inbetweening and cel animation [TJ84] have been introduced by Disney for manual traditional animation.

The rate of the animation refers to how many frames are displayed within a given amount of time. If the rate is too low, which is lower than the brain visual retention, the animation becomes jerky because the brain retains the empty frame from the previous image to the next image.

A frame rate, which is the time between two updates of the display, describes the update frequency. In computer games, frames are often discussed in terms of frames

per second (fps). The lower bound for smooth animation is between 22 to 30 frames per second.

For many years' research, computer-animation has been developed dramatically to replace the amount of manual traditional animation. The techniques of key-framing, morphing, and motion capture [HO99] have been widely used.

- Key-frame animation: is based on manual animation. It is a sequence of images of the same object with its local transformations, e.g. values for translation, rotation and scale, computed by interpolating between keyframes.
- Morphing: is a method usually used to estimate and generate a sequence of frames between one object to the other object with same number of vertices. Morphing is a good animation technique when using skeletal animation would be too complex, e.g. facial animation.
- Motion capture: is the method that attaches sensors on actors bodies and records the data for their movements and apply these data to a computer generated characters.

1.3 Elastic Animation

There are two different methods about elastic animation modeling, which depends on the predefined simulation or simulation in real time.

Kinematic modeling predefines the positions and velocities of objects. It does not concern what causes movement and how things get where they are in the first place and only deals with the actual movement. For example, given that a ball's initial speed is 10 kilometers per hour on a perfect smooth plane, we can use kinematic method to calculate how far it travels in two hours.

Dynamic modeling also termed as physically based modeling, is the study of masses and forces that cause the kinematic quantities, such as acceleration, velocity, and position, to change as time progresses. For example, when we know the ball's initial speed, we need to know how far it travels after an external force dynamically applied to it.

For elastic object movement, the dynamic methods calculate how the soft body behaves after external force applied dynamically. The animator does not need to specify the exact path of an object compared to using the kinematic modeling method. The system predefines the initial condition of the elastic object, such as position and gravity force. The animation of the object movement is updated each time step based on the acceleration derived from Newton's Laws of motion. The dynamic simulation method is more advanced, easier to achieve the realistic motion than kinematic method. Therefore, we will only represent dynamic simulation of elastic object in this thesis.

1.4 Applications

Elastic modeling has been developed and used in several fields, including geometric modeling, computer vision, computational mathematics, physics engines, biomechanics, engineering, character animation, and many other fields [GM97].

Character Animation There is much advanced animation modeling software, which has advanced features for modeling, texturing, and lighting. However, for modeling the simulation of elastic objects, 3D artists have to do it manually, frame-by-frame because most of the current 3D software does not provide soft object simulation functionality. Artists have to use not only their drawing skills and intuition, but also posses some knowledge of physics to make the objects behave as if they are in the real world or close.

The techniques of the non-physically based modeling of the elastic object include modeling the group of control points and changing their property parameters manually frame-by-frame. The virtual objects will not convince audiences because no natural laws of physics are applied. Moreover, key-frame animation is an inefficient way to model elastic objects without functionality provided by software. Hence, most of 3D film animators have to ignore the movement details of soft objects.

The latest version of the most advanced animation tool, Maya, provides the Soft Mod Deformer tool, which allows smooth sculpting of a group of objects [Wag07]. However, users need to have knowledge about how to use this complex software in order to access this advanced functionality. Moreover, users can only animate

elastic object with Kinematic modeling method by setting values through the software interface rather than interact with the object in real time.

Computer Games Compared to the fancy and lifelike character animation widely used in 3D films, computer games are more concerned about computation efficiency because users interact with the software in real time. As one might notice, the majority of computer games do not portray the characters in detail, even with the mesh and texture modeling. It is not likely that elastic simulation will be widely introduced to computer games because existing elastic models usually require expensive calculation and are inconvenient to use in real time simulation.

Surgical Training Surgeons benefit from the rapid development of computer graphics and hardware techniques. The computer generated visual virtual environment imitates the reality of medical operations and organ construction to fulfill the training purpose. This new application improves surgical outcomes and decreases the research costs. However, the reality and accuracy of the software always require high-end knowledge of physics, mathematic and heavy computation. It makes it difficult for users to interact with virtual objects in real time.

1.5 Thesis Goal

The elastic object for dynamic simulation, which has been widely used, is the one layer elastic surface with different content within. The soft objects can be squashed and stretched according to external and internal forces applied on them. Its computation depends on geometric modeling methods and physical equations. However, this method is inefficient to imitate the behaviors of real human's tissue because human's or animal's soft body does not consist of only one layer with either liquid or air inside. Figure 2 from a biological research group demonstrates the complexity of human tissue [McE05]. A tissue is composed of epidermis, dermis, fat, fascia, and muscle layer.

- The epidermis is skin's outermost layer. It is responsible for the skin coloring because it contains the skin's pigment.

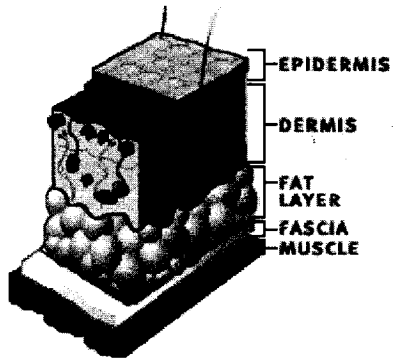


Figure 2: Human Tissue Layers

- The dermis, which is the layer that lies below the epidermis, consists entirely of living cells. It provides the skin elasticity because this layer is composed of bundles of fibers and small blood vessels.

- The fat, the fascia, and the muscle layer are hypodermis layer of skin. It is an inner layer of and cushion for the skin. This subcutaneous tissue layer varies throughout the body region and hormonal influence. The fat and muscle increase the tension of the skin and protect the bones.

Soft tissues are multi-composite layers; therefore, one layer elastic object is not accurate to model the kind of soft body exemplified by human tissue. Moreover, it is difficult to represent the object's inertia caused by the internal material realistically and its liquidity motion based on the various material densities.

In this thesis, we investigate a more accurate two-layer elastic object. The outer layer of the elastic object represents the epidermis and the dermis layer of a real tissue. The inner layer of this object corresponds to the hypodermis layers of skin. This two-layer computer generated elastic object provides more accurate modeling method based on the main feature of human tissue. Its deformation is based on the realistic physical consistency of tissues and the laws of established physics. The objective of this new model is to be convincing and to have distinct realism to the animated scene by applying proper physics. The program should be easy in implementation, convenient to re-use, and gives best elastic body behavior at the minimum cost rather than model the absolute complex object with the exact accurate physical equations. Users should be able to interact with the soft body in real-time and the collision

detection and response must be handled correctly.

1.6 Organization

This chapter starts with the introduction of elastic objects, their applications, some basic concepts related to physical based deformable simulation, and the thesis goal. Chapter 2 surveys and analyzes the existing elastic simulation system and its problems. Chapter 3 describes the modeling methodology of elastic objects in one-dimension, two-dimension, and three-dimension. Physically-based modeling and simulation map a natural phenomena to a computer simulation program. There are two basic processes in this mapping: mathematical modeling and numerical solution [Lin06]. Chapter 4 introduces mathematical modeling, which describes natural phenomena by mathematical equations. Chapter 5 presents the dynamics numerical equation of motion by using ODE (ordinary differential equation) associated with our elastic simulation system, and discusses the complexity and improvement of the different numerical time integrator of Euler, Midpoint, and Runge Kutta 4th order. Chapter 6 presents the detailed design and implementation of the simulation system. Chapter 7 shows our experimental results with the animation sequences of the elastic object simulation and discusses the effects of changing the simulation parameters. Chapter 8 gives the conclusion of our current system, summarizes our contributions based on the existing elastic simulation models, and analyzes the possible development and related work in the future.

Chapter 2

Related Work and Background Material

Research about modeling deformable objects in computer graphics field has been going on for over 40 years and a wide variety techniques have been developed. In this chapter, we will review the existing geometric approaches for modeling elastic objects. These models are all based on physical laws. From the early elastic model, such as particle model, mass-spring model, finite element model, to recent development such as fluid based model, and pressure model, we briefly introduce their physically-based modeling methods and compare these approaches with their advantages and disadvantages.

2.1 Existing Elastic Object Models

Particle Model has been used by Reeves [Ree83] and to model the natural phenomena such as fire, water, liquid as shown in Figure 3. Particles move under the force field and constraint without interacting with each other.

The advantage of this particle model is that the method is easy to implement. It is the earliest model to imitate the natural phenomena.

The disadvantage is that all the particles are independent and there are no forces connecting the particles. Therefore, for some phenomena, such as springs and mass, the method cannot achieve.

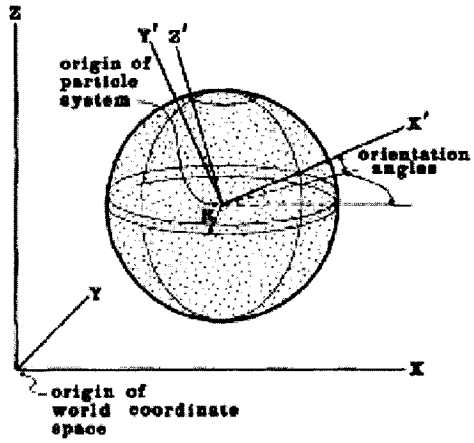


Figure 3: Particle System

Deformable Surface was introduced first time by Terzopoulos *et al.* [TPBF87], using finite difference for the integration of energy-based Lagrange equations based on Hooke's law.

It was very successful in creating and animating surfaces. However, this method requires not only the discretization of the surfaces into spline patches, but also the specification of local connectivity for spring mass systems. These involve a significant amount of manual preprocessing before the surface model can be used.

Linear Mass Spring System has been widely used for modeling elastic objects as shown in Figure 4. It is actually derived from the particle model; however, it simplifies the modeling of the inter-particle connection by using flexible springs. Three dimensional systems contain a finite set of masses connected by springs, which are assumed to obey Hooke's Law.

This method was first introduced by Terzopoulos [TJF89] to describe melting effects. Particles, which are connected by springs, have an associated temperature as one of their properties. The stiffness of the spring is dependent on the temperature of the linked two particles. Increased temperature decreases the spring stiffness. When the temperature reaches the melting point, the stiffness becomes zero.

The advantages of mass-spring model are that it is easy to construct and display the simulation dynamically.

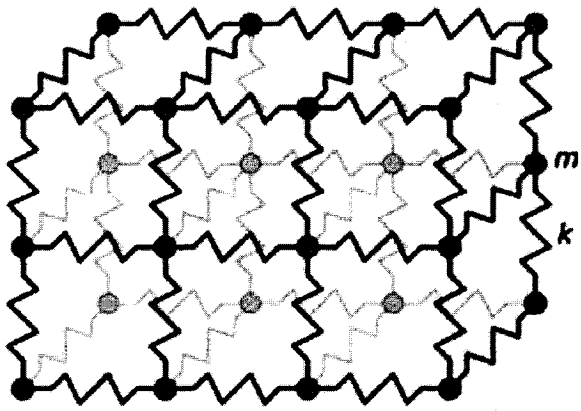


Figure 4: Mass-spring Model

The disadvantages are that such system restricts to only the objects with small elastic deformation with approximation of the true physics, not for the objects that require large elastic deformation, such as fluid. This method also has difficulties to simulate the separation and fusion of a constant volume object. Moreover, the spring stiffness is problematic. If the spring is too weak, for the closed shape model with only simple springs to model the surface will be very easy to collapse. If we try to avoid the collapse, we need to model with spring stiffer, and then we will have difficulty to choose the elasticity because the object is nearly rigid. Another disadvantage is that the mass spring system has less stability and requires the numerical integrator to take small time steps [DW92] than FEM model.

Finite Element Method known as FEM Model [GM97], is the most accurate physical model compared to other models. It treats deformable object as a continuum, which means the solid bodies with mass and energy distributed all over the object. This continuum model is derived from equations of continuum mechanics. The whole model can be considered as the equilibrium of a general object subjected to external forces. The deformation of the elastic object is a function of these forces and the material property. The object will stop deformation and reach the equilibrium state when the potential energy is minimized. The applied forces must be converted to the associated force vectors and the mass and stiffness are computed by numerically integrating over the object at each time step, so the re-evaluation of the

object deformation is necessary and requires heavy pre-processing time [GM97].

The advantage of FEM model is that it gives more realistic deformation result than mass-spring system because the physics are more accurate.

The disadvantage is that the system lacks efficiency. Because the energy equation will be used, the FEM is only efficient for the small deformation of the elastic object, such as application to the plastic material, which has a small deformation range. Alternatively, the object has less control elements needed to be computed, as in cloth deformation. If we need to simulate the human soft body parts or facial animation, the deformation rate is very high. It will be very difficult and sometimes impossible to carry out the integration procedure over the entire body. Therefore, it has been limited to apply in real-time system because of the heavy computational effort (usually it is done off-line). Moreover, the implementation is complicated.

Fluid Based Model [DL02] consists of two components: an elastic surface and a compressible fluid as shown in Figure 5. The surface is represented as a mass spring system. The fluid is modeled using finite difference approximations to the Navier-Stokes equations of fluid flow. Figure 5 describes how this model simulates the fluid flows down a sink simulated. The inner layer is modeled by a particle system, which is similar to real water molecules. Using the numerical methods, the motion of each particle can be computed. In this example, the motion of the each particle is at the center of the basin, and points down to the sink.

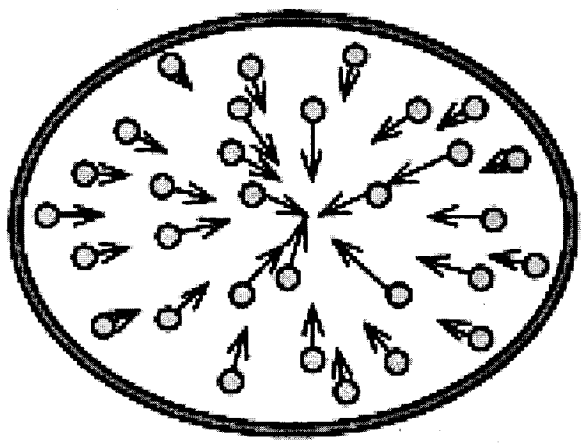


Figure 5: Fluid-Based Soft-Object Model

The fluid based model uses physically based modeling and it produces realistic fluid animation. It illustrates the behavior of fluid in environments with gravity and collisions with planes.

The disadvantage of this model is the heavy computation for both elastic surface and density inside fluid. It also provides a solution to the constant volume problem.

Pressure Model was introduced by M. Matyka [Mat03, Mat04b, Mat04b]. It simulates an elastic deformable object with a internal pressure based on the ideal gas law as shown in Figure 6. The object volume is calculated approximately by bounding box, shaped as sphere, cube, or ellipsoid. Another method to determine the object volume is based on Gauss's Theorem.

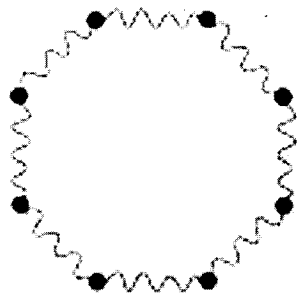


Figure 6: Pressure Soft Body Model

Advantage of this model is that it gives very convincing effects about elastic property in real time simulation. The object behaves like a balloon filled only with air.

However, it can not imitate more interesting effects, such as human tissue. It can not achieve the effect of semi-liquid deformable object because the air pressure density is uniform inside of the object, which is different from liquid with non-uniform density. It is not accurate for describing the inertia of the semi-liquid object.

2.2 Summary of The Existing Models

Previous work on deformable object animation uses physically-based methods with local and global deformations applied directly to the geometric models. Based on the survey of the existing elastic models, we conclude their usage as the two types:

- Interactive models are used when speed and low latency are most important and physical accuracy is secondary. Typical examples include mass-spring models and spline surfaces used as deformable models. These can satisfy the character animation with exaggerated unrealistic deformation.

- Accurate models are chosen when physical accuracy is important in order to accomplish the surgical training purpose which requires the accurate tissue modeling. The continuum simulation model, for instance, the most accurate model, FEM, is not ideal for simulation requiring real time interaction and the object undergoing large deformation.

In short, elastic object simulation is a dilemma of demanding accuracy and interactivity.

Chapter 3

Procedural Modeling Methodology

3.1 Graphics Objects Modeling Methods

Polygonal Methods create geometric objects that can be described by their convex planar polygonal surfaces. These methods are easy to describe the shapes of mathematical objects rendered on graphics system. However, they are difficult to describe physical objects, such as cloud and fire, and their complex behaviors combined with physical laws [Ang03].

Procedural Methods build natural phenomena, 3D models and textures in an algorithmic manner and generate polygons only during the rendering process. The details of the object modeling can be controlled upon vary requests. Meanwhile, these methods are easy to combine computer graphics with physical laws [Ang03, Wik07].

3.2 Procedural Methods

We use procedural modeling methods in our elastic object simulation system. One of the possible approaches to procedural modeling, a particle system, is designed to model elastic objects as described in this section. This particle system is also capable of describing the complex behaviors of elastic objects based on physical laws, such as solving differential equations of thousands of particles on those elastic objects. Another approach is language-based procedural method [Ang03], which generates complex objects with simple programs.

In order to model an elastic object, we need to study the following basic data structures, which are varied in one-dimensional, two-dimensional, and three-dimensional modeling methods.

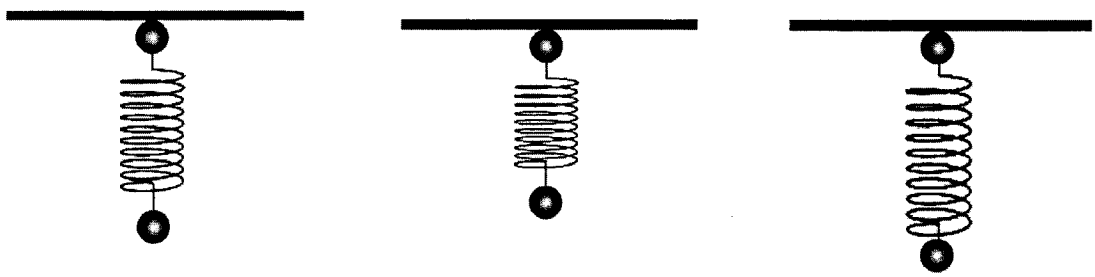
Particles are objects that have mass, position, velocity, and forces applied on them, but have no spatial extent. Moreover, the differential position and velocity change are two properties for these computation of each particle.

Springs are massless with natural length not equal to zero. They are the linkage of particles. There must be at least one spring connects with two particles paired by modeling algorithm.

Faces are the data type that consists of springs as the edges and particles as the vertices.

3.3 1D

The techniques used in an one-dimensional object are presented here, which are applied subsequently to models in two and three dimensions. An one-dimensional object with one end fixed as shown in Figure 7(a). The other end is interacted by users with mouse as in Figure 7(b). The interacted force is restricted to one dimension.



(a) The Initial Spring System is at Rest

(b) The Compressed Force is Applied

(c) The Stretched Force is Applied

Figure 7: One Dimensional Elastic Object

3.3.1 Geometric Data Type

Particle There are two mass particles P_0 and P_1 on a single spring SP_1.

Spring In one-dimensional object, only one type of the spring, structural spring, is introduced. Structural spring, is used to model the object shape, connected by the two mass particles in this case. In Figure 7(a), the spring is at the initial state of equilibrium. The natural length of the spring is l and the force f is zero. In Figure 7(b), the spring is compressed by an external mouse force. The current spring length $l' < l$ and the spring force restores the elastic object to its equilibrium position $f > 0$. When the spring is stretched out as shown in Figure 7(c), the current spring length $l' > l$ and the force of the spring $f < 0$.

3.3.2 Modeling Algorithm

- Step1: Create two particles P_0 and P_1 with positions (x_0, y_0) and (x_1, y_1) shown in Figure 8.
- Step2: Create a spring S_1 with these two particles as two ends Sp_1 and Sp_2.

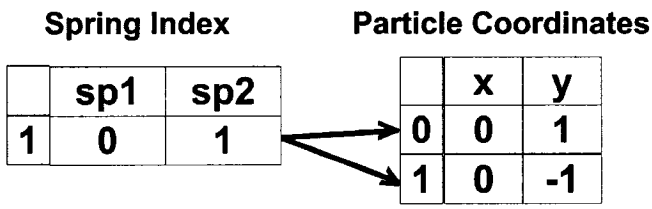


Figure 8: Data Structure for One-dimensional Object Representation

3.4 2D

In this section, we extend the one-dimensional elastic object to two dimensions. We create two separated elastic circles, inner circle and outer circle. Both of them consist of the same modeling structure as one-dimensional mass particles and springs. Then, the two concentric circles, inner and outer, are connected by various springs to become

one two-layered elastic object. However, the distinct features in two-dimensional object have more types of springs presented and the air pressure inside the two-layer close shape is calculated. The spring surface prevents infinite expansion of the air; meanwhile the internal pressure avoids the surface collapsing.

3.4.1 Geometric Data Type

Two-dimensional object is made of three types of primitives, mass particles, springs, and indexed triangular faces.

Particles are defined based on their coordinates related to x and y axes. Consider a unit circle with twelve particles as an example shown in Figure 9. The spatial position for each particle P_i is (x_i, y_i) can be defined by the two equations:

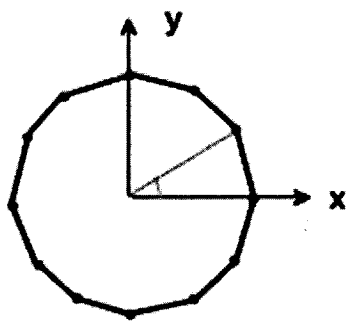
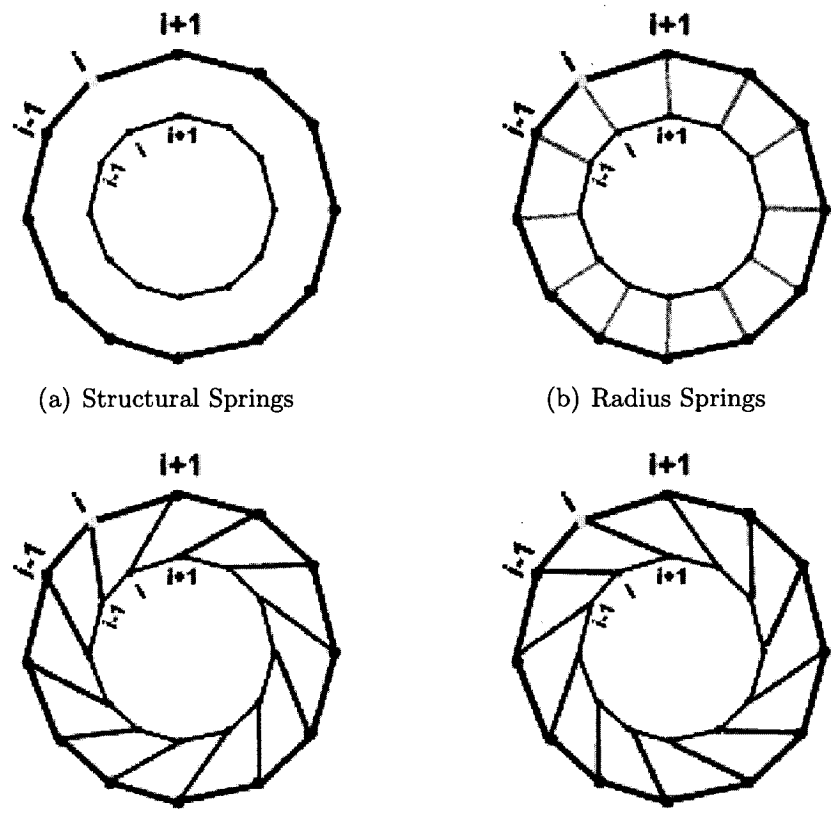


Figure 9: Two-dimensional Elastic Object with Single Layer

$x(\theta) = \cos(\theta + \Delta\theta)$
$y(\theta) = \sin(\theta + \Delta\theta)$
where
$0^0 \leq \theta \leq 360^0$
$\Delta\theta$ is a small angle stepping along θ

Springs In additional to the structural spring, which also exists in one-dimensional object, there are two other types of springs, radius spring and shear spring.

- ◇ Structural springs: give the basic structure of inner circle and outer circle to prevent neighboring particles from getting too close to one another as shown in



(a) Structural Springs (b) Radius Springs

(c) Shear Left Springs (d) Shear Right Springs

Figure 10: Four Types of Springs on a Two-dimensional Object

Figure 10(a). Linkage of each structural spring i is to connect with two particlesP_i and P_{i+1} or P_{i-1} and P_i.

◇ Radius springs: are the springs connected from particles on inner circle to the particle on the outer circle as part of the circle radius in order to prevent the bending of the surface as shown in Figure 10(b). Linkage of each radius spring i is to connect with particle P_i^{inner} and the particle P_i^{outer}.

◇ Shear springs: are springs connected from particles on inner circle to their diagonal neighbors on outer circle in order to avoid the surface fold over . Linkage of each left shear spring i is to connect with particle P_i^{outer} diagonally and the particle P_{i-1}^{inner}; connect with particle P_{i+1}^{outer} diagonally and the particle P_i^{inner} and so on as shown in Figure 10(c). Linkage of each right shear spring i is to connect with particle P_{i-1}^{outer} diagonally and the particle P_i^{inner}; connect with particle

P_i^{outer} diagonally and the particle P_{i+1}^{inner} and so on as shown in Figure 10(d).

Faces are the data structure for the only purpose of drawing and displaying a filled object to a two-dimensional object. The triangular facets can be drawn separately and visualized as a filled circle as shown in Figure 11.

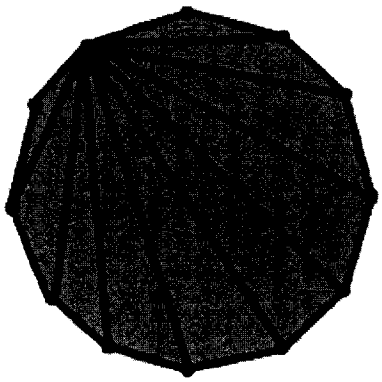


Figure 11: Two-dimensional Elastic Object Facets

3.4.2 Modeling Algorithm

- Step 1: Define the number of particles as $n = 12$ in our example. Then, the step size is $\Delta\theta = \frac{360^0}{12} = 30$ degrees.

- Step 2: Define the group of particles' position on inner circle and the ones on outer circle as shown in Figure 12. The first particle P_0 is at $(\cos\theta, \sin\theta)$ where $\theta = 0^0$, the second particle is at $(\cos\theta, \sin\theta)$ where $\theta = \theta + \Delta\theta = 30^0$... By multiplying the inner and outer coordinates with a different radius value, for example, $Radius_{inner} = 1.5$, and $Radius_{outer} = 2$ to create two concentric circles.

- Step 3: Add the structural springs S_0, S_1, ...S_{11} to the inner circle according to the spring index of inner particles as shown in Figure 12. The same method is applied to outer structural springs on outer circle. The last spring, S_{11} in our example, is composed of two particles P_{11} and P_0 as two ends in order to make a close shape.

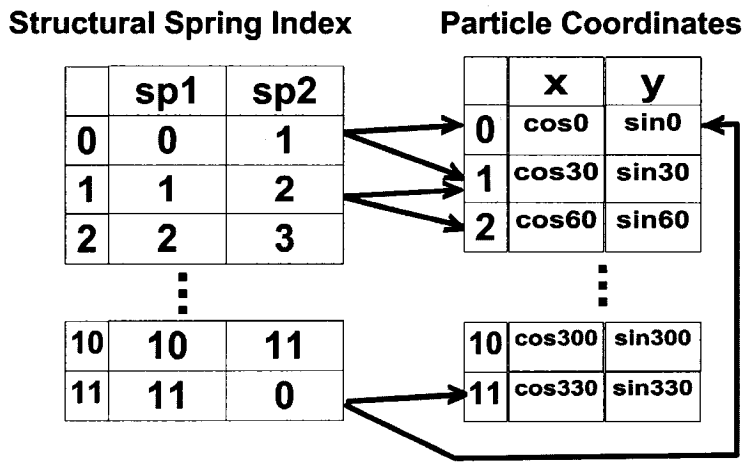


Figure 12: Data Structure for Two-dimensional Object Representation 1

- Step 4: Loop through the number of structural springs $n = 12$ to add the same number of radius springs according to the linkage of the inner and outer particles as shown in Figure 13.

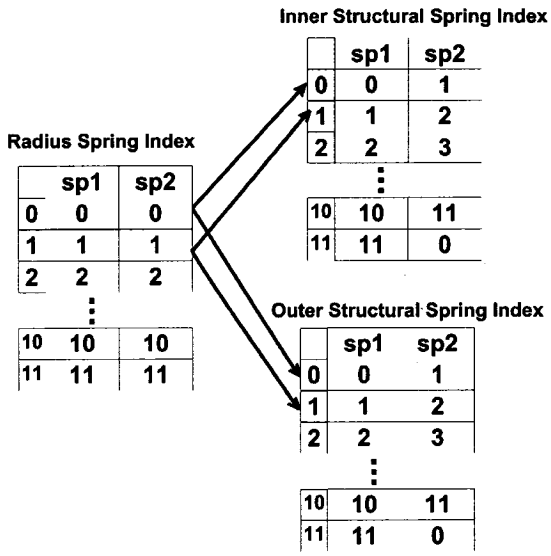


Figure 13: Data Structure for Two-dimensional Object Representation 2

- Step 5: Loop through the number of structural springs to add the same number of shear left springs and shear right springs according to the linkage of the inner

and outer particles as shown in Figure 14.

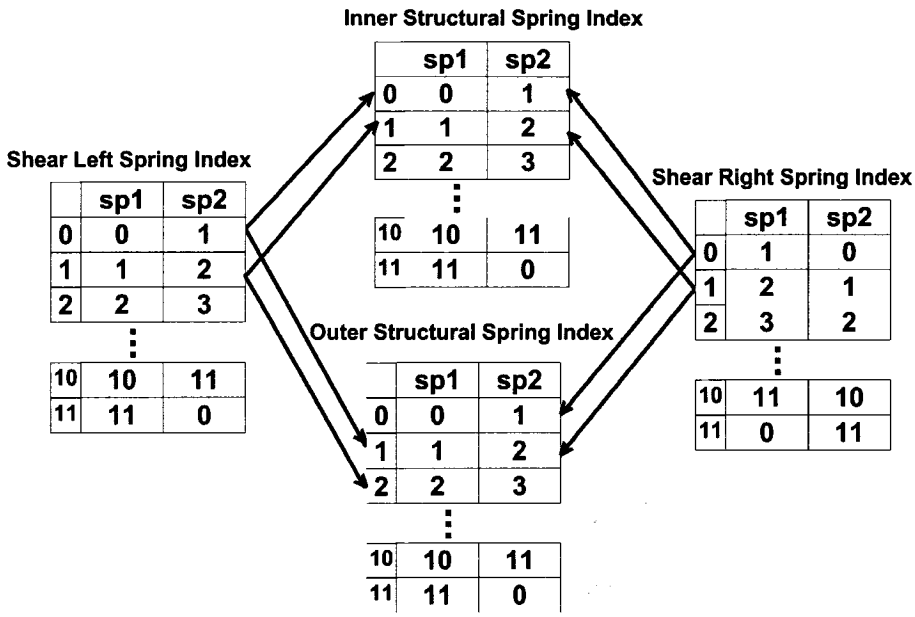


Figure 14: Data Structure for Two-dimensional Object Representation 3

3.5 3D

In this section, a more complicated three-dimensional elastic object is extended from the two-dimensional object. In the two-dimensional model, the structural springs' index is the most important key data structure to link up all the particles and reference about the index of the particles. This spring linkage method will still work for the model based on the non-uniform sphere geometric modeling method. However, in the other geometric modeling method, the uniform sphere modeling, the faces' index is the key data structure of the linkage to other data structure, such as particles and springs. The reason is because in the later geometric modeling method, each facet on the object is used for subdivision of other facets in each iteration. Compared with a two-dimensional object, the three-dimensional object consists the same types of primitives, such as particles, springs, and faces, but extended to z axis.

3.5.1 Non-Uniform Sphere

3.5.1.1 Geometric Data Type

One of the simplest non-uniform modeling methods to generate an approximate facet sphere uses Polar to Cartesian Coordinates method. Consider θ the angle on xy-plane (around z-axis), known as the Azimuthal Coordinate. The angle ϕ is from z-axis, known as the Polar Coordinate. If we fix θ and draw curves as we change ϕ, we get circles of constant longitude; if we fix ϕ and vary θ, we obtain circles of constant latitude [Ang03].

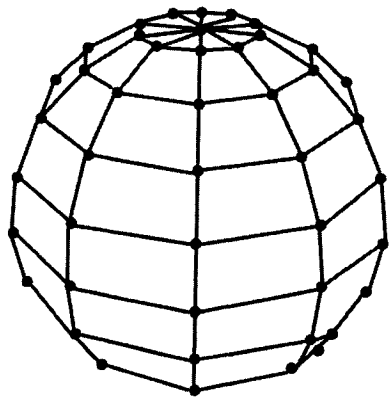


Figure 15: Polar Cartesian Coordinates Non-uniform Sphere Generation

Particles The spherical coordinates for a particle i can be defined by the three equations:

$x(\theta, \phi) = \cos\theta \sin\phi$

$y(\theta, \phi) = \cos\theta \cos\phi$

$z(\theta, \phi) = \sin\phi$

where

$0^0 <= \theta <= 360^0$

$-90^0 <= \phi <= 90^0$

By stepping θ and ϕ in small angles $\Delta\theta$ and $\Delta\phi$ between their bounds as the number of slices and stacks, the particles are:

$P_0(x_0, y_0, z_0)= (\sin\theta \cos\phi, \cos\theta \cos\phi, \sin\phi)$

$P_1(x_1, y_1, z_1)= (\sin\theta \cos(\phi + \Delta\phi), \cos\theta \cos(\phi + \Delta\phi), \sin(\phi + \Delta\phi))$

$P_2(x_2, y_2, z_2)$= $(\sin(\theta + \Delta\theta)\cos\phi, \cos(\theta + \Delta\theta)\cos\phi, \sin\phi)$
$P_3(x_3, y_3, z_3)$= $(\sin(\theta + \Delta\theta)\cos(\phi + \Delta\phi), \cos(\theta + \Delta\theta)\cos(\phi + \Delta\phi), \sin(\phi + \Delta\phi))$
...

However, at the North and South Pole areas, we can only use triangles to present because all lines of latitude are converged.
The particle at the North Pole area can be presented as:
$P(x, y, z)$= $(\sin(\theta + \Delta\theta)\cos 90^0, \cos(\theta + \Delta\theta)\cos 90^0, \sin 90^0)$
The particle at the South Pole area can be presented as:
$P(x, y, z)$= $(\sin(\theta + \Delta\theta)\cos 90^0, \cos(\theta + \Delta\theta)\cos 90^0, -\sin 90^0)$

Springs There are also three types of springs in three-dimensional objects as we described in two-dimensional objects, such as structural, radius, and shear springs.

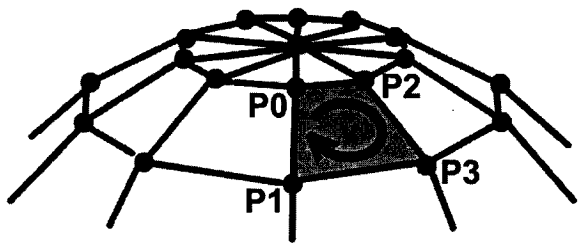


Figure 16: Quadrilaterals and Triangles on Non-uniform Sphere

- Structural spring is still the basic data structure to form the shapes of inner and outer spheres. Four particles define four springs as the proper order. Taking the first four particles P_0, P_1, P_2, and P_3 as an example, the first four springs's are $S_0 = P_0P_2$, $S_1 = P_2P_3$, $S_2 = P_3P_1$, $S_3 = P_1P_0$ shown in Figure 16. The structural springs on two poles are also defined by the particles on poles as proper order.

- Radius and shear springs, which connect inner and outer layers, follow the same methods as in two-dimensional object.

Faces Any quadrilateral-facet on the body of sphere can be represented by four springs: S_i, S_{i+1}, S_{i+2}, and S_{i+3}. Any triangular-facet on the poles can be represented by three springs: S_j, S_{j+1}, and S_{j+2}.

3.5.1.2 Modeling Algorithm

- Step 1: Define the number of slices and stacks of a sphere, $n_{slice} = 10$ and $n_{stack} = 10$ in our example. Then, the step size is $\Delta\theta = \frac{360^0}{10} = 36^0$ and $\Delta\phi = \frac{180^0}{10} = 18^0$.

- Step 2: Define the group of particles' position on inner circle and the ones on outer circle shown in Figure 17. By multiplying the inner and outer coordinates with a different radius value, for example, $Radius_{inner} = 1.5$, and $Radius_{outer} = 2$ to create two concentric spheres.

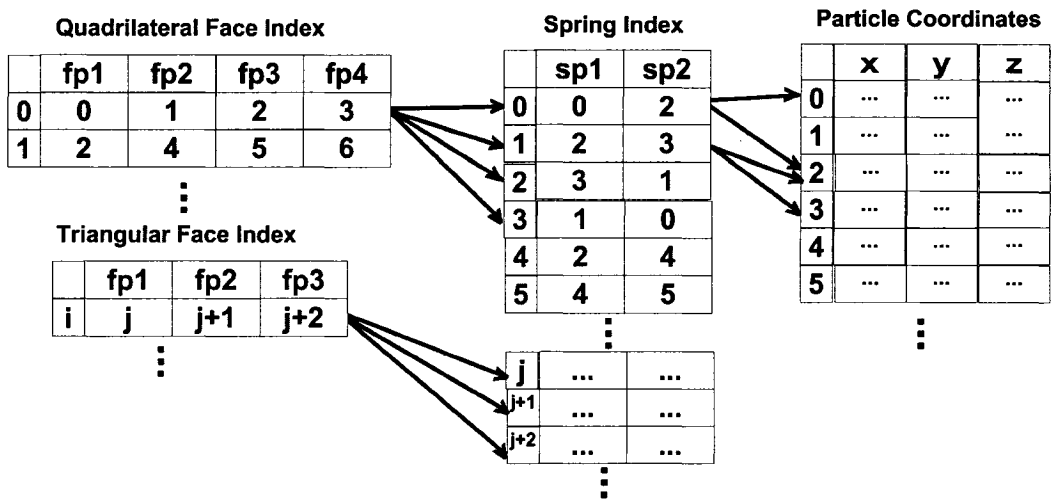


Figure 17: Data Structure for Three-dimensional Non-uniform Object Representation

- Step 3: Add the structural springs S_0, S_1, ... to the inner circle according to the spring index of inner particles as shown in Figure 17. The same method is applied to outer structural springs on outer circle.

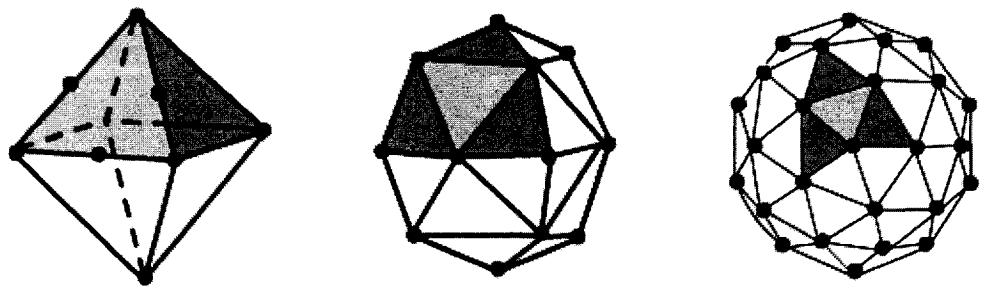
- Step 4: Loop through the number of structural springs to add the same number of radius springs and shear springs according to the linkage of the inner and outer particles as described in two-dimensional object modeling method, shown in Figure 13 and Figure 14.

3.5.2 Uniform Sphere

An important drawback of the non-uniform sphere model is that the faces vary in both shape (some are triangles and some are quadrilaterals) and size.

3.5.2.1 Geometric Data Type

Surface refinement is a simple way for uniform modeling. It is started with a kernel polyhedron, which is a regular polyhedron with faces that are equilateral triangles. We have used an octahedron with bisecting each face at the same time recursively. This method is a powerful technique for generating approximations to curves and surfaces of a sphere to any desired level of accuracy.



(a) The Initial Octahedron Shape

(b) The Unit Facet Sphere Object With One Iteration

(c) The Unit Facet Sphere Object With Second Iteration

Figure 18: Uniform Sphere Generation

Particles The algorithm starts with a regular octahedron shown in Figure 18(a). The shape is composed of eight equilateral triangles, determined by six vertices, $P_0(0,0,1)$, $P_1(0,0,-1)$, $P_2(-1,-1,0)$, $P_3(1,-1,0)$, $P_4(1,1,0)$, and $P_5(-1,1,0)$. The vertices of the kernel polyhedron are known to lie on the surface of a unit sphere of radius $r = 1$. We fix the two vertices P_0 and P_1 on z axis and normalize the other five vertices by multiplying $\frac{1}{\sqrt{2}}$ in order to make them lie on the unit sphere, centered at the origin. The six vertices after normalization are $P_0(0,0,1)$, $P_1(0,0,-1)$, $P_2(-0.7,-0.7,0)$, $P_3(0.7,-0.7,0)$, $P_4(0.7,0.7,0)$, and $P_5(-0.7,0.7,0)$.

Faces We talk about faces before talking about springs because the face is the key data structure for recursive subdivision and its index is referenced by spring index. The first eight triangular faces defined by the six particles are $f_0 = P_0P_3P_4$, $f_1 = P_0P_4P_5$, $f_2 = P_0P_5P_2$, $f_3 = P_0P_2P_3$, $f_4 = P_1P_4P_3$, $f_5 = P_1P_5P_4$, $f_6 = P_1P_2P_5$, $f_7 = P_1P_3P_2$.

Springs Each face is composed of three springs. Therefore, the first twelve springs on the octahedron are $S_0 = P_0P_3$, $S_1 = P_3P_4$, $S_2 = P_4P_0$, $S_3 = P_4P_5$, $S_4 = P_5P_0$, $S_5 = P_5P_2$, $S_6 = P_2P_0$, $S_7 = P_2P_3$, $S_8 = P_1P_4$, $S_9 = P_1P_3$, $S_{10} = P_1P_5$, and $S_{11} = P_2P_1$.

Subdivision We can subdivide a single triangular face of the kernel polyhedron by projecting the midpoints *pa*, *pb*, *pc* of its three edges onto the surface of the sphere as shown in Figure 19.

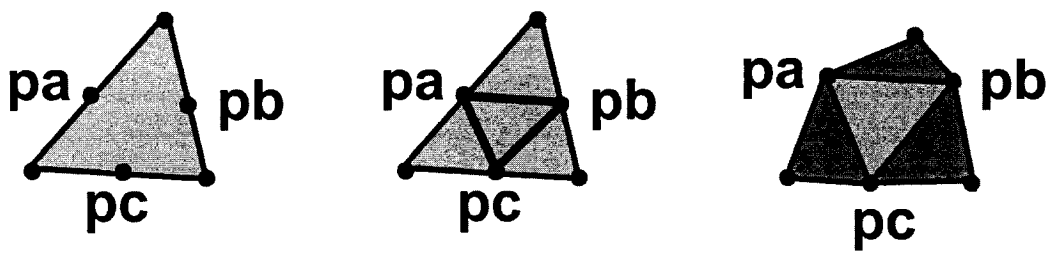
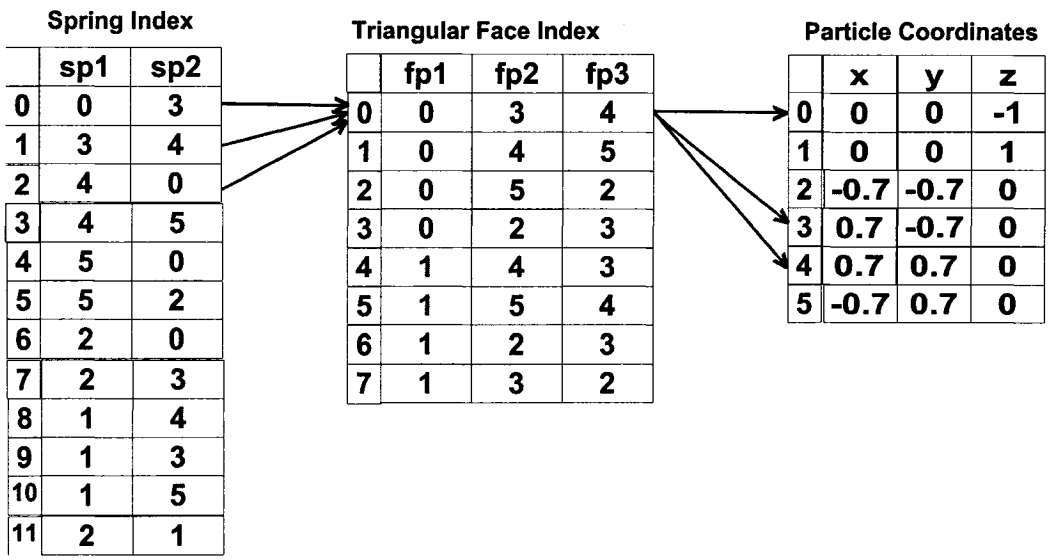


Figure 19: Subdivision of A Triangle By Bisecting Sides

This face is split into four faces by bisecting each edge. The four new triangles are still in the same plane as the original triangle. We move the new vertices *pa*, *pb*, *pc* to the unit sphere by normalizing each new vertices. The number of particles increases by a factor of 2. The number of springs increases by a factor of 3. The number of facets increases by a factor of 4. We subdivide another 7 triangles with the same method. After subdividing the octahedron once, the number of particles are 12, the number of triangular faces is 32, and the number of springs is 36. We can repeat the subdivision process n times to generate successively closer approximations to the sphere.

3.5.2.2 Modeling Algorithm

- Step 1: Define a collection of particles to create a closed equilateral triangles shape of the elastic object. Define an octahedron object as the initial object with 6 particles, 8 triangular faces, and 12 structural springs as shown in Figure 20.



	sp1	sp2
0	0	3
1	3	4
2	4	0
3	4	5
4	5	0
5	5	2
6	2	0
7	2	3
8	1	4
9	1	3
10	1	5
11	2	1

	fp1	fp2	fp3
0	0	3	4
1	0	4	5
2	0	5	2
3	0	2	3
4	1	4	3
5	1	5	4
6	1	2	3
7	1	3	2

	x	y	z
0	0	0	-1
1	0	0	1
2	-0.7	-0.7	0
3	0.7	-0.7	0
4	0.7	0.7	0
5	-0.7	0.7	0

Figure 20: Data Structure for Three-dimensional Uniform Object Representation without Subdivision

- Step 2: Connect the particles with the structural springs according to the edge order of the octahedron to make an inner layer of the three-dimensional object. Each particle is separated equidistantly from its neighbors.

- Step 3: Check if there is need of subdivision to approach a more spherical object.

- Step 4: In the first subdivision, the object becomes 12 particles, 32 triangles, and 36 structural springs. The Figure 21 shows how new data can be inserted into a collection of particles, faces, and springs after subdivision.Use the first triangle as a concrete example. In the initial octahedron shape, find the midpoint of each edge on F_0. Normalize the coordinates of these three new particles to make them lie on the sphere. Push these three new particles to the particle container. The first face of the initial octahedron has three pointers that point to particles P_0, P_3, and P_4 as shown in Figure 20. After subdividing this triangle, it becomes

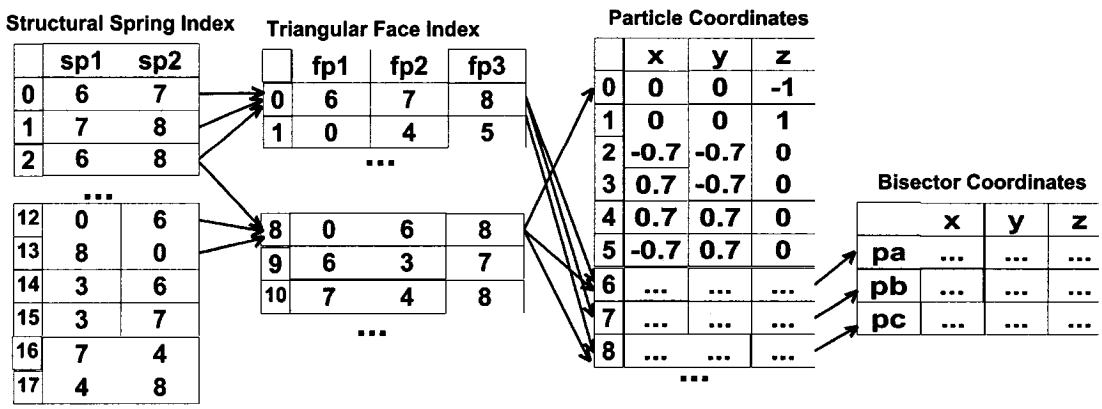


Figure 21: Data Structure for Three-dimensional Uniform Object Representation with the Number of Subdivision n=1

four smaller triangles connected by the bisectors *pa*, *pb*, and *pc*. The three new triangles are pushed onto container *Faces*. The middle triangle replaces the original big triangle because the original triangle does not exist anymore. The pointers on each face point to the correspondent particles as indicated in Figure 21. New structural springs are added to Spring container correspondent to new faces only if there is no such spring has existed yet. The subdivision of the remaining faces follows the same method. More faces are approaching the object to a unit facet sphere.

- Step 5: Repeat Step 1 to 5 to create an outer layer with larger radius value than the inner layer.

- Step 6: Loop through the number of structural springs to add the same number of radius springs according to the linkage of the inner and outer particles.

- Step 7: Loop through the number of structural springs to add the same number of shear left springs and shear right springs according to the linkage of the inner and outer particles.

3.5.3 Comparison of Non-uniform and Uniform Methods

The advantages of both methods are that they can be used to describe complex behaviors combined with physical laws, such as elasticity. Additionally, the level of detail (LoD) of the object can be adjusted depending on the proximity of the object on the display to the human's eye.

The disadvantage of the non-uniform sphere modeling method is that the facets of the sphere do not have approximately equal size. The facets become smaller at the poles and bigger at the "equator". Therefore, the springs are shorter at the poles and longer at the equator. The normal of each spring varies from equator and the ones on the poles. Consequently, this non-uniform modeling method increases errors in force computations for each particle.

The disadvantage of the uniform facet sphere generation algorithm is that it can not generate surfaces of arbitrary resolution. It can be shown that at all levels of recursion, particles at the kernel points are connected to four springs if the kernel object is an octahedron (as shown in Figure 18(b)). In other cases, all the particles at the kernel points are connected to five springs if the kernel object is an icosahedron (20 faces); all the particles at the kernel points are connected to three springs if the kernel object is a tetrahedron. All particles at recursively generated points are connected to six springs. This will result the irregular surface stiffness and might cause the non-spherical shape because the same pressure will displace the regions of a surface about the kernel points further than the rest of the surface.

To solve this problem, the sum of the spring forces accumulated at a particle can be normalized by multiplying a factor of $\frac{6}{n_{springs}}$, where $n_{springs}$ is the number of springs connected to this particle. For example, if $particle_a$ is a kernel point, which is connected to four springs and the sum of the spring forces is f_a; and if $particle_b$, which is the point generated from subdivision, connects to six springs and the sum of the six spring forces is f_b. f_a is multiplied by a factor of $\frac{6}{4}$ and f_b is multiplied by a factor of $\frac{6}{6}$.

Our simulation system ignores the described drawback resulting from the uniform sphere modeling method. We find a set of air pressure and spring stiffness parameter values at which the simulation is stable by trial and error. Thus, the difference of the forces for every particle either connected to four springs or six springs is not addressed in this work.

Chapter 4

Physical Based Modeling Methodology

A one-dimensional object model includes gravity force $\mathbf{F}^g$, user applied force $\mathbf{F}^a$, and collision force $\mathbf{F}^c$ as external forces; linear structural spring force $\mathbf{F}^h$ and spring damping force $\mathbf{F}^d$ as internal forces.

$$\mathbf{F} = \mathbf{F}^g + \mathbf{F}^h + \mathbf{F}^d + \mathbf{F}^a + \mathbf{F}^c \tag{1}$$

A two-dimensional object model is considered as a closed shape with air pressure inside. Then, the air pressure $\mathbf{F}^p$ is a new internal force exist in two-dimensional object in addition to the common forces in one-dimensional object. Accumulation of forces on a three-dimensional object is similar to forces applied on the two-dimensional one. The only difference is that all forces on three-dimensional objects are extended to axis z.

$$\mathbf{F} = \mathbf{F}^g + \mathbf{F}^h + \mathbf{F}^d + \mathbf{F}^a + \mathbf{F}^p + \mathbf{F}^c \tag{2}$$

4.1 Gravity Force

Gravity force is a constant force at which the earth attracts objects based on their masses. In most cases, the particle system does not include gravitation, but, in our system, particle gravities represent object's density. Users can set particle gravities to a non-zero value. g is a constant scalar of the gravitational field.

$$\mathbf{F}^g = mg \tag{3}$$

4.2 Spring Hooke's Force

Spring force is a linear force exerted by a compressed or stretched spring upon two connected particles. The particles which compress or stretch a spring are always acted upon by this spring force which restores them to their equilibrium positions. It is calculated as following according to Hooke's law, which describes the opposing force exerted by a spring.

$$\mathbf{F}_{12}^h = -\left(||\mathbf{r}_2 - \mathbf{r}_1|| - r_l\right) k_s \tag{4}$$

where

$\mathbf{r}_1$ is the first particle position,

$\mathbf{r}_2$ is the second particle position,

r_l is default length of the resting spring between the two particles,

k_s is the stiffness of the spring,

when $||\mathbf{r}_2 - \mathbf{r}_1|| - r_l = 0$, the spring is resting,

when $||\mathbf{r}_2 - \mathbf{r}_1|| - r_l > 0$, the spring is extending,

when $||\mathbf{r}_2 - \mathbf{r}_1|| - r_l < 0$, the spring is contracting.

We have discussed the type of structural spring in one-dimensional object. In two and three dimensional object model, the same method applies on the other three types of spring, such as radius springs, shear left springs, and shear right springs with different spring stiffness and spring damping factor. So, the total Hooke's spring force is:

$$\mathbf{F}_{total}^h = \mathbf{F}_{structure}^h + \mathbf{F}_{radius}^h + \mathbf{F}_{shearleft}^h + \mathbf{F}_{shearright}^h \tag{5}$$

4.3 Spring Damping Force

Spring damping force is also called viscous damping. It is opposite force of the Hook spring force in order to simulate the natural damping and resist the motion. It is

also opposite to the velocity of the moving mass particle and is proportional to the velocity because the spring is not completely elastic and it absorbs some of the energy and tends to decrease the velocity of the mass particle attached to it. It is needed to simulate the natural damping due to the forces of friction. More importantly, it is useful to enhance numerical stability and is required for the model to be physically correct [BA97].

$$\mathbf{F}_{12}^{d} = (\mathbf{v}_2 - \mathbf{v}_1) \cdot \left(\frac{\mathbf{r}_2 - \mathbf{r}_1}{||\mathbf{r}_2 - \mathbf{r}_1||} \right) k_d \tag{6}$$

where

$\left(\frac{\mathbf{r}_2-\mathbf{r}_1}{||\mathbf{r}_2-\mathbf{r}_1||}\right)$ is the direction of the spring,

$\mathbf{v}_1$ and $\mathbf{v}_2$ is the velocity of the two masses,

k_d is spring damping coefficient.

When the two endpoints moving away from each other, the force imparted from the damper will act against that motion; when the two endpoints moving toward each other, the damper will act against the squeeze motion. The damper will always acts against the motion. The total spring damping force is:

$$\mathbf{F}_{total}^{d} = \mathbf{F}_{structure}^{d} + \mathbf{F}_{radius}^{d} + \mathbf{F}_{shearleft}^{d} + \mathbf{F}_{shearright}^{d} \tag{7}$$

4.4 Drag Force

Drag force is the force when users interact with the elastic object through mouse. At the moment users click the mouse, the simulation system finds the nearest particle i to the current position of the mouse. If users drag this particle i, the drag force contributes to force of this nearest particle. The forces applied on rest particles are effected by the new user applied force, which is passed through by springs.

We consider one end of the string connects to the mouse position and the other end of the string connects to the nearest particle on the object. This string is elastic, so it has all the spring's properties, such as hook spring force and damping force. The drag force can be presented as following:

$$\mathbf{F}^{a} = -\left(||\mathbf{r}_m - \mathbf{r}_i|| - r_{lm}\right) k_{sm} + (\mathbf{v}_m - \mathbf{v}_i) \cdot \left(\frac{\mathbf{r}_m - \mathbf{r}_i}{||\mathbf{r}_m - \mathbf{r}_i||} \right) k_{dm} \tag{8}$$

where

$\mathbf{r}_m$ is the mouse position,

$\mathbf{r}_i$ is the particle position nearest to mouse,

$\mathbf{r}_{lm}$ is default length of the resting mouse spring,

k_{sm} is the stiffness of the mouse spring,

$\mathbf{v}_m$ is the velocity of the mouse represented as a mass

$\mathbf{v}_i$ is the mass for the nearest particle,

k_{dm} is spring damping coefficient for the mouse spring.

$\mathbf{F}^a$ is a momentary force for interacting with the elastic simulation system. This force is accumulated to the current forces already applied on this nearest particle.

In a one-dimensional object simulation system, the nearest particle to mouse is either P_0 or P_1. In a two-dimensional and three-dimensional object system, the drag force is only applied on the outer layer of the double layered object when user interacts the object with mouse.

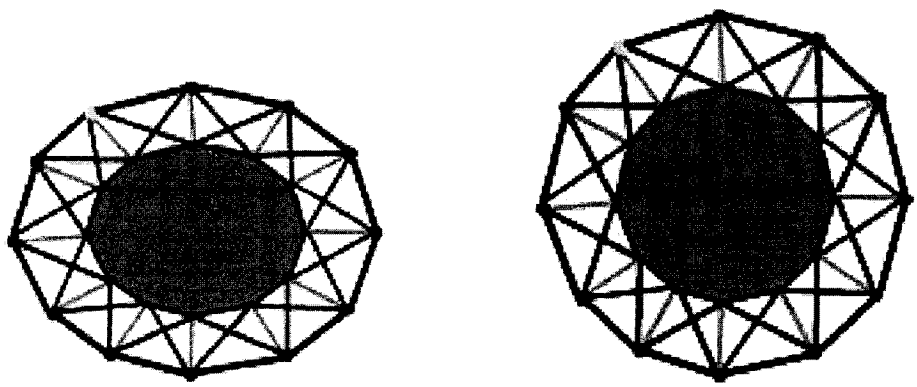
4.5 Air Pressure Force

In order to describe an elastic object more accurately, especially soft body of human beings and animals, the calculation only about the elastic force on the object's surface is not enough. We add the flow pressure force inside of the elastic object to make the object wobbly looking when it is deformed.

The pressure force will be calculated for every spring, then update each particle's direction. The pressure vector is always acting in a direction of normal vectors to the surface, so the shape will not deform completely. If pressure is simulated without also simulating the mass-spring system, the object will explode.

In Figure 22(a), the object is deformed from bottom because of the gravity force. If there is no internal air pressure, the object will collapse unless the springs are hard enough to avoid the failure. With the very hard springs, it is difficult to simulate the reality of the elasticity. The real elastic object restores its shape as described in Figure 22(b). The simplified version of the Ideal Gas Law [Mat03], also known as Clausius Clapeyron Equation, is used to describe such effect:

$$PV = NRT \tag{9}$$



(a) The External Gravity Force Is Applied Producing A Pressure Wave

(b) The Object Restores Its Shape With Internal Air Pressure Force

Figure 22: Double-layered Two-dimensional Elastic Object Filled With Air

where

P is the pressure value,

V is the volume of the object,

N is number of mols,

R is the gas constant,

T is the gas temperature. Therefore, the pressure force is:

$$\mathbf{F}^p = P\,\mathbf{n} \tag{10}$$

where

$\mathbf{F}^p$ is the pressure force vector,

$\mathbf{n}$ is the normal vector to the springs on the object.

$$P = \frac{NRT}{V} \tag{11}$$

4.5.1 Volume

In order to find an estimate pressure inside of the object which will be applied to particles later, we need to calculate the volume of the object. The approximation of the volume is calculated with Gauss' Theorem [Ker07]:

$$V = \int\int\int_v f(x,y,z)\,dx\,dy\,dz \Longleftrightarrow V = \int\int\int_v f(x,y,z)\,dV \tag{12}$$

where triple integrals of a function $f(x,y,z)$ define a volume integral of an elastic sphere. Moreover, triple integrals can be transformed into surface double integrals over the boundary surface of a region if the three-dimensional object is closed shape by divergence theorem [Ker07]:

$$V = \int\int\int_V \Delta\mathbf{F}\,dV \Longleftrightarrow S = \int\int_S \mathbf{F}\,dS \tag{13}$$

where

$\mathbf{F}$ is a vector field,

V is the object volume,

S is the object surface.

Double integrals over a plane region may be transformed into line integrals by Green's Theorem in the Plane:

$$\int\int_S \Delta\mathbf{F}\,dx\,dy \Longleftrightarrow \int_L \mathbf{F}\,dL \tag{14}$$

where L is the object edge and dL is the length of the edge.

Therefore, a triple integrals function $f(x,y,z)$ shown in Eq.12, which defines a volume integral of an elastic sphere, can be transformed to line integrals as shown in Eq.15.

$$V \approx \int_L \mathbf{F}\,dL \tag{15}$$

We assume on the line, the vector field $\mathbf{F} = (x,0)$, the simplified integration of body volume is [Mat03, Ker07]:

$$\int_L \mathbf{F}\,dL = \sum_{i=0}^{i=NUMS-1} \frac{1}{2}(\mathbf{x_1} - \mathbf{x_2})\,\mathbf{n_x}\,dL \tag{16}$$

where

V is the volume of the object,

$(\mathbf{x_1} - \mathbf{x_2})$ is the absolute difference of the line (represents spring here) of the start and end particles at axis x,

$\mathbf{n_x}$ is the normal vector to this line (spring) at axis x,

dL is the line's (spring's) length.

4.5.2 Normals

Normals are unit vectors perpendicular to specified data structure, such as particle (vertex psuedo-normals), spring (line), and face (polygonal facets) on the object.

- Particle normal, or vertex psuedo-normals, does not exist for vertices; however, it can be considered as the average of the normals of the subtended neighbor particles. To calculate the particle normal is to sum up the normals for each face adjoining this particle, and then to normalize the sum.
- Spring (line) normal in two dimension is based on the two particles P_1, P_2 connected on the spring. It is perpendicular to the spring itself.
- Face (plane) normal in three dimension is determined by right-hand rule, which is perpendicular to its surface based on the any pair of springs on the surface. The normal for a triangle surface composed with three particles P_1, P_2, P_3 is computed as the vector cross product of the springs $P_2 - P_1$ and $P_2 - P_3$.

The usage of normal calculation method in our elastic object simulation system is for analysis of the direction of the pressure force inside of the object. Only spring normal is calculated here because all the internal and external forces will apply on each spring, and the spring will define the particles' force which connect onto it.

4.5.2.1 2D Normals

For the single spring $Spring_{12}$, the Cartesian coordinates for particle P_1 is (x_1, y_1); the Cartesian coordinates for particle P_2 is (x_2, y_2). The normal to this spring is the spring rotate 90^0 at axis z according to the space position. So, we can get the components of the normal in x axis and y axis as following

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos 90^0 & -\sin 90^0 \\ \sin 90^0 & \cos 90^0 \end{bmatrix} \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -(y_2 - y_1) \\ x_2 - x_1 \end{bmatrix}$$

4.5.2.2 3D Normals

The calculation of the 3D normals of springs is important because it will define the direction of the internal air pressure force, either compress the elastic object or expand its volume. In theory, in three-dimensional simulation, the normal of a spring in the space position is represented as an average of the normals of faces connected to it. However, in our elastic object simulation system, we use a simplified estimated normal based on the normal algorithm of the two-dimensional calculation discussed above. Instead of rotating a line 90^0 at z axis to get its normal vector in two-dimension, the estimated algorithm is rotating a line 90^0 at z axis, y axis, and x axis to get its normal in three-dimension.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos 90^0 & -\sin 90^0 & 0 & 0 \\ \sin 90^0 & \cos 90^0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 90^0 & 0 & -\sin 90^0 & 0 \\ 0 & 1 & 0 & 0 \\ \sin 90^0 & 0 & \cos 90^0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos 90^0 & \sin 90^0 & 0 \\ 0 & -\sin 90^0 & \cos 90^0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \\ 1 \end{bmatrix}$$

Therefore,

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} z_2 - z_1 \\ y_2 - y_1 \\ -(x_2 - x_1) \\ 1 \end{bmatrix}$$

We use a vector $(1, 0, 0)$ as an example to prove this algorithm, the normal vector for this vector is:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 0-0 \\ 0-0 \\ -(1-0) \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1 \\ 1 \end{bmatrix}$$

This result is reasonably correct and believable despite of the fact that if vector $(1, 0, 0)$ lies in the xz-plane or lies in the xy-plane.

However, it shows that this estimation algorithm has the limitation for some cases, for example vector $(0, 1, 0)$, the normal vector is:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 0-0 \\ 1-0 \\ -(0-0) \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

This result shows the normal vector is the vector itself, which is obviously wrong. However, with this estimated algorithm, the simulation result appears enough realistic; moreover, it requires less computational effort[1].

4.6 Collision Force

If an object continues traveling under a force without colliding with other objects, it will be very difficult to describe objects' motion and elastic response in reality. Collision force is the force to make object bounce away from the fixed interacting plane when elastic object collision happens.There are two steps to describe the collision effects: detection and reaction. Detect the elastic object if particles hit anything; adjust their position by computing the impulse.

4.6.1 Collision Detection

Collision Detection is a geometric problem of determining if a moving object intersected with other objects at some point between an initial and final configuration. In our elastic object simulation system, we are concerned with the problem of determining if any of n particles collide with any of m solid planes.

Perfect Elastic Collision We take one particle collides with a plane shown in Figure 23 as an example. We can detect this collision by inserting the particle position into the plane equation:

$$P(x, y, z) = ax + by + cz + d \tag{17}$$

[1]Our estimation only takes 3 additions vs. 12 multiplications and 6 additions for two cross products and three more additions and divisions for the averaging.

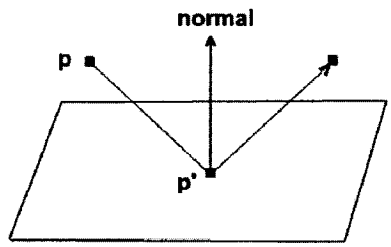


Figure 23: Particle Inelastic Collision and Impact

If $P(x, y, z) > 0$, the particle is within the plane. If $P(x, y, z) = 0$, the particle collides with the plane. If $P(x, y, z) < 0$, the particle penetrates the plane. At each time step, looping through all the particles on the object, each particle is checked if it is outside of the interacting plane.

When the particle i collides with the plane, if there is a perfect elastic collision as in Figure 23, the particle does not lose its energy, so its speed does not change. However, its direction after the collision is in the direction of a perfect reflection.

$$\mathbf{F}^c = 2((P - P') \cdot \mathbf{n})\,\mathbf{n} - (P - P') \tag{18}$$

where

$\mathbf{F}^c$ is the the direction of a perfect reflection

$\mathbf{n}$ is the normal at the point of collision P' and the previous position of particle P

$P - P'$ is the vector from the particle to the surface.

Damped Elastic Collision If there is a damped elastic collision, the particle cannot penetrate the surface, and it cannot bounce from the surface because of the force being applied to it, then we need to apply the damped elastic collision method. The particle loses some of its energy when it collides with another object. The coefficient of restitution of a particle is the friction of the normal velocity retained after the collision. Therefore, the angle of reflection is computed as for the inelastic collision, and the normal component of the velocity is reduced by the coefficient of restitution.

4.6.2 Collision Response

Collision Response is a physics problem of determining the forces of the collision. In elastic collision, elastic object should bounce away from the colliding plane and some energy is lost in the collision response as described in the penalty method.

$$\mathbf{F}^c = -e\,\mathbf{F}^c \tag{19}$$

where e is elasticity of the collision and $0.0 \leq e \leq 1.0$. At $e = 0$, the particle does not bounce at all; $e = 1$, the particle bounces with no friction.

In an one-dimensional object, the boundaries are the walls and floors. In a two-dimensional and three-dimensional object, the particles on the outer layer still follow the same method and same pre-defined boundary as the one-dimensional object. However, for the particles on the inner layer, the boundary is constrained to the outer layer instead of the wall and floor.

4.7 Force Accumulation Algorithm

The following algorithm describes how different forces are accumulated and applied to an elastic object. For a one-dimensional object, some steps will be skipped, for example, there are no other types of spring computations except structural springs because other types of springs only apply on two-layer 2D or 3D objects. Moreover, there are no pressure force accumulation and volume computation because these steps are only available for closed shape objects.

- Step 1: Loop through the number of particles to assign particles with mass value m and compute gravity force $\mathbf{F}^g$. Gravity force, which is independently on each particle, either depends on a constant force, or one or more of particle position, particle velocity, and time [Wit97]. If the object is one-dimensional, the mass of each particle can be different. If the object is two or three dimensional, the mass of the particles on inner or outer layer can also be set differently.

- Step 2: Loop through the number of the structural springs to accumulate the structural spring force.

- Step 3: Loop through the number of the radius springs to accumulate the radius spring force.

- Step 4: Loop through the number of the shear springs to accumulate the shear spring force.
- Step 5: Initialize density as gas, liquid, or rubber inside of the body and introduce some simple physics to describe it. In the current system, only air pressure material is considered and only pressure equation will be used for this extra force computation.
- Step 6: Calculate volume of the inner layer and outer layer of the elastic object.
- Step 7: Calculate the normals of springs on each triangular face to define the pressure force direction.
- Step 8: Calculate the force from the internal air pressure by multiplying the force value by normal vector of the spring.
- Step 9: Accumulate pressure force to each particle.
- Step 10: If users apply the drag force, compute the user applied force and accumulate this force to the dragged particle.
- Step 11: Integrate the object's momentum motion by calculating the derived velocity and its new position for each particle. This step will be explained in next chapter.
- Step 12: Resolve collision detection and response and define the updated position.

Chapter 5

Numerical Integration Methodology

Assume, after the elastic object simulation system creates an elastic object based on the methodology described in Chapter 3 with its initial force state in Figure 24(a) as described in Chapter 4, the system starts the simulation. The simulation system is updated a finite number of times. The object is at the state in Figure 24(b) after 50 discrete time steps.

(a) Elastic Object at the Initial Step

(b) Elastic Object at the Step 50

Figure 24: Elastic Object at Different Time States

In each update, the accumulated impact forces on the object tell it how to change the velocity for next step and result in a re-computation of the forces. The dynamic force applied on this object may be the collision force when the object reaches the boundary; or, the mouse dragging force when user interacts with the object. Overall,

the shape deformation, a mapping of the positions of every particle in the original object to those in the deformed body of this elastic object, is also computed in real time. Therefore, it is important to study differential equations, which govern dynamics and geometric representation of objects [Lin06] and tell us how the velocity and displacement of the particles are integrated dynamically from the knowledge of force applied onto them.

5.1 Differential Equations

Differential equations describe a relation between a function and one or more of its derivatives. The order of the equation is the order of the highest derivative it contains. The elastic object simulation system is associated with initial value problems because it always seeks the particles' velocity and position at next time step $t + h$ from their initial state at time t. We will concentrate on ODE (ordinary differential equation), where all derivatives are with respect to single independent variable, often representing time, such as position and velocity, during the derivate of the state at discrete time steps [Ang03].

$$y' = \mathbf{A}(y, t) \tag{20}$$

where
$\mathbf{A}$ is a function of y and t,
y is a vector, which is the state of the system,
y' is a vector, which is y's time derivative.
Suppose that we integrate the Eq.20 over a short time h

$$y(t + h) - y(t) = \int_t^{t+h} \mathbf{A}(y, t)\, dt \tag{21}$$

where
h is the small stepsize of time,
$y(t)$ is the initial state at the start point t,
$y(t + h)$ is the value we need to find over time thereafter.
Thus

$$y(t + h) \approx y(t) + h\mathbf{A}(y(t), t) \tag{22}$$

5.1.1 Explicit Euler Integrator

The simplest ODE integration method is Explicit Euler Integration method or Forward Euler method. It evaluates the forces at time t, compute derivatives $\mathbf{A}$ at the state of t by multiplying the interval h, and add it to the current state t. Consider a Taylor series expansion as in Eq.23:

$$y(t+h) = y(t) + hy'(t) + \frac{h^2}{2!}y''(t) + \frac{h^3}{3!}y'''(t) + \cdots + \frac{h^n}{n!}\left(\frac{\partial^n y}{\partial t^n}\right) + \cdots \tag{23}$$

Euler method retains only first derivative:

$$y(t+h) = y(t) + hy'(t) + O(h^2) \tag{24}$$

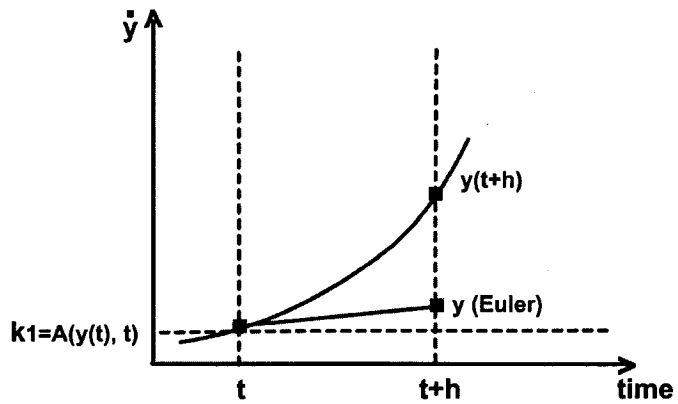


Figure 25: Euler Integrator

We split the series into elements, which we will later use in a re-usable manner throughout integrator framework, where
k_0 which represents the first term in Eq.24, is the initial state

$$k_0 = y(t) \tag{25}$$

k_1 which represents the second term in Eq.24, is the function to find the simplest estimation, the Euler slope of the interval.

$$k_1 = y'(t) = \mathbf{A}(y(t), t) \tag{26}$$

Thus

$$y(t+h) = k_0 + hk_1 \tag{27}$$

We can apply this method iteratively to compute further values at state $t + 2h$, $t + 3h$,.... [BD03] This method is easy to implement; however, it is a low accuracy prototype ODE. In Figure 25, we can see Euler method only calculates the derivative, also called slope, at the beginning of the interval and adds it to the value at the initial state; therefore, it is asymmetric and not stable. .

Pseudocode for Euler Method

```
Line 1: define A(y(t), t)
Line 2: initial values y0 and t0
Line 3: stepsize h and number of steps n
Line 4: for i from 1 to n do
Line 5: k1 = A(y(t), t)
Line 6: y = y + hk1
Line 7: t = t + h
```

5.1.2 Midpoint Integrator

Compared to the Euler method, the one-sided estimate algorithm, midpoint integrator is a symmetric estimate method with a higher per-step accuracy. It computes the derivative at the center of the interval first, then computes the end of the interval. The midpoint integrator, just like others, is based on the Taylor's series. It retains only first three derivative term:

$$y(t+h) = y(t) + hy'(t) + \frac{h^2}{2!}y'(t) + O(h^3) \tag{28}$$

We split the series into elements again for explanation of the method, where k_0, which represents the first term in Eq.28, is the initial state at time t.

$$k_0 = y(t) \tag{29}$$

k_1 which represents the second term in Eq.28, is the function to find the the simplest Euler slope of the interval at time t.

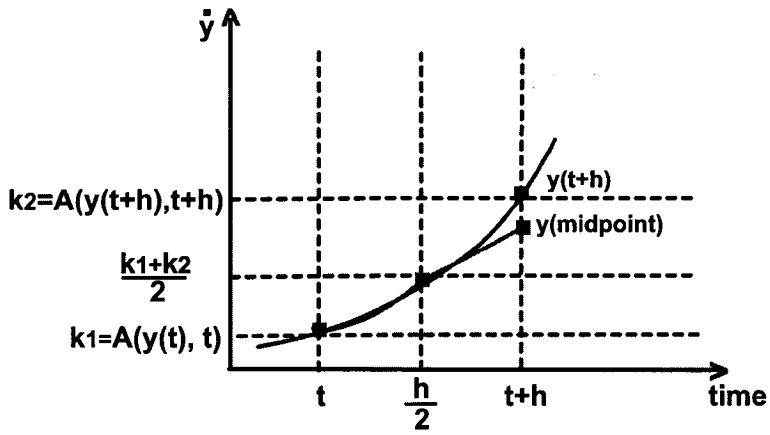


Figure 26: Midpoint Integrator

$$k_1 = y'(t) = \mathbf{A}(y(t), t) \tag{30}$$

k_2 is the function to find the the simplest Euler slope of the interval at time $t + h$.

$$k_2 = y'(t + h) = \mathbf{A}(y(t + h), t + h) \tag{31}$$

Since the unknown $(y + h)$ appears on the right side of Eq.32, in $\mathbf{A}(y(t + h), t + h)$ as one of the arguments of function $\mathbf{A}$, we can use the value obtained using the Euler method in Eq.24.

$$\mathbf{A}(y(t + h), t + h) \approx \mathbf{A}(y(t) + h(y(t), t), t + h) = \mathbf{A}(y(t) + hk1, t + h) \tag{32}$$

The midpoint integration technique obtains a more accurate estimate of the slope than Euler's technique. The following equation computes the integrand at the middle of the interval of t and $t + h$ shown in Figure 26. Thus,

$$y(t + h) = k_0 + h\frac{k_1 + k_2}{2} \tag{33}$$

Compared to Euler Method, Midpoint Method, also called the Runge-Kutta method of order 2, goes from t to $t + h$, we must evaluate function $\mathbf{A}$ twice. By using Taylor's theorem to evaluate the per-step error, we would find that it is now $O(h^3)$. Therefore, this method is more stable than Euler Method with same step size.

Pseudocode for Midpoint Method

```
Line 1: define A(y(t), t)
Line 2: initial values y0 and t0
Line 3: stepsize h and number of steps n
Line 4: for i from 1 to n do
Line 5: k1 = A(y(t), t)
Line 6: k2 = A(y(t+h), t+h)= A(y+hk1, t+h)
Line 7: y = y + h/2(k1+k2)
Line 8: t = t + h
```

5.1.3 Runge Kutta Fourth Order Integrator

Runge Kutta Fourth Order integrator evaluates the derivative four times. It is the most accurate integrator that we describe compared to Euler and Midpoint.

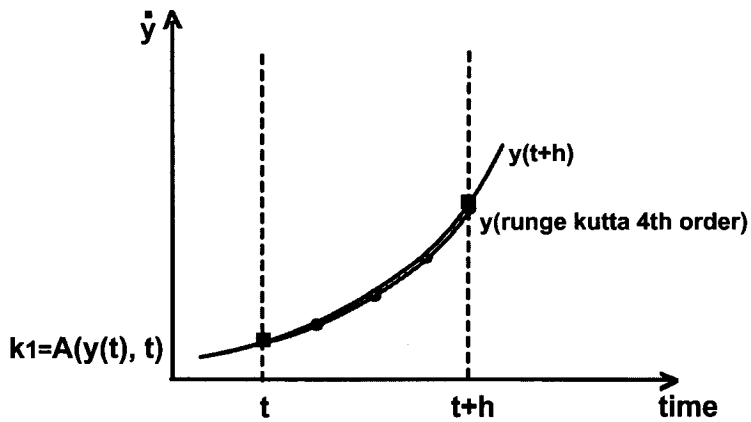


Figure 27: Runge Kutta 4th Order Integrator

The Runge Kutta Fourth integrator, is also based on the Taylor's series. It retains only first five derivative term with a local truncation error $O(h^5)$:

$$y(t+h) = y(t) + hy'(t) + \frac{h^2}{2!}y''(t) + \frac{h^3}{3!}y'''(t) + \frac{h^4}{4!}y''''(t) + O(h^5) \tag{34}$$

$$k_0 = y(t) \tag{35}$$

$$k_1 = y'(t) = \mathbf{A}(y(t), t) \tag{36}$$

$$k_2 = \mathbf{A}(y(t) + h\frac{k_1}{2}, t + \frac{h}{2}) \tag{37}$$

$$k_3 = \mathbf{A}(y(t) + h\frac{k_2}{2}, t + \frac{h}{2}) \tag{38}$$

$$k_4 = \mathbf{A}(y(t) + hk_3, t + h) \tag{39}$$

$$y(t + h) = k_0 + \frac{1}{6}h(k_1 + 2\,k_2 + 2\,k_3 + k_4) \tag{40}$$

where

k_0 is the initial state

k_1 is the slope at the left end of interval,

k_2 is the slope at the middle point using the Euler formula to go from t to $t + \frac{h}{2}$,

k_3 is the second approximation to the slope at the midpoint,

k_4 is the slope at $t + h$ using the Euler formula and the slope k_3 to go from t to $t + h$.

Pseudocode for Runge Kutta Fourth Order Method

```
Line 1: define A(y(t), t)
Line 2: initial values y0 and t0
Line 3: stepsize h and number of steps n
Line 4: for i from 1 to n do
Line 5: k1 = A(y(t), t)
Line 6: k2 = A(y+h/2(k1), t+h/2)
Line 7: k3 = A(y+h/2(k2), t+h/2)
Line 8: k4 = A(y+hk3, t+h)
Line 9: y = y + h/6(k1+2*k2+2*k3+k4)
Line 10: t = t + h
```

5.2 Newton's Laws

After the force accumulation on the object, it is important to find the acceleration **a** in order to define the motion of objects in their next time step. The physical law that governs the motion of objects is the Newton's Second law. It states that the force **F** is proportional to the time rate of change of its linear momentum. Momentum is the product of mass m and velocity **v**.

$$\mathbf{F} \approx m\frac{\Delta \mathbf{v}}{\Delta t} \tag{41}$$

Velocity **v** is the integral of acceleration **a** with respect to the time t. Therefore, integrating the acceleration gives us the new velocity **v**.

$$\mathbf{v} = \int \mathbf{a} dt \tag{42}$$

Position **r** is the integral of velocity **v** with respect to the time t. Therefore, integrating the velocity gives us the new position **r**.

$$\mathbf{r} = \int \mathbf{v} dt \tag{43}$$

Let's take one particle on the object as an example and understand how the different integrators work.

5.2.1 Newton's Laws in Euler Integrator

Based on the Euler Integrator method shown in Eq.24, the new velocity and position of a particle can be integrated follows.

Velocity can be represented as the following equation:

$$\mathbf{v}(t+h) \approx \mathbf{v}(t) + h\mathbf{v}'(t) \tag{44}$$

v_{k0} represents the first term in Eq.44, which is the initial velocity at time t

$$v_{k0} = \mathbf{v}(t) \tag{45}$$

v_{k1} represents the second term in Eq.44, which is the function to compute the derivative velocity in the period h

$$v_{k1} = h\mathbf{v}'(t) = \mathbf{a}(t)h \tag{46}$$

Position can be represented as the following equation

$$\mathbf{r}(t+h) \approx \mathbf{r}(t) + h\mathbf{r}'(t) \tag{47}$$

r_{k0} is the initial position at time t

$$r_{k0} = \mathbf{r}(t) \tag{48}$$

r_{k1} is the function to find the travel position in the period h

$$r_{k1} = h\mathbf{r}'(t) = \mathbf{v}(t)h \tag{49}$$

5.2.2 Newton's Laws in Midpoint Integrator

We apply the midpoint algorithm theory on the Newton's law in order to achieve higher accuracy in the the relationship between the velocity and the position according the Eq.28.

Velocity can be represented as the following equation

$$\mathbf{v}(t+h) \approx \mathbf{v}(t) + h\mathbf{v}'(t) + \frac{h^2}{2!}\mathbf{v}''(t) \tag{50}$$

v_{k0} is the initial velocity at state t

$$v_{k0} = \mathbf{v}(t) \tag{51}$$

v_{k1} is the function to compute the derivative velocity in the period h

$$v_{k1} = \mathbf{v}'(t) = \mathbf{a}(t)h \tag{52}$$

v_{k2} is the function to compute the derivative velocity in the period $t+h$

$$v_{k2} = \mathbf{v}'(t+h) = \mathbf{v}(t) + \mathbf{a}(t)h \tag{53}$$

Therefore, the new velocity of a particle is

$$\mathbf{v}(t+h) = v_{k0} + \frac{v_{k1} + v_{k2}}{2} \tag{54}$$

Position can be represented as the following equation

$$\mathbf{r}(t+h) \approx \mathbf{r}(t) + h\mathbf{r}'(t) + \frac{h^2}{2!}\mathbf{r}''(t) \tag{55}$$

r_{k0} is the initial position at state t

$$r_{k0} = \mathbf{r}(t) \tag{56}$$

r_{k1} is the function to find the travel position in the period h

$$r_{k1} = \mathbf{r}'(t) = \mathbf{v}(t)h \tag{57}$$

r_{k2} is the function to find the travel position in the period $t+h$

$$r_{k2} = \mathbf{r}'(t+h) = \mathbf{r}(t) + \mathbf{v}(t)h \tag{58}$$

Therefore, the new position of a particle is

$$\mathbf{r}(t+h) = r_{k0} + \frac{r_{k1} + r_{k2}}{2} \tag{59}$$

5.2.3 Newton's Laws in the Runge Kutta Fourth Order Integrator

Based on the Runge Kutta Fourth Order method we have shown in Eq.28, the new velocity and position of a particle can be integrated as following.

Velocity can be represented as the following equation

$$\mathbf{v}(t+h) \approx \mathbf{v}(t) + h\mathbf{v}'(t) + \frac{h^2}{2!}\mathbf{v}''(t) + \frac{h^3}{3!}\mathbf{v}'''(t) + \frac{h^4}{4!}\mathbf{v}''''(t) \tag{60}$$

v_{k0} is the initial velocity at time t

$$v_{k0} = \mathbf{v}(t) \tag{61}$$

v_{k1} is the function to compute the derivative velocity in the period h

$$v_{k1} = \mathbf{a}(t)h \tag{62}$$

v_{k2} is the function to compute the derivative velocity of the Euler integration in the period $h/2$ based on the previous step

$$v_{k2} = v_{k0} + \frac{v_{k1}}{2} \tag{63}$$

v_{k3} is the function to compute the derivative velocity of the second approximation based on the v_{k2} in the period $h/2$

$$v_{k3} = v_{k0} + \frac{v_{k2}}{2} \tag{64}$$

v_{k4} is the function to compute the final resulting velocity change of v_{k3} from v_{k0}

$$v_{k4} = v_{k0} + v_{k3} \tag{65}$$

Therefore, the new velocity of the particle is

$$\mathbf{v}(t+h) = v_{k0} + \frac{1}{6}h(v_{k1} + 2\,v_{k2} + 2\,v_{k3} + v_{k4}) \tag{66}$$

If we integrate the velocity vector over time, it gives us how the position vector changed over this time.

Position can be represented as the following equation

$$\mathbf{r}(t+h) \approx \mathbf{r}(t) + h\mathbf{r}'(t) + \frac{h^2}{2!}\mathbf{r}''(t) + \frac{h^3}{3!}\mathbf{r}'''(t) + \frac{h^4}{4!}\mathbf{r}''''(t) \tag{67}$$

r_{k0} is the initial position at time t

$$r_{k0} = \mathbf{r}(t) \tag{68}$$

r_{k1} is the function to find the travel position in the period h

$$r_{k1} = \mathbf{v}(t)h \tag{69}$$

r_{k2} is the function to find the travel position of the Euler integration in the period $h/2$ based on the previous step

$$r_{k2} = r_{k0} + \frac{r_{k1}}{2} = \mathbf{r}(t) + \frac{\mathbf{v}(t)h}{2} \tag{70}$$

r_{k3} is the function to find the travel position of the second approximation based on the r_{k2} in the period $h/2$

$$r_{k3} = r_{k0} + \frac{r_{k2}}{2} \tag{71}$$

r_{k4} is the function to find the travel position change of r_{k3} from r_{k0}

$$r_{k4} = r_{k0} + r_{k3} \tag{72}$$

Therefore, the new position of the particle is

$$\mathbf{r}(t+h) = r_{k0} + \frac{1}{6}h(r_{k1} + 2\,r_{k2} + 2\,r_{k3} + r_{k4}) \tag{73}$$

5.3 Comparison of Three Integrators

5.3.1 Efficiency

For a given step size, Euler is more efficient because it requires only one derivative evaluation per step. Mid Point requires about twice as much computation than the Euler integrator because Mid Point uses two steps to calculate velocity and position at the next time. Runge Kutta Fourth Order requires about four times as much computation as Euler integrator because it use four steps to calculate the velocity and position at the next time step [BD03]. For some configuration, if speed is the priority, Euler integration is convenient to use, but at the expense of accuracy and stability.

5.3.2 Accuracy

Smaller time steps means more stability and accuracy. But also means more computation. If a given step size is h, error of Euler method is $O(h^2)$ as a first-order method, error of midpoint is $O(h^3)$, and error of RK 4 is $O(h^5)$ [BD03].

- The Euler method is based on keep the first two terms of the Taylor series expansion

$$y(t+h) = y(t) + hy'(t) + O(h^2) \tag{74}$$

- An improved method which involves the second derivative is Midpoint method as following

$$y(t+h) = y(t) + hy'(t) + \frac{h^2}{2!}y''(t) + O(h^3) \tag{75}$$

- An improved method which involves the four derivative is Runge Kutta method as following

$$y(t+h) = y(t) + hy'(t) + \frac{h^2}{2!}y''(t) + \frac{h^3}{3!}y'''(t) + \frac{h^4}{4!}y''''(t) + O(h^5) \qquad (76)$$

5.3.3 Stability

With smaller step time value, such as 10 ms, the system integrated by any of the three methods is stable. However, if we give the system a higher step time value, such as 50 or 100 ms, with same mass, damping coefficient, gravity acceleration, the elastic object under Euler system will explode after a short period because its numerical instability causes the mass to oscillate out of control; midpoint and Runge Kutta Fourth Order integrator are more stable [BD03].

Chapter 6

Design and Implementation

In this chapter, we will present the detailed design of the two-layer elastic object physical based simulation system and its implementation.

6.1 Elastic Object Simulation System Design

In this section, an overview of the framework and the algorithm for the elastic simulation system is given.

6.1.1 Domain Analysis-Based Modeling

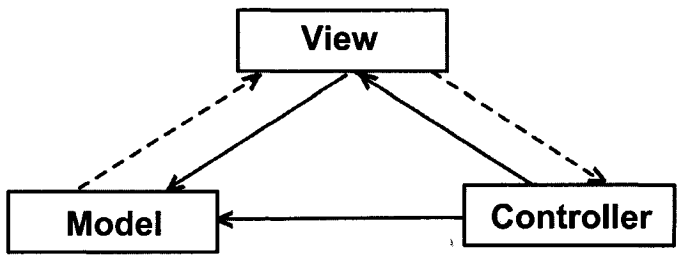


Figure 28: Model-View-Controller

This elastic object simulation system has been designed and implemented according to the well known architectural pattern, Model-View-Controller[Wik07]. This pattern is ideal for real time simulation because it simplifies the dynamic tasks handling by separating data (Model) from user interface (View). Thus, the user's interaction

with the software does not impact the data handling; the data can be reorganized without changing the user interface. The communication between the Model and the View is done through another component: Controller. In our current simulation system, the application has been split into these three separated components:

- Model is an application of object modeling. It stores the geometric modeling methods of the elastic objects and the data of the objects themselves, such as one-dimensional, two-dimensional, and three-dimensional elastic objects and their associated data structure, such as vector, particles, springs, and faces.

- View is the screen presentation to render the Model and a user interface for dynamical simulation. The view in my system is the GLUT window which displays the elastic object and allows the user to use mouse and keyboard to interact with the elastic object.

- Controller handles the processes and responds from the user interaction and invokes the changes to the model. When the user interacts with the elastic object through the GLUT window by dragging it with mouse, the controller handles the new dragging force from the user interface, integrates the new force to find out the change of the acceleration and velocity, and where the object should move to in next display update. This is done through the series of registered GLUT callback functions that process the input from the user.

6.2 Elastic Object Simulation System Implementation

The system is implemented using OpenGL and the C++ programming language with object oriented programming paradigm. Figure 29 describes structure of the software based on the classes.

- The three data structures, such as particle, spring, and face compose an elastic object.

- The elastic object types can be varied by the dimensionality: one-, two-, or three-dimensional.

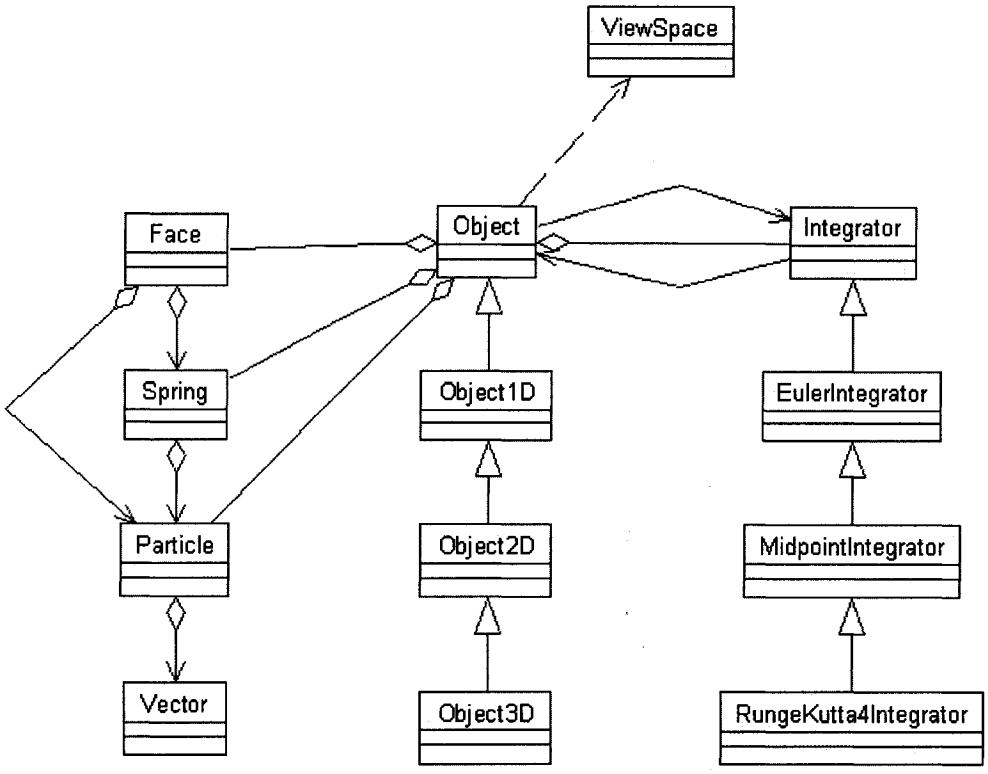


Figure 29: Class Diagram

- The types of integrators are also varied by their complexities, such as Euler, Midpoint, and Runge-Kutta.
- An "Object" instance contains an instance of an "Integrator". The relationship between them is aggregation rather than a common composition because when the elastic object is destroyed, the integrator object is not necessary destroyed. The "Object" has an aggregation of the "Integrator" by containing only a reference or pointer to the "Integrator".
- The classes "Object", "ViewSpace", and "Integrator" are associated to each other based the Model-View-Controller model.

Let's have a close view at each model and the related classes with their parameters and member functions.

6.2.1 Design and Implementation of Data Types

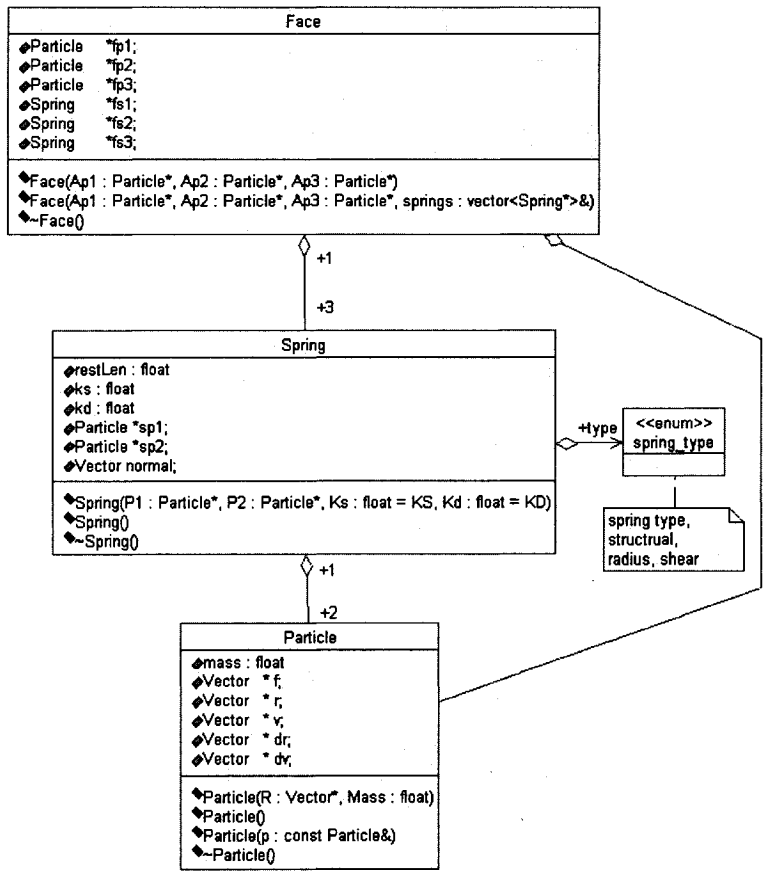


Figure 30: Face-Spring-Particle Class Diagram

The basic data structure is the object vector, which defines the the scalar value with direction. For the second basic data structure, particle, whose properties, such as position, velocity are made up of the object vector. The next higher data structure is spring, which is defined by two particle objects. Face, which is the highest data structure in this simulation system, is composed of three connected springs.

Particle In Figure 30, the particle class shows that each particle has mass *mass*, position **r**, velocity **v**, derivative of position **dr**, derivative of velocity **dv**, and force

vector **f**. Particle constructor sets up its properties with default values.

Spring As shown in Figure 30, the spring class, there are different types of springs to construct the object, such as structural, radius, shear-left, and shear-right springs, declared in the enum type *spring_type* and the default spring type is structural. $*sp1$ is the head of the spring and points to a particle; $*sp2$ is the tail of the spring and points to a particle. *restLen* is the spring length when it is in the resting state. ks is Hooke's spring constant and kd is the spring damping factor. The spring normal vector will be calculated and needed in pressure force calculation.

Face In Figure 30, the face class shows that a face contains $*fp1$, $*fp2$, and $*fp3$ point to the first, the second, and the third particles as three of its vertices. It also contains $*fs1$, $*fs2$, and $*fs3$ point to the first, second, and third spring as three of its edges. There are two face constructors. The first one stores the information of three vertices that point to three particles. It represents faces on two-dimensional objects. The faces will only be needed at the display process.

Figure 31 represents another face constructor along with its algorithm implementation. It accepts three vertices on each face that point to the three particles, and constructs a spring and stores the spring information into the spring vector. This constructor is called by three-dimensional uniform modeling method. The index of face is the key data structure for subdivision method in subroutine. The constructor initializes the three springs based on the three particles. First spring contains particle $p1$ and $p2$; the second spring contains particle $p2$ and $p3$; the third spring contains particle $p3$ and $p1$. A special care is taken not to duplicate existing springs (which would result in incorrect behaviour of the model); therefore, we only allow the new and non-existing springs to be saved in the spring vector. If the first spring already exists with particles $p1$ and $p2$, the new spring $fs1$ will point to the existing spring. Same method is applied on the second spring $fs2$ and third spring $fs3$. Otherwise, the new spring will be pushed and saved into the spring vector. Please refer to the actual code for the complete implementation.

```
Face(Particle *Ap1, Particle *Ap2, Particle *Ap3, vector<Spring*> &springs)
    : fp1(Ap1), fp2(Ap2), fp3(Ap3) {
  fs1 = new Spring(Ap1, Ap2); fs2 = new Spring(Ap2, Ap3); fs3 = new Spring(Ap3, Ap1);
  bool a = false, b = false, c = false;
  for(int o = 0; o < springs.size(); o++) {
    if(springs[o]->sp1 == Ap1 && springs[o]->sp2 == Ap2) {
      delete fs1; fs1 = springs[o]; a = true;
    }
    if(springs[o]->sp1 == Ap2 && springs[o]->sp2 == Ap3) {
      delete fs2; fs2 = springs[o]; b = true;
    }
    if(springs[o]->sp1 == Ap3 && springs[o]->sp2 == Ap1) {
      delete fs3; fs3 = springs[o]; c = true;
    }
  }
  if(!a) springs.push_back(fs1);
  if(!b) springs.push_back(fs2);
  if(!c) springs.push_back(fs3);
}
```

Figure 31: Special 3D Uniform Modeling Face Constructor

6.2.2 Design and Implementation of Components: Model

The class "Object" is the base class for elastic object of any supported dimensionality. It contains the most common data structure and properties of an elastic object. The geometric complexity is increased according to the dimensions. The "Object1D" inherits from the parent class "Object", "Object2D" inherits from "Object1D", and "Object3D" inherits from "Object2D". This type of inheritance hierarchy is in place because when each dimensionality is added, the new object type depends on some of the previous implementation and the new things that come with each additional dimension. For example, 1D object has a notion of structural springs varying in a single dimension; 2D takes the notion of structural springs and augments it with radius and shear springs as well as the notion of pressure inside an enclosed object; 3D extends 2D by adding the notion of face subdivision and volume making object more dynamic in terms of run-time number of vertices (to make it more or less smooth depending on the trade off between quality and performance). All objects share the same $Update()/Draw()$ mechanism, which is used by the OpenGL state machine to update all the vertices of an object in the Model and reflect the changes in the View by drawing the deformations in real-time.

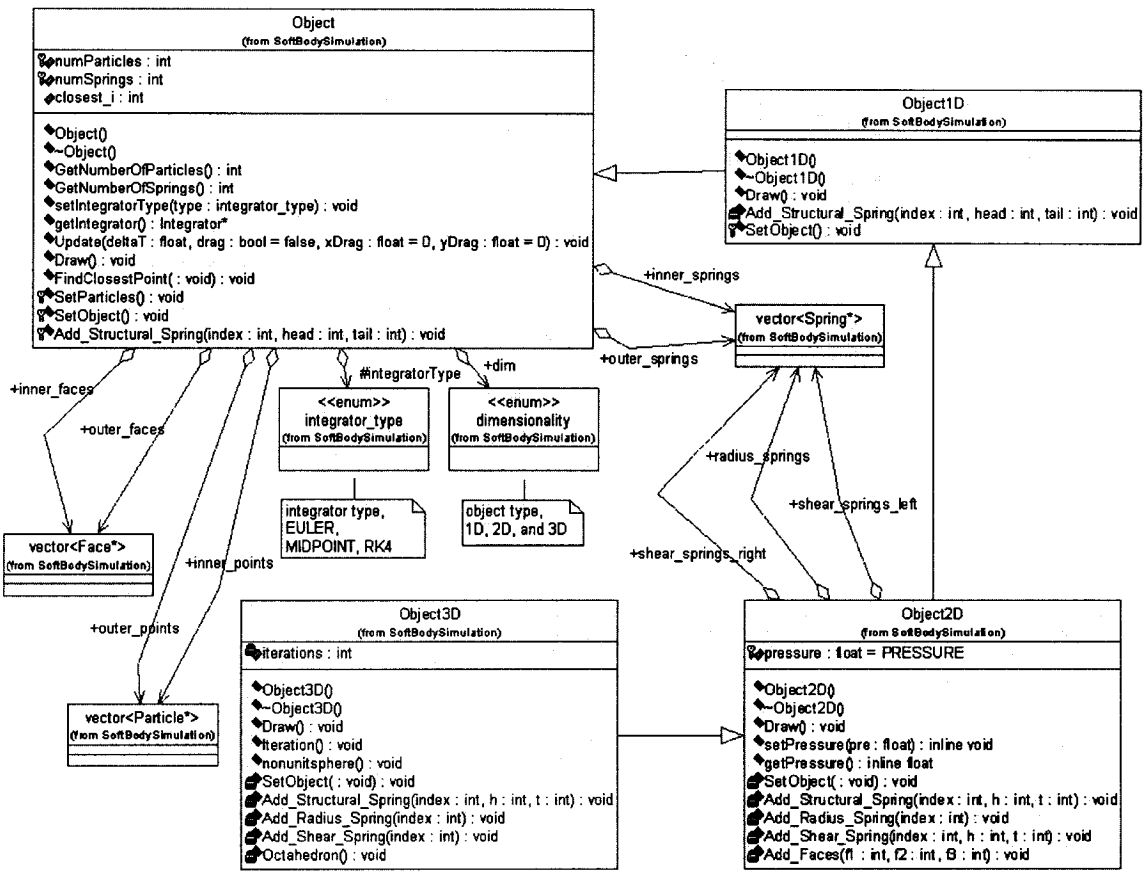


Figure 32: Model Object Class Diagram

Object As shown in Figure 32, the object class, an elastic object contains a particle object, a spring object, a face object, and an integrator object. The data structure varies from inner to outer layers, for example, the pointers to the particles on the inner layer and on the outer layer of the object are saved in different data vectors. *SetObject*() constructs the geometric shape of the elastic object, which, in turn, constructs the particles *SetParticles*() and connects the particles by the structural springs via the *Add_Structural_Spring*() call. The enum type *dimensionality* has one of the values ($DIM1D, DIM2D, DIM3D$) to determine the object's dimensionality type: 1D, 2D, or 3D; the enum type *integrator_type* determines which type of integrator the simulation system uses, Euler, Midpoint, or Runge Kutta Fourth Order integrator. Such design allows extension to add new integrators and select existing integrators at run-time. The variable $closest_i$ is the closest point on the outer layer

to mouse position and *FindClosestPoint*() is the function to find such a particle (used in dragging force application when dragging the object across the simulation window). The function *Update*() modifies the simulated object's state (either each time point when idle or application of the drag force by the user), and determines the object's overall forces, velocity, position in the next time step. *Draw*() visualizes the object after each update.

```
void Idle() {
  object1D.Update(DT, mousedown != 0, xMouse, yMouse);
  object2D.Update(DT, mousedown != 0, xMouse, yMouse);
  object3D.Update(DT, mousedown != 0, xMouse, yMouse);
  glutPostRedisplay();
}
```

Figure 33: *Idle*() Model Updates

```
void Object::Update(float deltaT, bool drag, float xDrag, float yDrag) {
  if(integrator == NULL) {
   switch(integratorType) {
    case EULER:
      integrator = new EulerIntegrator(*this);
      break;
    case MIDPOINT:
      integrator = new MidpointIntegrator(*this);
      break;
    case RK4:
      integrator = new RungeKutta4Integrator(*this);
      break;
    default:
      assert(false);
      return;
    }
    integrator->setDimension(dim);
  }
  integrator->integrate(deltaT, drag, xDrag, yDrag);
}
```

Figure 34: General *Update*() Function

In the main simulation, the *Idle*() function shown in Figure 33, elastic objects update at every time step *DT* to tell the the system how the objects behave and the change for their velocity and position. There are four parameters for *Update*() as

shown in Figure 34, the time step *deltaT*, if there exists user interaction $drag = 0$ by default, the mouse position on x and y axises (for dragging upon mouse release) is at 0 by default. The general algorithm of the *Update*() presented, illustrates that the most of the actual modifications are based on the dynamically selected integrator and the dimensionality of the simulation object being integrated. If in the feature a new integrator is added, this function has to be updated to account for it in the framework.

1D Object In Figure 32, the "Object1D" class shows that an one-dimensional object contains two particles and one spring. The type of particles is *outer_points* and spring type is structural *outer_springs*.

2D Object In Figure 32, the "Object2D" class shows that an two-dimensional object contains inner and outer layers. The type of particles is *inner_points* and *outer_points*. The spring type is structural *inner_springs* and *outer_springs*; moreover, there are another three new types of springs, *radius_springs*, *shear_springs_left*, and *shear_springs_right*. The function *Add_Structural_Spring*() models the shape of the inner circle by connecting *inner_springs* and the outer circle by connecting the *outer_springs* separately. *Add_Radius_Spring*() adds the radius springs with the inner point i and outer point i. *Add_Shear_Spring*() adds the left shear springs with inner point i and outer point $i+1$ and the right shear springs with inner point $i+1$ and outer point i. The variable *pressure*, which is an additional inner force compared to "Object1D", is at each spring along its normal.

3D Object In Figure 32, the "Object3D" class shows that a three-dimensional object uses similar method as a two-dimensional object by extending the variables into the z axis. However, there are two methods introduced to create a three-dimensional object, such as *nonunitsphere*() and *SetObject*(), which uses iteration to define an uniform sphere. The base shape for subdivision a sphere is defined in *Octahedron*() and *Iteration*() computes the coordinates of the newly generated particles and springs based on the level of detail, the variable *Iterations*.

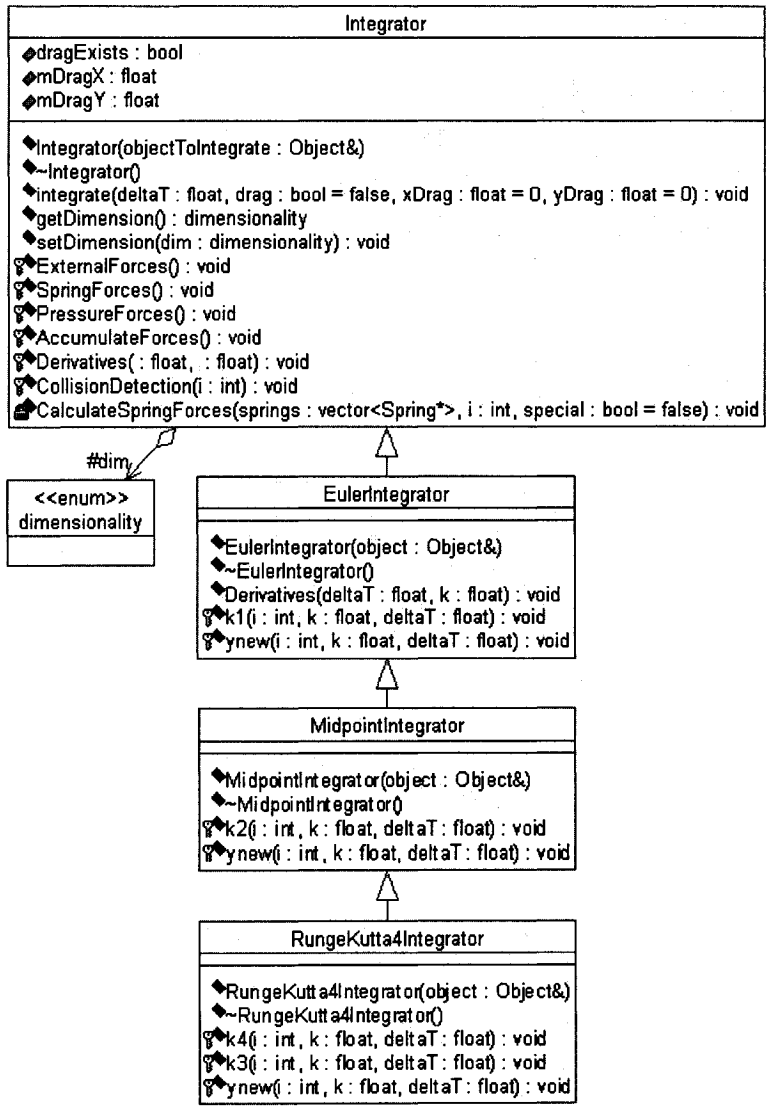


Figure 35: Integrator Framework Class Diagram

6.2.3 Design and Implementation of Components: Controller

The types of integrators are varied by their complexities, such as Euler, Midpoint, and Runge-Kutta. The common attributes and methods are defined in the parent class "Integrator", as shown in Figure 35. The subclasses "EulerIntegrator", "MidpointIntegrator", and "RungeKuttaIntegrator" inherit the super classes based on the complexity. The Euler integrator is a basic building block for other integrators which provides the first step of computation of k_1 in $k1()$. Midpoint integrator uses Euler's

$k1()$ implementation and provides the 2nd step, k_2 implemented in $k2()$. Finally, the RK4 integrator adds the last two refinement steps k_3 (function $k3()$) and k_4 (function $k4()$) in addition to what Euler and midpoint have provided. Thus, RK4 implementation depends on the midpoint which, in turn, depends on the Euler integrator with different parameters.

```
void Integrator::integrate(float deltaT, bool drag, float xDrag, float yDrag) {
  dragExists = drag; mDragX = xDrag; mDragY = yDrag;
  AccumulateForces();
  Derivatives(deltaT, 1.0);
}
...
void Integrator::AccumulateForces() {
  ExternalForces();
  SpringForces();

  switch(dim) {
    case DIM1D:
      break;

    case DIM2D:
    case DIM3D:
      PressureForces();
      break;
  }
}
```

Figure 36: General *integrate*() and *AccumulateForces*() Functions

In Figure 36 there is a general *integrate*() function (which is called from *Object* :: *Update*()) and a general *AccumulateForces*() function, both of which play a vital role in the integrator framework in this thesis. They illustrate the general algorithm of integration applied to the Model's data: first, the effect of all the forces is accumulated (which includes external forces, such as gravity and drag, as well as forces induced by springs and pressure); then, the integrator-specific derivation is performed to each particle of an object. In the general "Integrator" the *Derivatives*() function is pure virtual as is left to be overridden by the "EulerIntegrator", "MidpointIntegrator", and "RungeKutta4Integrator" concrete implementations. It is important to note that the reverse forces are also accounted at the collision detection at the end of each *Derivatives*() implementation. Another note worth mentioning is that the pressure forces are not applicable in the 1D case as there is no enclosed object, which can

hold pressure in this cases. *ExternalForces*() checks for the existence of the mouse drag force (from the user) as well as gravity and sums them up. *SpringForces*() accumulates contributions for all spring types (a subject to dimensionality as well, e.g. 1D case does not have radius or shear springs, only one structural spring).

6.2.4 Simulation Loop Sequence

The sequence diagram in Figure 37 describes the control-flow of the simulation sequence and logic of the elastic object simulation system. The following sequence of steps describes all of the possible states of the elastic object as events occur in greater detail. There we track the different states how the physical simulation loop works, such as display of the objects, accumulation of forces, integration of forces, and so on. In other words, this is the main algorithm of the entire simulation system.

- Step 1: "ViewSpace" initializes the virtual world and provides the user an interactive environment. It provides the interface to allow user to drag the object, or choose the parameters. For example, user can choose the object type, one-dimensional, two-dimensional, or three-dimensional. User can choose the integrator type, Euler, Midpoint, or Runge Kutta 4. User can set up the springs' stiffness, damping variable, and the pressure.

- Step 2: *SetObject*() function creates an elastic object based on the interface variable set from Step 1.

- Step 3: *SetParticles*() function sets up the particles' position and their other initial properties, such as mass and velocity.

- Step 4: *AddSprings*() function connects particles with springs according to their index.

- Step 5: *AddFaces*() connects the springs with faces based on proper index. This step will be ignored if the object is one-dimensional.

- Step 6: *SetIntegratorType*() function tells the Controller which integrator users select through the interface.

- Step 7: *Update*() updates the integrator's time step.

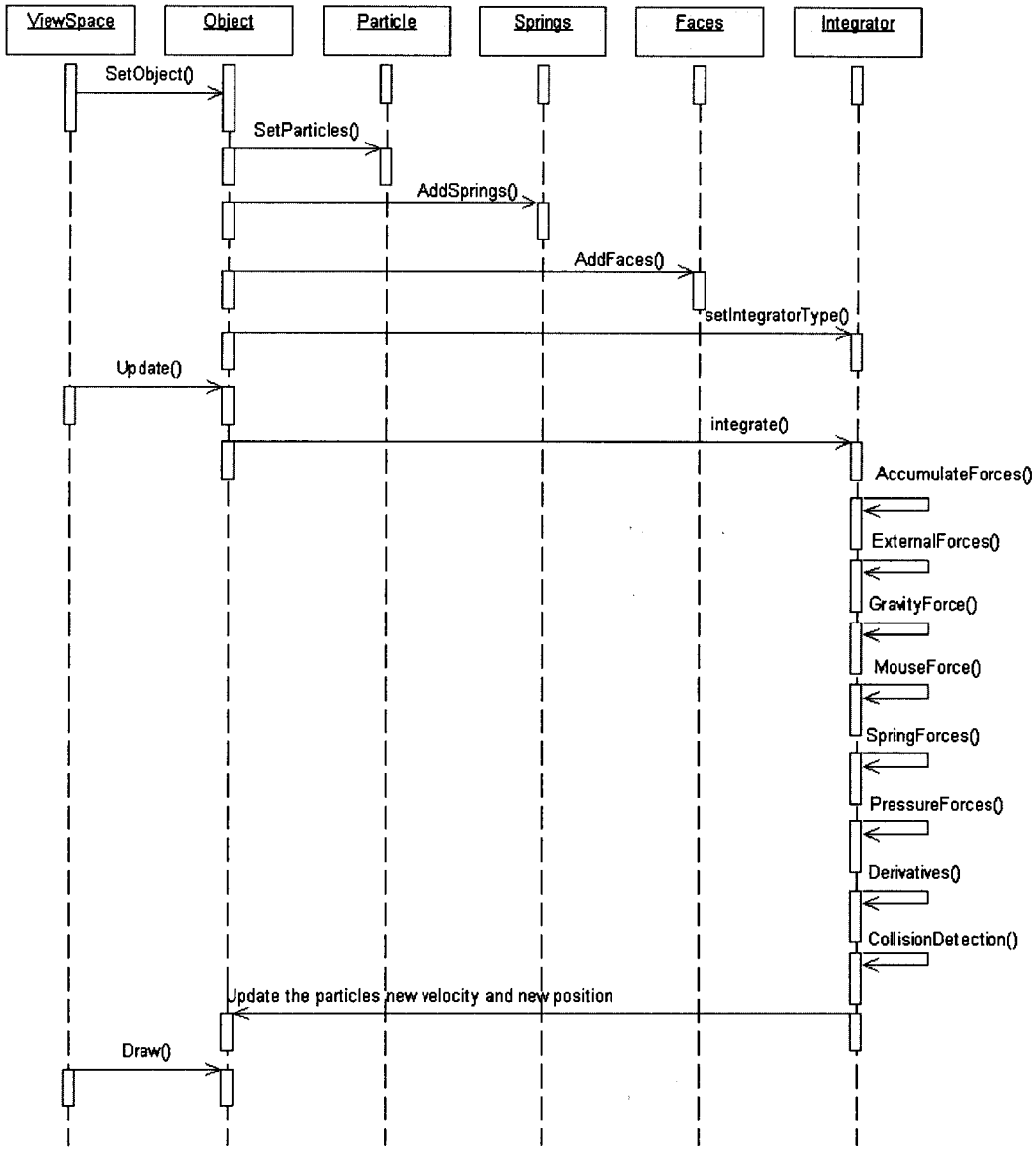


Figure 37: Simulation Loop Sequence Diagram

- Step 8: *Integrate*() contains two functions, *AccumulateForces*() and *Derivatives*(). It is based on all the object geometric information modeled and all the forces information accumulated, to integrate over the time step to get new object position and orientation.

- Step 9: *AccumulateForces*() state is to sum up the forces accumulated on each

particle.

- Step 10: *GravityForce*() is to accumulate gravity force based on the particles' masses.
- Step 11: *MouseForce*() is the external force from the interface when user interacts with the object. It will be added or subtracted from the particles depends on the force's direction.
- Step 12: *SpringForce*() is to accumulate internal force of the particles connected by springs.
- Step 13: *PressureForce*() is to accumulate the internal pressure acted on the particles. For one-dimensional object, this state is omitted.
- Step 14: *Derivatives*() does the real derivative computation of acceleration and velocity in order to get new velocity and position of elastic objects based on the integrator type defined by users.
- Step 15: *CollisionForce*() is to check if the object is out of boundaries after the integration state. If the new position is outside of the boundary, then it will be corrected and reset on the edge of the boundary. Moreover, the new collision force will be added to the object.
- Step 16: *Draw*() displays the object with new position, velocity, and deformed shape.

Chapter 7

Experimental Results

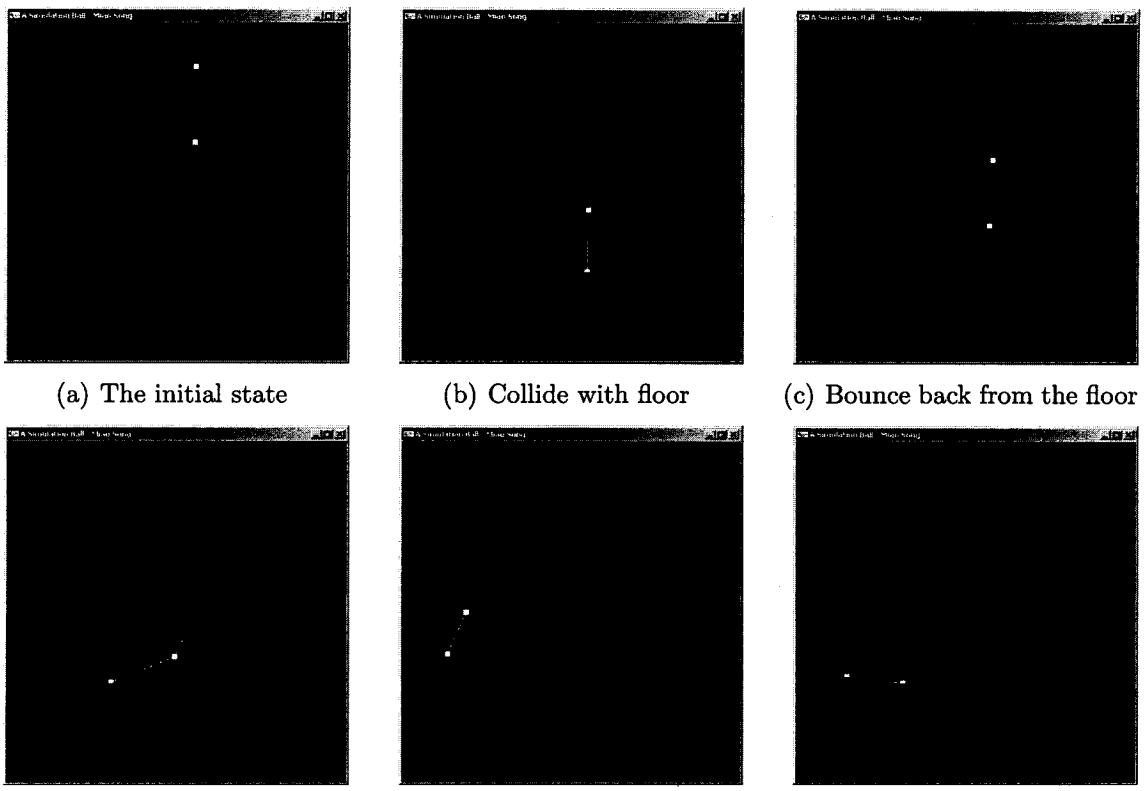
In this chapter, the one-dimensional, two-dimensional, and three-dimensional objects are illustrated at different animation sequences, with different simulation parameters, and by simulation with different numerical integration methods.

7.1 Animation Sequence

The screenshots in this section present the animation sequence of the one-dimensional, two-dimensional, and three-dimensional objects when they are at the initial state, colliding with floor, bouncing back from the floor, responding to user's external dragging, and at the resting state.

7.1.1 1D

This simulation shows two masses connected with one spring. The one-dimensional object moves in a three-dimensional environment, which consists of ceiling, walls, and floor. Users can drag the mass with the mouse to change the object's position and direction. Figure 38(a) presents the initial state of the object; Figure 38(b) shows the object collides with the floor when it drops with gravity force; Figure 38(c) displays the collision response of the object based on the penalty method; Figure 38(d) shows the moment when users drag the object; Figure 38(e) shows how the object reacts on the external impact, such as mouse dragging force or bouncing force with walls; Figure 38(f) displays the object resting on the floor after a while when there is no interaction from the user.



(a) The initial state (b) Collide with floor (c) Bounce back from the floor

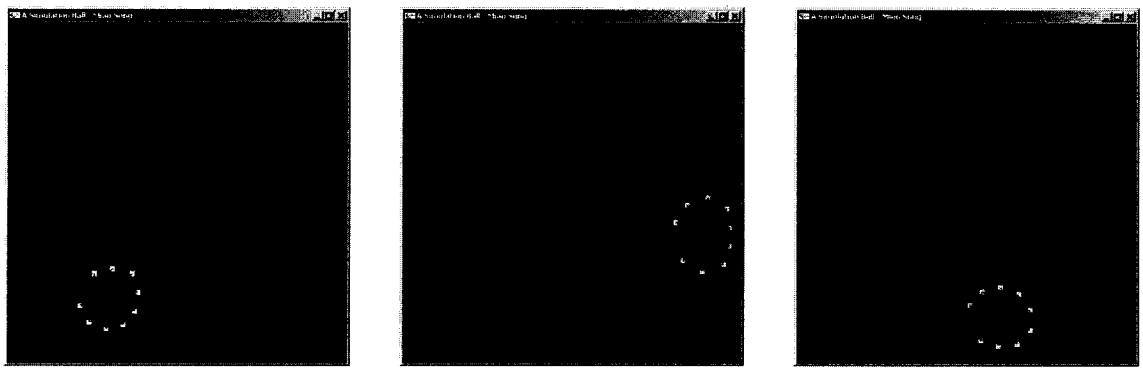
(d) Drag the object (e) Response to compact (f) The resting state

Figure 38: Animation Sequence of One Dimensional Elastic Object

7.1.2 2D

The simulation as shown in Figure 39(a) through Figure 39(f) is how a two-dimensional object moves in a three-dimensional environment. This two-layer object consists of 10 particles and 10 structural springs on both inner and outer circles. Moreover, it contains 10 radius springs, 10 shear left springs, and 10 shear right springs between the inner and outer layers. If a two-dimensional object with only one layer, or the object has no pressure force within, the spring's stiffness has to be a larger value than without, then the object will not collapse. However, as shown in Figure 39(b), if the spring stiffness is small enough, the object does not collapse, neither overlap with the layers because of the stability of the two-layer structure.

(a) The initial state (b) Collide with floor (c) Bounce back from the floor

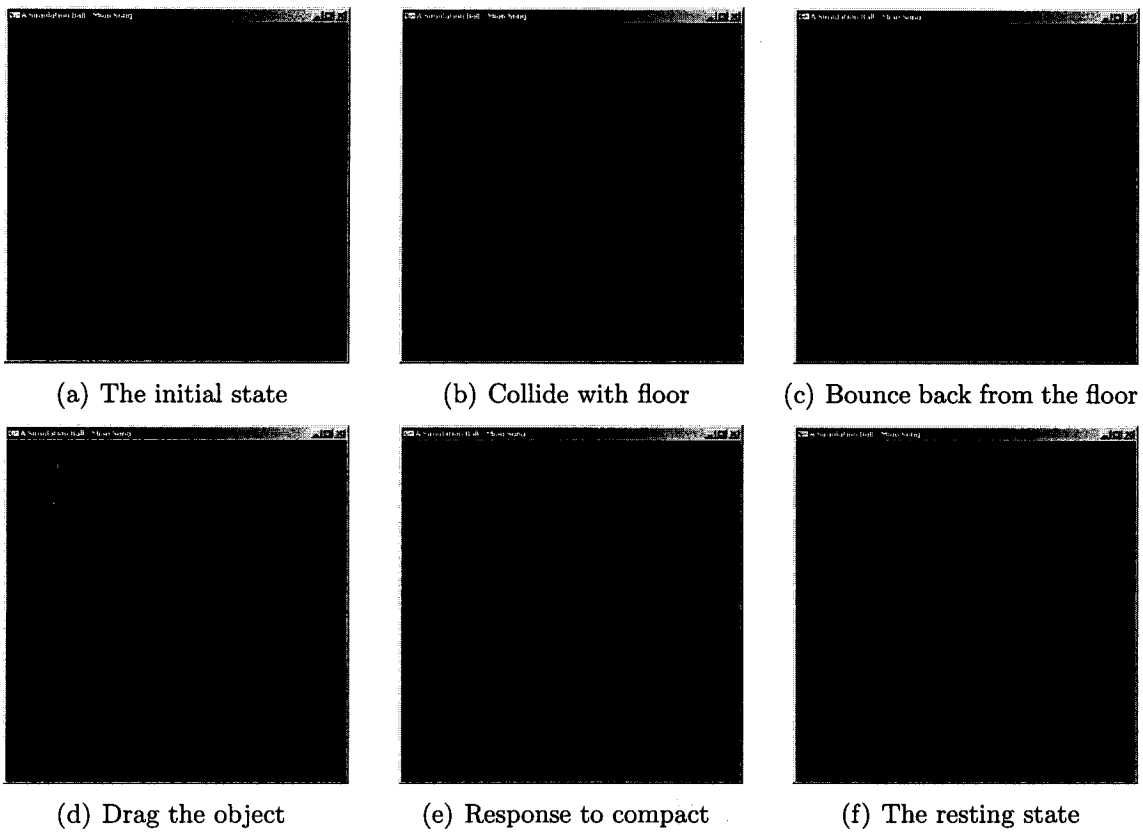


(d) Drag the object (e) Response to compact (f) The resting state

Figure 39: Animation Sequence of Two Dimensional Elastic Object

7.1.3 3D

The simulation as shown in Figure 40(a) through Figure 40(f) is how a three-dimensional uniform facet object moves in a three-dimensional environment. This two-layer object, which is generated by subdividing an octahedron once, consists of 12 particles, 36 structural springs, and 32 faces, on both inner and outer spheres. Moreover, the object also contains 36 radius springs, 36 shear left springs, and 36 shear right springs between the inner and outer layers. Just like in two dimensions, the two-layer structure gives the three-dimensional sphere more stability.



(a) The initial state (b) Collide with floor (c) Bounce back from the floor

(d) Drag the object (e) Response to compact (f) The resting state

Figure 40: Animation Sequence of Three Dimensional Elastic Object

7.2 Simulation Parameters

The parameters in the simulation such as mass, spring stiffness, and friction (damping) can be changed. One can drag the object mass with a mouse to change its position. Effects of different simulation parameters are discussed.

7.2.1 Summary of the Adjustable Parameters

The parameters that influence the behavior of the simulated environment are summarized below, with their default values. Most initial and default values were based on the 2D case from [Mat03]; otherwise, the values are empirical and are partially dependent on the hardware the simulation is executing on.

- KS = 800.0f where KS is structural spring stiffness constant. The larger this

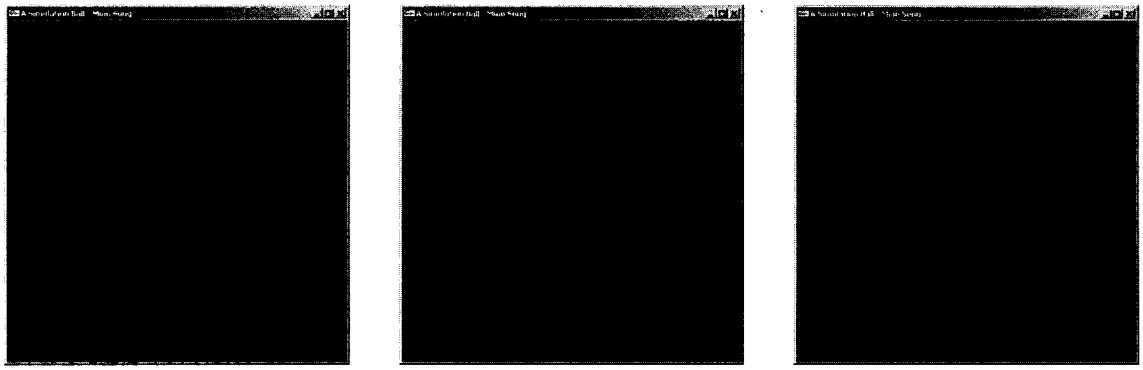
value is, the less elastic the object is and it is more resistant to the inner pressure and deformation. The lesser this value is the more object is deformable and a subject to break up if the inner pressure force is high.

- KD = 15.0f where KD is structural spring damping constant, opposite to the spring retraction force. It denotes how fast the object is to resist its motion.
- RKS = 700.0f where RKS is radius and shear spring stiffness constant, similar to KS, but for radius and shear springs as opposed to the structural springs.
- RKD = 50.0f where RKD is radius and shear spring damping constant, similar to KD, but for radius and shear springs.
- MKS = 150.0f where MKS is the spring stiffness constant of the spring connected with the mouse and the approximate nearest particle on the object. This constitutes the elasticity of the "drag" spring connected to the mouse: the lesser the value is, the more elastic it is, and the harder it is to drag the object as a result.
- MKD = 25.0f where MKD is the damping constant of the spring connect with the mouse and the approximate nearest point on the object.
- PRESSURE = 20.0f where PRESSURE is gas constant used in the ideal gas equation mentioned earlier to determine the pressure force inside the enclosed object. If this constant is too high, and the combined spring stiffness for all the spring types is low enough, the object can "blow up".
- MASS = 1.0f where MASS is the mass for each particle. The object can be made heavier or lighter if this value is larger or smaller respectively, in order to experiment with the gravity effects. Naturally, the heavier objects will be more difficult to drag upwards in the simulation environment. Conversely, the smaller-mass object can be dragged around with less effort given the rest of the parameters remain constant.

7.2.2 Stability vs. Time Step

First, the figures in this section (Figure 41(a), Figure 41(b), and Figure 41(c)) show the stability of the three integrators. We consider the integration time step parameter

in these scenarios only, assuming all the other parameters (discussed later) are not change for the described simulations. As shown in those figures, when the time step is small, such as $DT = 0.003$[1], three of the integrators behave well and the object does not "blow up". However, when one increases the time step by a factor of 10 to $DT = 0.03$, the midpoint (see Figure 42(b)) and RK4 (see Figure 42(c)) integrators are still stable and the object integrated with Euler integrator "blows up" as in Figure 42(a). Furthermore, when the time step is increased 10-fold more to $DT = 0.3$, only the object integrated with RK4 (see Figure 43(c)) is stable and another two objects integrated with Euler (Figure 43(a)) and Midpoint (Figure 43(b)) methods "blow up".



(a) The object integrated with Euler Method

(b) The object integrated with Midpoint Method

(c) The object integrated with RK4

Figure 41: Elastic Object at Timestep = 0.003

7.2.3 Efficiency and Accuracy

The more computational effort is required, the less efficient algorithm is. Likewise, the more accurate algorithm is, the more computation effort it requires, the less efficient it is. Thus, in our simulation system the most efficient and least accurate integration method is Euler's, followed by Midpoint (about twice as more accurate and slower), followed by RK4 (four times slower than Euler's and the most accurate of the three). This can be illustrated in Figure 41(a), Figure 41(b), and Figure 41(c) running concurrently with the same time step of 0.003, where one can see the simulation with

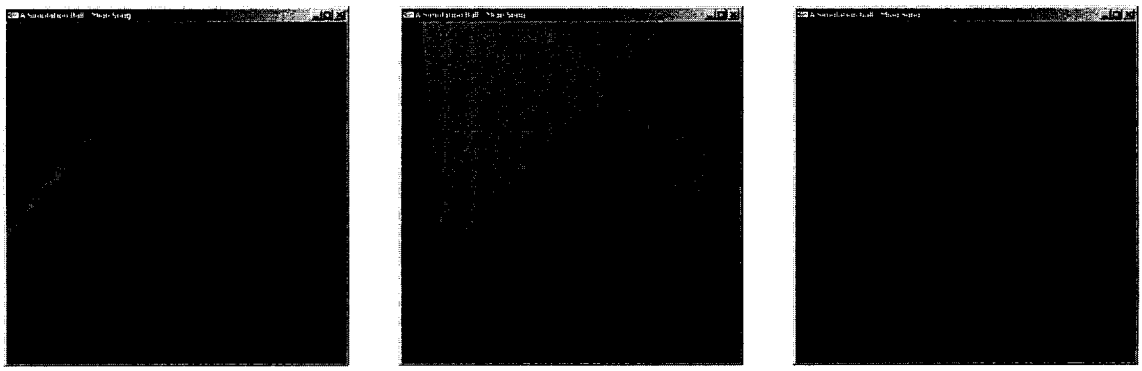
[1]This is an empirical value; dependent on the performance of the hardware.

(a) The object integrated with Euler Method

(b) The object integrated with Midpoint Method

(c) The object integrated with RK4

Figure 42: Elastic Object at Timestep = 0.03



(a) The object integrated with Euler Method

(b) The object integrated with Midpoint Method

(c) The object integrated with RK4 Method

Figure 43: Elastic Object at Timestep = 0.3

Euler's method reaches the floor fastest and RK4 slowest. Of course, the efficiency of the simulation and the accuracy of the shape and movement depends on the amount of particles (and as a result, all kinds of springs) in the object.

7.3 Computational Errors

This section briefly summarizes the error accumulated in the application of the described algorithms and their effects.

7.3.1 Collision Detection

We have applied the Penalty Method in our simulation system. This simple but inaccurate algorithm causes the object to "stick" on the collision surface when dragging the object at the same time and it may become difficult to drag the object away for a period of time.

7.3.2 Subdivision Method

The spherical shape is not perfect round because the number of springs associated to each particle is not uniform. If one wants more quality subdivision has to be done in more than one subdivision operation, but the simulation may rapidly become very slow as the number of particles grow requiring a much greater computational effort, which is suitable only for the high-end hardware if one wishes to do it in real-time. In Figure 44 is an example of the two iterations of the subdivision.

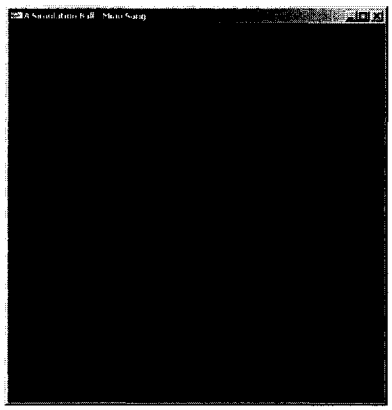


Figure 44: Second Subdivision Iteration

Chapter 8

Conclusion and Future Work

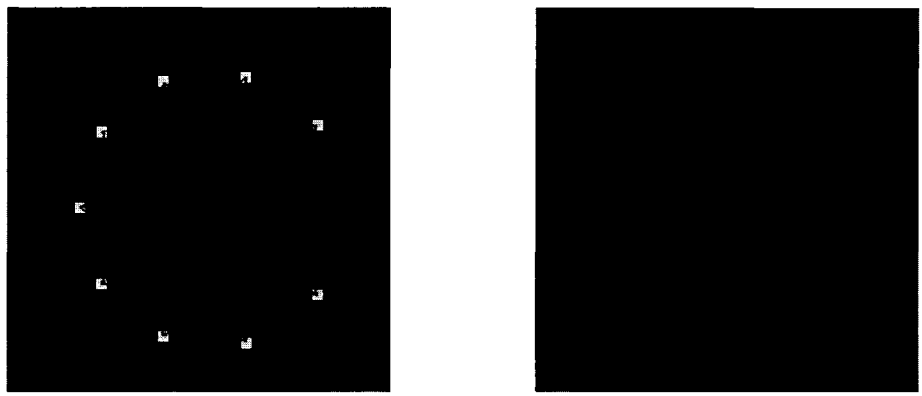
This chapter describes our contribution based on the existing elastic model and analyzes the possible development and related work in the future.

8.1 Contribution

The new model, two-layer elastic object with uniform-surfaces is a simple, efficient approach to imitate the liquid effects of elastic object, such as human's tissue and soft body. Since the modeling and structure of the tissue kind elastic object is closer to real tissue than an one layer object, the level of realism has been increased. The images in this chapter are screenshots from the elastic simulation system we have developed. The modeling method and the density setting provides significant improvements on the conflicts of accuracy and interactivity on previous models. The realism of the results, such as liquid motion and inertia effects are also enhanced.

Procedural Modeling We have applied the procedural modeling method with particle system to model elastic objects. From simple one-dimensional to most complicated three-dimensional object, we introduced the modeling method for different dimensional objects and related physics knowledge gradually. In the elastic object simulation system, each particle has its local coordinate which is easy to be computed at every time step. Moreover, this modeling method can efficiently control the level of detail as required by graphics artists and computer hardware available. As shown

in Figure 45(a) and Figure 45(b), this modeling method also most approximately approaches the ideal equal faces; therefore, the edges(springs) on the faces and the forces on each particle are approximately to be equal at initial state in order to minimize the computation error caused by the object geometry.



(a) Two-dimensional Object (b) Three-dimensional Object

Figure 45: Uniform Shape Modeling

Density As shown in Figure 46(a) and Figure 46(b), the density is defined only for each particle on the elastic surface and the internal density is represented by air pressure physics equation. The weights of particles on inner and outer layer can be set differently. For example, a balloon half filled with liquid, the bottom is heavier than the top part because the density is at the bottom is liquid and top part is air. The weights on inner layer can be set much heavier than outer layer. This special feature gives us flexibilities to imitate different material effects with such simple model.

Inertia Inertia effect is a unique effect in two layer-elastic simulation system, which can not be achieved with one-layer object. Figure 47(a) and Figure 47(b) show the inertial movement of a two-dimensional and three-dimensional elastic object. In Figure 47(a), the inner layer and the outer layer have the opposite internal force drive them along axis x. Since the two layers are connected by springs, the inner particles and outer particles have an extra force applied on them, interactive force between inner and outer particles. And their movement, position, and acceleration will be computed according to the contribution of this extra interactive force. This

(a) Two-dimensional Object (b) Three-dimensional Object

Figure 46: Non-Uniform Density

interactive force does not exist in a single layer object. Figure 47(b) displays the moment when the elastic object drops down onto the ground. The outer and inner particles will fall with the object based on their gravity and springs force. Here, the inertia for inner particle and outer particle are dependent not only on the force from their own motion, the force from the neighbors on the same layer, but also from the interaction on the other layer. This simulation system is more accurate to describe the inertia property happened in the liquid object.

(a) Two-dimensional Object (b) Three-dimensional Object

Figure 47: Liquid Motion and Inertia

Stability The two layered system is stable. Even without the internal pressure force, the shape will not collapse because the two layers are connected by different types of springs. The simulation system works well even with the very inaccurate Euler integrator at large time step, which will result shape collapse or blow up on a one-layer object with the same set of values. We have also implemented the higher level integrators, such as Midpoint and Runga Kutta 4.

Re-usability The design of this simulation system is based on well-known software design pattern. It decomposes the novel concepts into concrete small components. The functions and classes are easy to be plugged and adapted into other program.

This elastic simulation model simplifies the physical modeling method with a group of masses and springs. Also, the simulation is computed in real time based on the numerical integration of the physical laws of dynamics.

8.2 Conclusion

We have developed a one-dimensional elastic object, a two-layer two-dimensional elastic object, and extend it into three-dimensions. These models are all physically based, making use of results from gravity and pressure forces and are implemented with three types of integrations: Euler, Midpoint, and Runge Kutta Fourth Order. The procedural uniform surface generation algorithm provides a convenient mechanism for collision detection. It can generate convincing behaviors when the objects collide with rigid floors or walls because all the particles are checked in every update cycle. Moreover, the rendering is fast because graphics software and hardware renders triangular facets very efficiently.

8.3 Future Work

Character Animation The functionality development of elastic simulation modeling for 3D software design and implementation has emerged as a new challenge in

computer graphics. One of the existing software with the elastic modeling functionality is Maya, which provides shape deformation, especially facial animation, for a group of objects. It is more convenient than traditional frame animation. However, the elastic object movement is not attached to skeleton animation. Furthermore, this elastic simulation is not in real time.

A possible future work that can be done based on the elastic simulation is to define a skeleton system and to map the mesh body onto it. The different parts of the body can be defined as the different freedom of deformable based on the elasticity. For example, the mesh is less elastic on the arms, legs; the mesh is more elastic on the areas that consist fats, like breast, belly. The weight of the elastic property of the muscles can be mapped and dynamically set according to the skeleton. The system can be integrated into advanced animation software as a Plug-in.

Collision Detection between soft objects is a complex phenomenon, which has not been widely developed in physics. In our current system, we are using the penalty methods [MJ88], which do not generate the contact surface between the interacting objects. This method uses the amount of inter-penetration for computing a force which pushes the objects apart instead. Even though the result is fair enough based on estimation, in reality, the contact surfaces should be generated rather than local inter-penetrations. Especially, if we want to use computer animation to imitate organ surgery and help surgeon practice as if interact with real objects, the penalty method is no longer appropriate. There must be a more accurate algorithm to define the collision between rigid body and soft body, or soft body to soft body. Our software should be able to describe other soft body deformation, such as fractures.

Bibliography

[Ang03] Edward Angel. *Interactive Computer Graphics: A top-Down Approach Using OpenGL.* Addison-Wesley, 2003.

[BA97] D. Baraff and A.Witkin. Physically based modeling: Principles and practice. SIGGRAPH97 ACM, Course 34, August 1997.

[BD03] William E. Boyce and Richard C. Diprima. *Elementary Differential Equations and Boundary Value Problems.* John Wiley and Sons, Inc, 2003.

[Bou92] Paul Bourke. *Sphere Generation.* `http://local.wasp.uwa.edu.au/~pbourke/modelling_rendering/sphere_cylinder`, 1992.

[Bou97] Paul Bourke. *Solving Systems Defined by Differential Equations.* `http://local.wasp.uwa.edu.au/~pbourke/modelling_rendering/solver/`, 1997.

[Bou98] Paul Bourke. *Particle System Example.* `http://local.wasp.uwa.edu.au/~pbourke/modelling_rendering/particle/index.html`, 1998.

[Bri98] M. James Bridge. The way balls really bounce. volume 33, pages 236–241. Physics Education, November 1998.

[BW98] David Baraff and Andrew Witkin. Large steps in cloth simulation. pages 43–54. SIGGRAPH 98 Conference Proceedings, July 1998.

[CGCP02] Steve Capell, Seth Green, Brain Curless, and Zoran Popvic. Collisions and deformations: A multi resolution framework for dynamic deformations. pages 41–48. Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation, July 2002.

[DL02] Nixon D. and R. Lobb. A fluid-based soft-object model. volume 22, Iss. 4, pages 68–75. Comp. Graph. and App. IEEE, July 2002.

[DMB00] Desbrun, M.P.Cani, and A. Barr. Adaptive simulation of soft bodies in real-time. pages 133–144. Computer Animation, May 2000.

[DW92] D.Baraff and A. Witkin. Dynamic simulation of non-penetrating flexible bodies. volume 26, No. 2, pages 303–308. Computer Graphics, 1992.

[FW98] Jessica Fitch and J. Reed Walker. *Deformable ball simulation.* `http://vorlon.case.edu/~jrw24/CG/`, 1998.

[GM97] F. Gibson and Brain Mirtich. A survey of deformable models in computer graphics. Mitsubishi Electric Research Laboratories TR97 -19, November 1997.

[Gro01] Peter Grogono. *A Latex2e Gallimaufry Techniques, Tips, and Traps.* Department of Computer Science, Concordia University, 2001.

[Gro02a] Peter Grogono. *Getting Started with OpenGL.* Department of Computer Science, Concordia University, 2002.

[Gro02b] Peter Grogono. *Lecture Notes of Advanced Computer Graphics.* Department of Computer Science, Concordia University, 2002.

[HB02] Donald Hearn and M.Pauline Baker. *Computer Graphics.* Prentice Hall, 2002.

[Hec96a] Chris Hecker. Behind the screen: Physics, part 2: Angular effects. pages 14–22. Game Developer Magazine, December 1996.

[Hec96b] Chris Hecker. Physics, the next frontier. pages 12–20. Game Developer Magazine, October 1996.

[Hec97a] Chris Hecker. Physics, part 3: Collision response. pages 11–18. Game Developer Magazine, March 1997.

[Hec97b] Chris Hecker. The third dimension. pages 17–18. Game Developer Magazine, June 1997.

[HO99] J.K Hodgins and J.F. O'Brien. Computer animation. volume 3, pages 686–690. The Wiley Encyclopedia of Electrical and Electronics Engineering, John G. Webster Ed., 1999.

[JS03] J.Teran and S.Blemker. Cloth and deformable bodies: Finite volume methods for the simulation of skeletal muscle. pages 68–74. Euro.Symp. on Comp.Anim. (SIGGRAPH Proc), 2003.

[KCH00] Y.M. Kang, J.H. Choi, and H.G.Cho. Fast and stable animation of cloth with an approximated implicit method. pages 247–262. Computer Graphics International, 2000.

[Ker07] Erwin Keryszig. *Advanced Engineering Mathematics.* John Wiley and Sons, Inc., 2007.

[KO03] Kazuya G. Kobayashi and Katsutoshi Ootsubo. Ffd: free-form deformation by using triangular mesh. pages 226–234. Proceedings of the eighth ACM symposium on Solid modeling and applications, 2003.

[Lan99a] Jeff Lander. Devil in the blue faceted dress: Real-time cloth animation. pages 17–21. Game Developer Magazine, May 1999.

[Lan99b] Jeff Lander. Lone game developer battles physics simulator. pages 15–17. Game Developer Magazine, April 1999.

[LH02] Matti Larsson and Christian Holmboe. *VR Dynamics - A primer.* `http://www.cs.umu.se/kurser/TDBD07/VT02/uppsproj/resultat/uppsats/Matti&Christian/VR_Dynamics_023.doc`, 2002.

[Lin06] Ming C. Lin. *Lecture for Class: Physically-Based Modeling, Simulation and Animation.* Department of Computer Science College of Arts and Sciences, The University of North Carolina at Chapel Hill, 2006.

[Mat03] Maciej Matyja. *Pressure Model of Soft Body Simulation.* SIGRAD2003 conference in UMEA (Sweden) paper, November 2003.

[Mat04a] Maciej Matyja. *How to Implement a Pressure Soft Body Model.* `http://panoramix.ift.uni.wroc.pl/~maq/soft2d/index.php`, 2004.

[Mat04b] Maciej Matyja. *Inverse Dynamic Displacement Constraints in Real-Time Cloth and Soft-Body Models in Graphics Programming Methods.* Charles River Media, Inc., 2004.

[McE05] Kevin J. McElwee. *Keratin - Introduction to skin and hair biology.* `http://www.instylehair.com/modules.php?name=News&file=article&sid=7`, 2005.

[MDMJ02] Matthias Muller, Julie Dorsey, Leonard McMillan, and Robert Jagnow. Collisions and deformations: Stable real-time deformations. pages 49–54. Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation, July 2002.

[MJ88] M.Moore and J.Wilhelms. Collision detection and response for computer animation. volume 22, No. 4, pages 289–298. Computer Graphics, August 1988.

[Oli01] Gustavo Oliveira. *Exploring Spring Models.* `www.gamasutra.com/features/20011005/oliveira_01.htm`, 2001.

[Pla88] Barr Platt. Constraint methods for flexible models. volume 22, No. 4, pages 279–288. SIGGRAPH88, August 1988.

[Pro95] Xavier Provot. Deformation constrains in a spring-mass model to describe rigid cloth behavior. pages 147–154. Graphics Interface, May 1995.

[Ree83] William T Reeves. Particle systems - a technique for modeling a class of fuzzy objects. volume 17, pages 359–376. Computer Graphics, 1983.

[TF96] Thomas and Finney. *Calculus with Analytic Geometry.* Addison-Wesley, 1996.

[TF98] Demetri Terzopoulos and Kurt Fleischer. *A survey of deformable models in computer graphics*, volume 7. The Visual Computer, November 1998.

[TJ84] Frank Thomas and Ollie Johnston. *Disney Animation: The Illusion of Life.* Abbrville Press, 1984.

[TJF89] Demetri Terzopoulos, J.Platt, and K. Fleischer. From gloop to glop: heating and melting deformable objects. pages 219–226. Proceedings of Graphics Interface, 1989.

[TM92] Demetri Terzopoulos and Dimitri Metaxas. Dynamic deformation of solid primitives with constraints. volume 26, No. 2, pages 269–278. ACM SIGGRAPH Computer Graphics, July 1992.

[TPBF87] D. Terzopoulos, J. Platt, A.H. Barr, and K.W. Fleischer. Elastically deformable models. 21(4):205–214, 1987.

[TW88] Demetri Terzopoulos and Andrew Witkin. Physically-based models with rigid and deformable components. In *Proc. Graphics Interface*, pages 146–154, June 1988.

[TW90] Demetri Terzopoulos and K. Waters. Physically-based facial modeling, analysis, and animation. volume 1, pages 73–80. Journal of Visualization and Computer Animation, 1990.

[Wag07] Sean Wagstaff. *Maya Complete 6: Industry-Strength Visual-Effects Application Upgrades Tools, Adds Photoshop Support.* `http://www.macworld.com/2004/12/reviews/mayacomplete6/index.php`, 2007.

[WB93] A. Witkin and D. Baraff. An introduction to physically based modeling. SIGGRAPH Course Notes 32, 1993.

[Wik07] Wikipedia. *Procedural Modeling.* `http://en.wikipedia.org/wiki/`, 2007.

[Wit97] Andrew Witkin. *Particle System Dynamics.* SIGGRAPH97 Course Notes `http://www.cs.cmu.edu/afs/cs/user/baraff/www/sigcourse/notesc.pdf`, 1997.

[WM86] G. Wyvill and C. McPheeters. *Data Structures for Soft Objects*, volume 22, No. 3. The Visual Computer, April 1986.