Simulation Techniques For Deformable Animated Characters

by

Remco K. Chang

B. A., Johns Hopkins University, 1997

A thesis submitted in partial fulfillment of the
requirements for the Degree of Master of Science
in the Department of Computer Science at Brown University

Providence, Rhode Island

April 2000

Abstract of "Simulation Techniques For Deformable Animated Characters" by Remco K. Chang, Sc. M., Brown University, April 2000.

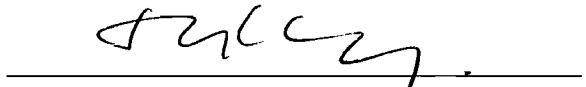
This thesis surveys three different deformation models for physically based simulation. The ability to model and manipulate deformable objects is essential to many computer graphics applications, especially in modeling and simulation of realistic and complex environments. Approaches for modeling object deformation, however, have been constrained by the limitation of hardware. Although realistic behavior of the deformed objects is most desirable, many methods sacrifice the realism for speed.

The three deformation models that we survey are based on individual nodal point analysis, a reduced degree of freedom model, and an approximate continuum model, each with its own strengths and weaknesses. We examine these deformation models using the same set of parameters, and measure their performance in terms of speed, stability, and robustness.

AUTHORIZATION TO LEND AND REPRODUCE THE THESIS

As the sole author of this thesis, I authorize Brown University to lend it to other institutions or individuals for the purpose of scholarly research.

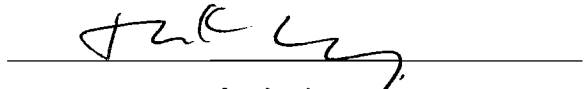
Date 4/28/2000



author's signature

I further authorize Brown University to reproduce this thesis by photocopying or other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Date 4/28/2000



author's signature

This thesis by Remco K. Chang is accepted in its present form by
the Department of Computer Science as satisfying the thesis requirement
for the degree of Master of Science.

Date 4/28/2000

Nancy S Pollard

Dr. Nancy Pollard, Director

Approved by the Graduate Council

Date _____

_____

Peder J. Estrup
Dean of the Graduate School and Research

Acknowledgments

I would like to thank my advisor Nancy Pollard for helping me find my thesis topic, stepping through all the algorithms and derivations we found in related papers, and reviewing and editing countless drafts of this thesis. Without her guidance, this thesis would not have been possible. I would also like to thank Daniel Acevedo, Vasiliki Chatzi, Galina Shubina, John Hughes, and David Laidlaw for brainstorming with me on my crazy ideas. Lastly, I would like to thank Sonia Leach, Nathan Lauster, and Emmanuel (Manos) Renieris for proof-reading and formatting this thesis.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Simulating *3D* continuous mesh models using physically based deformation techniques has always been a difficult task. In the past two decades, there has been a significant amount of research involving deformable objects, but only in recent years has the effort been concentrated in the area of animation. The limitation of hardware has probably been the primary reason for the delay, and with increasing computational power, researchers are now able to simulate more complex objects and behaviors. Use of mesh models for *2D* objects is becoming commonplace as simulations of *2D* meshes approach interactive speed, but research in the *3D* realm remains scarce and slow.

In the movie *Flubber*, animators at Disney hand animated the dance sequence of each flubber. Flubbers possess the unique characteristic of being "Jello-like" or "putty-like" while performing complex movements. The animators therefore keyframed not only the kinematics of their motion, but also an obvious amount of "stretch and squash" to give them the look and feel of soft rubber. The goal of this thesis is to explore different approaches to automate the dynamic behavior of a flubber-like material. We intend to animate flubbers as viscoelastic models driven by motion captured dance sequences.

To create realistic motion, we use motion capture data as input for our model. Obtaining motion capture data is a relatively fast procedure compared to traditional keyframing, and motion capture data is becoming more widely available. Using motion capture data, however, requires the deformable model to be a passive system where the motion is applied to the character, and the "Jello-like" behavior is passively simulated. Most deformation systems currently available are passive only to *external* forces such as collision forces or other type of constraints. In this thesis, we translate the motion capture data into *internal* forces and simulate motion as end results of applying these internal forces.

In most character animation systems, individual body parts may be represented as rigid objects, which creates seams or discontinuities between body parts. To circumvent this problem, we choose to deform the character model as a fully enclosed continuous mesh with volume. Because the motion capture data we possess describe the character movements based on the position and orientation of each body part during each frame, we first have to *map* regions of the continuous mesh to the body parts. Then we employ volume deformation techniques to animate the character's underlying structure, thus deforming the mesh.

We implemented three different deformation models and tested their performance under different sets of parameters. The models that we tested are a mass spring model, a reduced degree of freedom model, and an approximate continuum model. These models have different characteristics that are difficult to quantify when simulating a complex character, therefore we tested these models using a simple geometric cube.

Chapter 2

Related Work

Deformation has been studied for more than two decades in computer graphics. Gibson and Mirtich [8] have done an extensive technical report covering the major techniques and their strengths and weaknesses. The two main categories of deformation methods are non-physically and physically based modeling. The most prominent technique in non-physically based modeling is free form deformation, and the most common techniques in physically based modeling include mass spring models, continuum models and finite element methods, approximate continuum models, and reduced degree of freedom models.

2.1 Free Form Deformation

Free form deformation (FFD) is a general deformation technique that is based on space warping functions introduced by Barr [9]. Sederberg and Parry [10] coined the term "Free Form Deformation" when they created *lattices* that define the space to be deformed, thus adding an intuitive interface for the user. The lattices are defined by a set of points, and by moving the points, the space enclosed by that lattice is warped to a new shape.

The principle of FFD has been extended in many ways. Coquillart introduced *extended free form deformations* [11] to create lattices of non-parallelpipedical shapes, and then added *animated free form deformation* [12] to animate the deformed objects. To allow more intuitive control, Hsu et al [13] allowed direct manipulation of the deformed objects on top of FFD. McCracken and Joy [14] adopted Catmull-Clark subdivision surfaces [7] to subdivide three-dimensional lattices, creating lattices of arbitrary topology.

Because FFD is a space warping modeling tool, it requires the user to define lattices of the necessary shape and detail on top of keyframing the deformations associated with the motions. This is often a non-trivial task for complex meshes, thus requiring many levels of

refinements before the user could achieve the desired deformation.

2.2 Mass Spring Models

Mass spring models are perhaps the most intuitive tool for modeling deformable objects, and have been studied and used extensively in modeling and simulation. Objects in mass spring models are composed of a collection of nodal points connected by springs in a lattice structure. Springs can have different properties for simulating different material behaviors. With careful placement of nodal points and selection of the springs properties, mass spring models can model very complex and realistic objects.

Mass spring models have been widely in animation. Chadwick et al [15] combined mass spring models with free form deformation to animate cartoon characters. Tu and Terzopoulos [16] modeled the muscles of artificial fish using a mass spring system based on an implicit Euler method. Waters [18] used mass spring systems in modeling facial animation. They gave their springs different properties to represent the dermal, fatty tissue, and muscle layers of human skin.

Although mass spring models are easy to construct, they certainly have their drawbacks. Using discrete nodal points and springs to model realistic objects as continuous bodies is a significant approximation. The selection of nodal points can often create the problem of either over or under sampling, and the selection of spring properties such as spring constants and damping constants is often difficult. Furthermore, simulating rigid objects with high spring constants can create a *stiffness* problem causing the numerical integrator to take smaller time steps in order to maintain numerical stability.

2.3 Continuum Models and Finite Element Methods

Unlike mass spring models, continuum models and finite element methods (FEM) treat the deformable objects as solid bodies with continuous mass and energy, and derive the numerical integration from equations of continuum mechanics. Although FEM still requires a discrete time step as an approximation of real elapsed time, the continuum mechanics provide a more physically realistic simulation than mass spring models.

The flip side to the continuum mechanics and FEM is the computational requirement. Due to this limitation, the use of FEM has been limited in computer graphics until recent years. Celniker and Gossard [19] applied FEM to generate primitives that build continuous meshes designed to support a free form modeling paradigm. Gourret et al [20] modeled

interactions between a human hand performing a grasping task on a deformable object. Chen and Zeltzer [21] captured the geometry and underlying material properties of muscles using FEM. Bro-Nielse and Cotin [22] applied FEM for modeling human tissue deformation for surgical simulation. O'Brien and Hodgins [23] formulated their FEM using internal energy to model stress and fracture of deformable and rigid bodies. Faloutsos et al [24] combined free form deformation and FEM to create dynamic deformable objects with the look and feel of cartoons, and Baraff and Witkin [1] demonstrated that even with the heavy computation required by FEM, they could still achieve near interactive time modeling complex cloth models.

2.4 Approximate Continuum Models

In between the discrete models created with the mass spring system and the continuous mass and energy models described by FEM lie the approximate continuum models. These models utilize the same continuous energy concept as FEM, but formulate it discretely to achieve certain desired effects.

Kass, Witkin, and Terzopoulos [25] introduced *snakes* for solving low level tasks in computer vision such as edge or line detection, stereo matching, and motion tracking. Snakes respond interactively to internal forces that resist stretching and bending based on energy minimization. Terzopoulos et al [26] employed elasticity theory for deforming curves, surfaces, and solids for animation applications. Their method used a discretized continuum model for the potential energy due to deformation. Terzopoulos and Witkin [27] represented a deformable object based on a rigid reference body that captures the rigid-body motion, and a discretized deformation function that defines the movements of mesh points. Terzopoulos and Fleischer [28] further expanded this technique to simulate viscoelasticity, plasticity, and fracture in deformable bodies.

2.5 Reduced Degree Of Freedom Models

Simulating physically based models using the techniques discussed above often leads to systems with many degrees of freedom since the object's state is defined by the nodal points' positions and velocities. The high degree of freedom can cause systems to slow during simulation, thus limiting their use in real time settings. Low degree of freedom models address this problem by sacrificing certain physical attributes for speed.

Pentland and Williams [29] developed a simplified FEM expression for the dynamics

of deformable bodies using modal analysis. Their system allows the user to independently compute different *modes* of deformation. These modes could be summed up to create a more accurate simulation, or the user could selectively ignore modes that are often not necessary for the purpose of most animation and computer graphics applications. Witkin and Welch [30] adopted a technique for animating and globally deforming bodies using the space-warping functions introduced by Barr [9]. They added a time dependency in the process, which allows for animating deformable objects. Baraff and Witkin [31] further expanded this technique to simulating combinations of rigid and deformable bodies with non-penetration constraints.

2.6 Our Models

In this thesis, we implemented a mass spring model, a reduced degree of freedom model, and an approximate continuum model. We examined the possibility of using FFD as a deformation tool based on the concept presented by Chadwick[15], but the fact that FFD is a space warping technique ignoring the structure of the deformed object makes it a poor choice for our purpose (Figure (2.1)).

Our mass spring model is based on simple linear springs, and thus similar to that of many generic mass spring models. The reduced degree of freedom model is based on the work by Witkin and Welch [30]. In this approach, deformation is limited to a 2nd order function, ignoring the higher frequency deformation to decrease computation time. Our approximate continuum model adopts the implicit integration technique used by Baraff and Witkin[1]. The deformation is based on energy functions described by finite element methods, but the numerical integrator calculates the energy discretely on each individual nodal point.

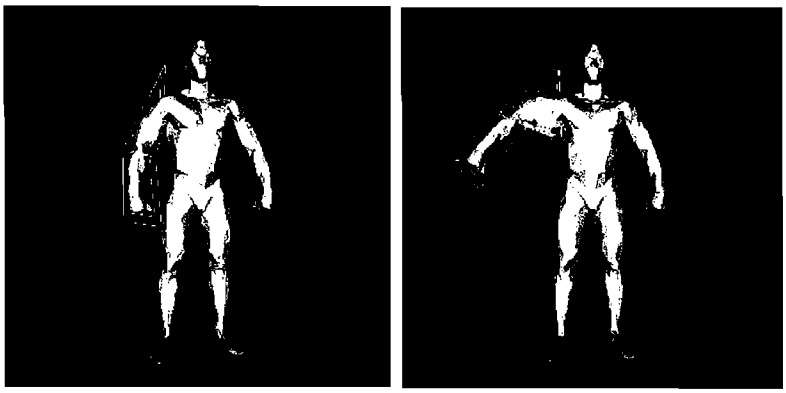


Figure 2.1: Using FFD to deform a continuous mesh model. The character's arm is deformed based on the change in the shape of the lattices. Since FFD is a space warping technique, the structure of the underlying mesh is not being considered.

Chapter 3

Creating the Model

We have chosen to animate any given mesh model by giving the model a set of *skeletons*, and using the mesh as a layer of *skin*. The skeletons make up a rough representation of the skin itself. But unlike the skin, skeletons cannot be deformed. During the initialization process, we connect the skeletons and the skin with sets of springs creating a mass spring system. Moving or rotating the rigid skeletons then would push and pull on the springs, causing the skin to deform.

3.1 Notation

For the rest of this thesis, we will use bold faced letter to denote a vector containing information for a collection of elements. For example, $\mathbf{x}$ would denote the position vector of a collection of points; $\mathbf{x}_i$ will denote the (x, y, z) coordinate of the element i, and x_i will be the equivalent of the x coordinate of $\mathbf{x}_i$. Furthermore, we will also let $\tilde{\mathbf{x}}$ denote an *undeformed* position vector.

3.2 Model Generation

We created two sets of models. The first has a simple geometric shape of a cube as skin and a single rectangular block as skeleton (Figure (3.1)), and the second is a full character with 19 independent skeletons representing 19 separate body parts (Figure (3.2)). The character's skin is a continuous mesh generated using the algorithm created by Markosian [2].

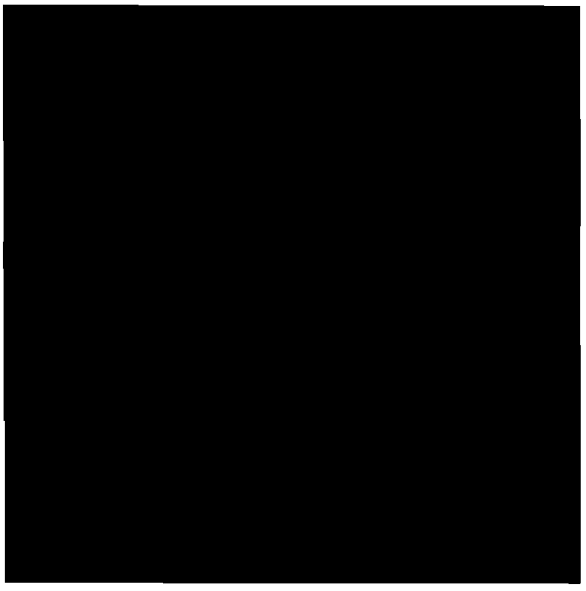


Figure 3.1: The single skeleton model with skeleton and sampling

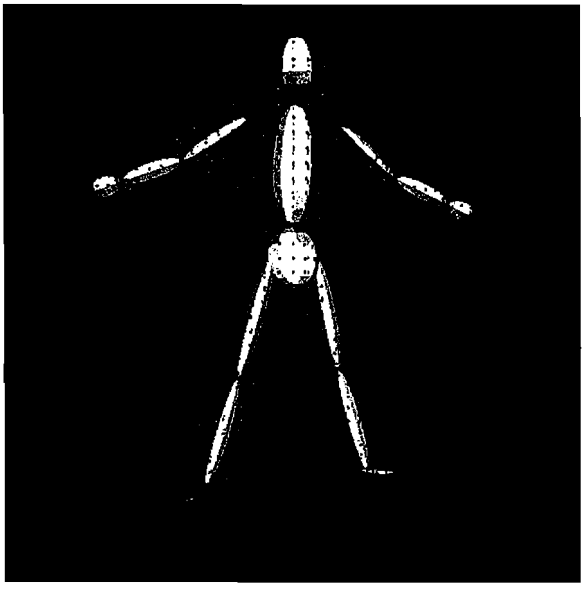


Figure 3.2: The full character model with skeletons and sampling

3.3 Sampling and Nodal Point Placement

To generate the mass spring system, we first create an axis-aligned bounding box around the model and sample points within the bounding box. The sample points are chosen such that they form a grid with cells of approximately equal volume. To determine the spacing between the sample points, we first identify the shortest side of the bounding box, and find the spacing distance of that axis based on an user-defined sampling rate. This *sampling distance* is then used on the other two sides of the bounding box to determine the number of samples required for each side.

For each sample point inside the bounding box, we choose a random infinite ray anchored to the sample point. If this vector intersects an odd number of polygons on the skeleton mesh, the sample point is identified to be inside the skeleton and categorized as a *SkeletonPoint.* However, if the number of intersections is even, we perform the same intersection check again but using the skin mesh. The sample point is determined to be inside the skin mesh and categorized as a *VolumePoint* if the number of intersections is odd. In the full-character model where there are 19 disjoint skeleton meshes, The SkeletonPoint test is done with each skeleton mesh iteratively tested until either the sample point is assigned to a particular skeleton, or the sample point has been tested against all skeletons and found not to be inside any of the skeletons.

For the purpose of speed and stability which will be discussed in the later chapters, the skin in the full-character model needs to be divided into 19 separate body parts associated with each of the 19 skeletons (Figure (3.3)). Determining which body part a VolumePoint or SurfacePoint belongs to is similar to that of decomposing the skin as a continuous mesh into one or more body parts. The points are tested during an initialization step to find out which skeleton they are closest to and then assigned to the body part that the skeleton represents. After all the points have been assigned, each point then checks to see if its neighboring points are associated with the same body part. If any of its neighbors belong to a different part, the point computes the *percentage* of how much it belongs to each body part using the following equation.

$$c_i = \frac{\frac{1}{l_i}}{\sum_{i=0}^{n} \frac{1}{l_i}} \tag{3.1}$$

where c denotes the percentage, i represents all the neighboring body parts of the tested point, and l_i represents the length from the tested point to the closest point on body part i.

Figure 3.3: Different body parts of the human character including the over-lapped sections

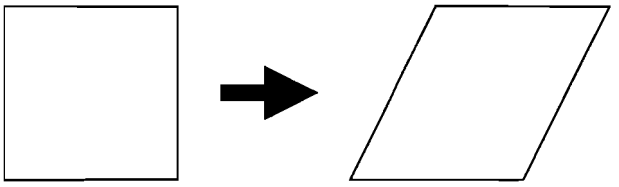


Figure 3.4: A configuration with no diagonal support

3.4 Spring Creation

Once the nodal points are established, we connect them using regular springs that have a user-defined spring constant and a damping constant. Each spring has its own rest length, and follows the standard spring equation:

$$\mathbf{f}_i = k(|\mathbf{x}_j - \mathbf{x}_i| - r)\frac{\mathbf{x}_j - \mathbf{x}_i}{|\mathbf{x}_j - \mathbf{x}_i|} - d(\mathbf{v}_i - \mathbf{v}_j) \tag{3.2}$$

where $\mathbf{f}_i$ denotes the total force exerted on point i, k the spring constant, d the damping constant, r the rest length, $\mathbf{x}_i$ and $\mathbf{x}_j$ the positions of the two points connected by the spring, and $\mathbf{v}_i$ and $\mathbf{v}_j$ the velocities of the two points.

Choosing the correct nodal points and connecting them with springs will determine the *stiffness* and structural stability of the system. For example, the configuration in Figure (3.4) gives no diagonal support to the structure, therefore the structure could easily collapse

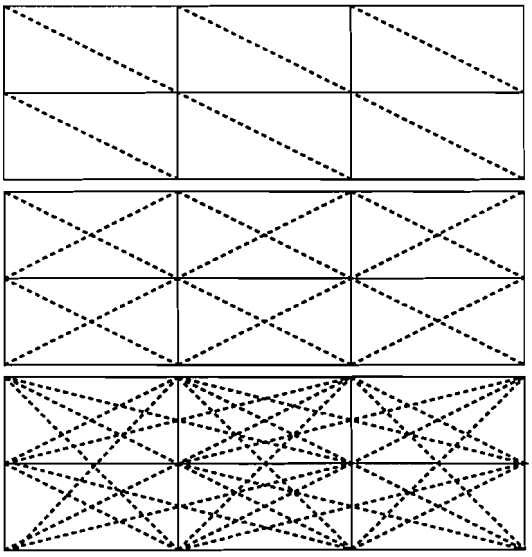


Figure 3.5: Some configurations with diagonal support

without incurring forces. Figure (3.5) demonstrates some different connectivities that will add angular, or diagonal, support that prevents the structure from collapsing. Since each of these configurations is *correct*, choosing the simplest form with the least number of springs will decrease the amount of computation required.

For the single skeleton case, we connected our 3D mesh similar to that of the second diagram in figure (3.5). Each nodal point in this case is connected with 15 springs. After several experiments, we discovered that in $2D$, breaking up rectangles into triangles would give angular support to the structure efficiently. Therefore, in our full character model, we inferred that the same reasoning would apply to $3D$, and broke up the cubes made up of nodal points into tetrahedrons (Figure (3.6)).

3.5 Undersampling

Using the heuristic for nodal point placement described above creates an aliasing effect. The problem becomes even more obvious in cases where the user-defined sampling rate is low. At places where the skeleton is *skinny*, there could exist only one or sometimes no nodal point to represent the skeleton. Similarly, the skin could experience the same problem where the finer parts are undersampled. This undersampling problem is currently a research area in Computational Geometry with no clear solution [3] [4].

We attempt to alleviate the undersampling problem by *moving* nodal points onto the

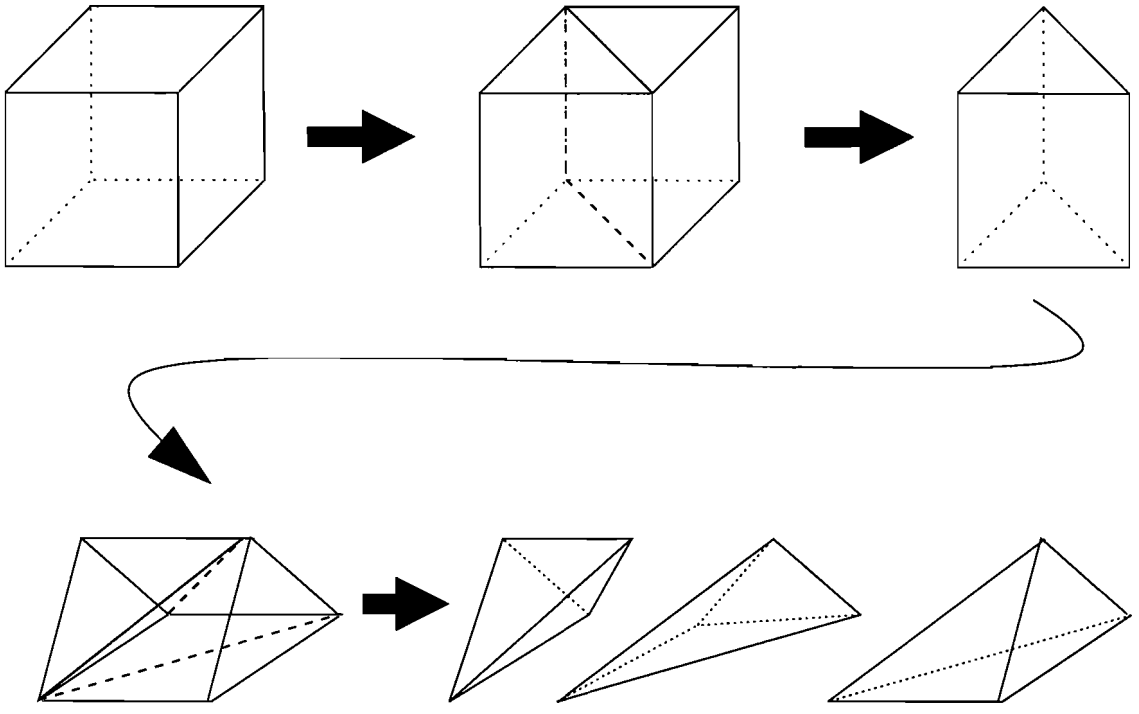


Figure 3.6: (a) Breaking up a cube into tetrahedrons (b) Cutting the cube diagonally (c) Due to symmetry, only one part has to be examined (d) Rotating and cutting (e) The resulting tetrahedrons from the cuts

surface of the skeletons and the skin. The *moving* process takes place after the SkeletonPoints and the VolumePoints have been determined. To better represent the skeletons, each VolumePoint finds out its distance to the closest skeleton. If any of the x, y, z components of this distance vector are less than the sampling distance as described in Section 3.3, the VolumePoint is determined to be *close enough* to the skeleton, and moves onto the skeleton to become a SkeletonPoint.

In dealing with the aliasing effect along the skin, we first determine all the VolumePoints that have neighbors outside of the boundary. We then find all the grid points that are directly adjacent to these points but lie outside of the skin. Each of these newly found grid points is tested for a closest point on the skin and then moved onto the skin to become a *SurfacePoint.*

3.6 Spring Elimination

Since the initial sampling was done along a grid, the springs forming the tetrahedrons could easily connect nodal points between different body parts at places where springs might be undesirable as shown in figure (3.7). The problem becomes more obvious as SurfacePoints are created and moved onto the skin because many of these SurfacePoints that were originally next to each other on the grid now become much further apart. To eliminate this problem, we iteratively check the length of each spring after the SurfacePoints have been created. If any of the x, y, or z component of the length vector exceeds the sampling distance, the spring is discarded (Figure (3.8)).

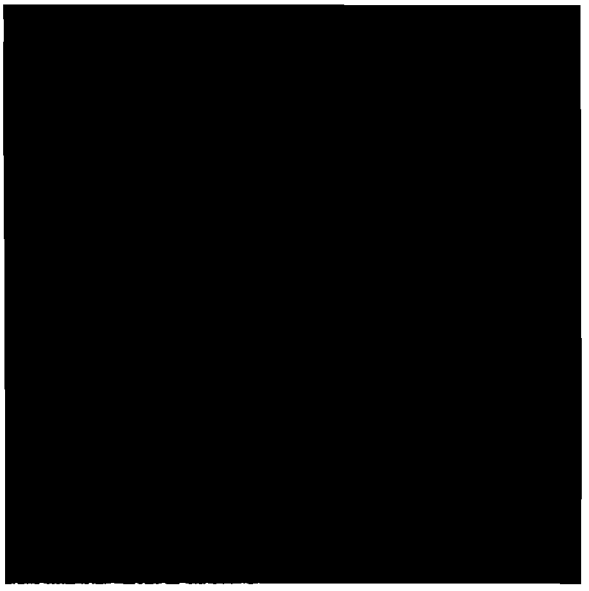


Figure 3.7: Original configuration without cutting any springs

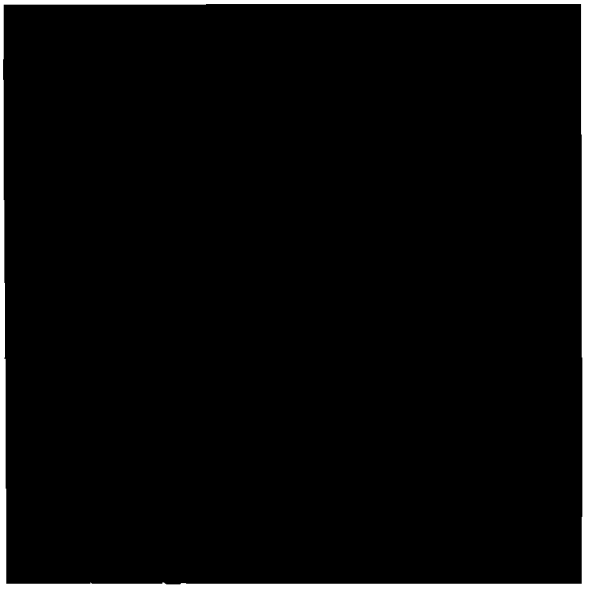


Figure 3.8: After spring elimination

Chapter 4

Animating The Skeleton

The primary difference between SkeletonPoints, VolumePoints, and SurfacePoints is the fact that SkeletonPoints are attached to skeletons, and therefore not subject to the effect of springs. When the skeleton is translated or rotated due to applied forces (refer to the next section for more detail), the SkeletonPoints undergo the same amount of transformation. The movement of SkeletonPoints in turn stretches or compresses the springs attached. These springs will then pull or push on the VolumePoints or SurfacePoints, causing the entire mass spring system to deform.

4.1 Applied Forces

With the mass spring system in place, creating the animation is as simple as moving the skeletons and simulating the nodal points. In the simpler model where there exists only one skeleton, keyframed information on the skeleton is defined within a text file that the system parses during the initialization process. In the full character model, we use motion capture data as input. The motion capture data specifies both position and orientation of each skeleton during each frame of animation. Our motion capture data was collected at 30 frames per second, therefore we linearly interpolate between frames when the simulation is running at a higher frame count. Since the translational data is given in x, y, z format in relation to the origin, and the rotational data is given in degrees of rotation around each axis, the interpolation is nothing more than

$$\rho(t + c\Delta t) = \rho(t)c + \rho(t + \Delta t)(1 - c)$$

where ρ is the translational vector, and c is the percentage of the desired interpolated time in relation to Δt. Similarly,

$$\theta(t + c\Delta t) = \theta(t)c + \theta(t + \Delta t)(1 - c)$$

where θ denotes the vector that contains the Euler angles of rotation.

Chapter 5

Deformation Models

This chapter describes the mass spring model, the reduced degree of freedom model, and the implicit integration model. Because these are volume deformation techniques, each model is associated with a skin deformation process that is separate from its numerical integration process.

5.1 Mass Spring Model

In the single skeleton case, the effects of applying forces to the skeleton create rigid transformations such as rotation or translation. The skeleton follows Newton's law of motion when it is subjected to external forces.

$$\mathbf{a}(t) = \frac{\mathbf{f}(t)}{m}$$

$$\mathbf{v}(t + \Delta t) = \mathbf{v}(t) + \mathbf{a}(t)\Delta t$$

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \mathbf{v}(t)\Delta t + \frac{1}{2}\mathbf{a}(t)\Delta t^2$$

where $\mathbf{f}$ is the external force, m is the mass, $\mathbf{a}$ is the acceleration, $\mathbf{v}$ is the velocity, $\mathbf{x}$ is the position, t is the elapsed time, and Δt is the time step.

For orientation, on top of the initial orientation of the skeleton, we keyframe the additional degree of rotation around each axis during each frame of animation. The skeleton's subsequent orientation is computed by adding these additional degree of rotation onto its existing orientation.

As mentioned in the previous section, moving the skeleton causes the SkeletonPoints to move as well, which in turn changes the lengths of the springs. When a spring is not at its rest length, it generates forces on the two nodal points connected to it. The spring's original rest length, its spring constant, damping constant, and the relative velocities of the two nodal points determine the amount of forces as shown in equation (3.2).

The forces cause acceleration on the VolumePoints. Using the same motion equations described above for skeletons, each VolumePoint arrives at a new state with a new acceleration, velocity, and position for the next time step.

5.1.1 Rendering

One way to render the mesh using a mass spring model is to associate the vertices on the mesh with certain nodal points in the mass spring system. When these nodal points change their positions during the deformation process, the vertices would move accordingly. In our single skeleton case, because the skin is nothing more than a geometric cube, we easily pick nodal points that lie exactly on top of the skin and assign them as vertices of the mesh. After each iteration of the simulation, the rendering engine draws the mesh based on the calculated positions of these particular nodal points (Figure (3.1)). Because the VolumePoints are point masses, orientation is omitted.

5.2 Reduced Degree of Freedom Model

The reduced degree of freedom (reduced DOF) model limits the deformation process using nth order deformation functions. We adopted Witkin and Welch's algorithm [30] where we characterize the deformation by a map from $\Re^3$ to $\Re^3$ while adding time dependence for the purpose of animation. Specifically, if $\tilde{\mathbf{x}}_i$ is the location of a point on the undeformed object, the location of the point on the deformed object is

$$\mathbf{x} = R\mathbf{p}_i \tag{5.1}$$

where

$$\mathbf{p}_i = w(\tilde{\mathbf{x}}_i)$$

The function w converts an undeformed position in 1st order to nth order coordinate. It does not depend on R or time. In our implementation, we chose $w(\tilde{\mathbf{x}}_i) = w(\tilde{x}_i, \tilde{y}_i, \tilde{z}_i)$ to be $[\tilde{x}_i, \tilde{y}_i, \tilde{z}_i, \tilde{x}_i\tilde{y}_i, \tilde{y}_i\tilde{z}_i, \tilde{x}_i\tilde{z}_i, \tilde{x}_i^2, \tilde{y}_i^2, \tilde{z}_i^2, 1]$. The 3×10 matrix R transforms the undeformed $\tilde{\mathbf{x}}_\mathbf{i}$ into the deformed position $\mathbf{x}_i$. R is defined to be

$$R(t+\Delta t) = R(t) + \dot{R}(t)\Delta t + \frac{1}{2}\ddot{R}(t)\Delta t^2$$

and $\dot{R}(t)$ is defined to be

$$\dot{R}(t+\Delta t) = \dot{R}(t) + \ddot{R}(t)\Delta t$$

and $\ddot{R}(t)$ is equal to

$$\ddot{R}(t) = Q(t)M^{-1}$$

In order to compute $\ddot{R}$, we first compute the constant symmetric mass matrix M

$$M = \sum_i (m\mathbf{p}_i\mathbf{p}_i^T)$$

where i represents all the mass points in the system. The generalized force Q given the force $\mathbf{f}_i$ applied to the world-space point $\mathbf{x}_i = R\mathbf{p}_j$ is defined as

$$Q(t) = \sum_i \mathbf{f}_i(t)\mathbf{p}_i^T$$

Because R is a map between deformed and undeformed object, the mass matrix M only needs to be computed and inverted once. In other words, the set of undeformed points are computed once during the initialization step, and used in equation (5.1) at the end of each time step of the simulation. For the full derivation of these equations, refer to [30].

In the full character model, where the deformation is much more complex, a 3×10 2nd order deformation matrix is no longer sufficient to describe the deformation process of the entire character. Therefore we break down the character into separate body parts where each part is described by its own 3×10 deformation matrix. To ensure the continuity between body parts, the position of each nodal point between body parts is computed:

$$\mathbf{x}_i = \sum_{b=0}^{n} c_{b,i}(R_b\mathbf{p}_i) \tag{5.2}$$

where $c_{b,i}$ is the percentage computed during initialization described in equation (3.1) for body part b and node i. Similarly, R_b denotes the deformation matrix for body part b. In this equation, we define the sum of two points as the sum of each x, y, z component of the points' coordinates.

5.2.1 Rendering

Since the reduced DOF model is a space-warping technique, the rendering process requires no extra algorithm. The vertices on the skin are deformed using equation (5.1) in the single skeleton case, and equation (5.2) in the full-character model after the deformation matrices are computed for each body part.

5.3 Implicit Integration and Finite Element Method using Approximate Continuum Model

Although Finite Element Methods (FEM) have traditionally been slow, they are still used often for their stability and accuracy. The dynamic FEM equation is usually represented as

$$F = M\ddot{U} + C\dot{U} + KU$$

where M, C, K are the mass, damping, and stiffness matrices respectively for the entire object. F is the composite vector of equivalent applied forces, and U is the composite vector of node displacements.

We have chosen the approach described by Baraff and Witkin [1]. In their algorithm, they utilize implicit integration for solving the stiff differential equation described above. The implicit integration technique requires the use of the *backward Euler* method. Similar to the traditional *forward Euler* method, both approaches attempt to solve for a new position $\mathbf{x}(t+\Delta t)$ and a new velocity $\mathbf{v}(t+\Delta t)$ given the current position $\mathbf{x}(t)$ and velocity $\mathbf{v}(t)$. To compute the new state and velocity using an implicit technique, we first assume that the change of position is the same as velocity, and write the differential equation as

$$\frac{d}{dt}\begin{pmatrix}\mathbf{x}\\ \mathbf{v}\end{pmatrix} = \begin{pmatrix}\mathbf{v}\\ M^{-1}f(\mathbf{x},\mathbf{v})\end{pmatrix} \tag{5.3}$$

where M^{-1} is the inverse of the mass matrix, and $f(\mathbf{x},\mathbf{v})$ denotes the applied force. To simplify notation, we define $\mathbf{x}_0 = \mathbf{x}(t_0)$, and $\mathbf{v}_0 = \mathbf{v}(t_0)$. We also define $\Delta\mathbf{x} = \mathbf{x}(t_0+\Delta t) - \mathbf{x}(t_0)$, and $\Delta\mathbf{v} = \mathbf{v}(t_0+\Delta t) - \mathbf{v}(t_0)$.

The Forward Euler method applied to equation (5.3) approximates $\Delta\mathbf{x}$ and $\Delta\mathbf{v}$ as

$$\begin{pmatrix}\Delta\mathbf{x}\\ \Delta\mathbf{v}\end{pmatrix} = \Delta t\begin{pmatrix}\mathbf{v}_0\\ M^{-1}\mathbf{f}_0\end{pmatrix}$$

where $\mathbf{f}_0$ is defined to be $f(\mathbf{x}_0, \mathbf{v}_0)$. This approximation requires that Δt be quite small to ensure stability of the simulation. The backward Euler method appears similar, where $\Delta\mathbf{x}$ and $\Delta\mathbf{v}$ are approximated as

$$\begin{pmatrix} \Delta\mathbf{x} \\ \Delta\mathbf{v} \end{pmatrix} = \Delta t \begin{pmatrix} \mathbf{v}_0 + \Delta\mathbf{v} \\ M^{-1} f(\mathbf{x}_0 + \Delta\mathbf{x}, \mathbf{v}_0 + \Delta\mathbf{v}) \end{pmatrix} \tag{5.4}$$

The primary difference between the forward and backward Euler methods is that forward Euler is based solely on conditions at time t_0 while backward Euler starts from the output state $(\mathbf{x}_0 + \Delta\mathbf{x}, \mathbf{v}_0 + \Delta\mathbf{v})$ and uses a forward Euler step to run the system backward in time to get to the state $(\mathbf{x}_0, \mathbf{v}_0)$:

$$\begin{pmatrix} \mathbf{x}_0 \\ \mathbf{v}_0 \end{pmatrix} = \begin{pmatrix} \mathbf{x}_0 + \Delta\mathbf{x} \\ \mathbf{v}_0 + \Delta\mathbf{v} \end{pmatrix} - \Delta t \begin{pmatrix} (\mathbf{v}_0 + \Delta\mathbf{v}) \\ M^{-1} f(\mathbf{x}_0 + \Delta\mathbf{x}, \mathbf{v}_0 + \Delta\mathbf{v}) \end{pmatrix}$$

To find $\Delta\mathbf{x}$ and $\Delta\mathbf{v}$ requires that we solve for values of $\Delta\mathbf{x}$ and $\Delta\mathbf{v}$ that satisfy equation (5.4). But rather than solving this nonlinear equation iteratively, we approximate the function f using a first order Taylor series

$$f(\mathbf{x}_0 + \Delta\mathbf{x}, \mathbf{v}_0 + \Delta\mathbf{v}) = \mathbf{f}_0 + \frac{\partial f}{\partial x}\Delta\mathbf{x} + \frac{\partial f}{\partial v}\Delta\mathbf{v} \tag{5.5}$$

In this equation, $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial v}$ are Jacobian matrices describing the relationship between force, position, and velocity respectively. However, because we only need to compute the new states for *movable* points, i.e. VolumePoints and SurfacePoints, $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial v}$ has to be broken down into four matrices:

$$\frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f_m}{\partial x_m} & \frac{\partial f_m}{\partial x_s} \\ \frac{\partial f_s}{\partial x_m} & \frac{\partial f_s}{\partial x_s} \end{bmatrix}$$

$$\frac{\partial f}{\partial v} = \begin{bmatrix} \frac{\partial f_m}{\partial v_m} & \frac{\partial f_m}{\partial v_s} \\ \frac{\partial f_s}{\partial v_m} & \frac{\partial f_s}{\partial v_s} \end{bmatrix}$$

where x_m and v_m denote the position and velocity of movable points, x_s and v_s denote those of the SkeletonPoints, f_m denotes the force applied on movable points, and f_s denotes the force applied on SkeletonPoints. We can safely ignore $\frac{\partial f_s}{\partial x_s}$ and $\frac{\partial f_s}{\partial v_s}$ because the SkeletonPoints are moved rigidly and do not exert forces on themselves. $\frac{\partial f_s}{\partial x_m}$ and $\frac{\partial f_s}{\partial v_m}$ denote the amount of forces acting on the SkeletonPoints due to the change in positions or velocities of the *movable* points. Although these forces do exist, we are ignoring them because we would

like our SkeletonPoints to be transformed by the motion capture data, not dynamically simulated.

We can decompose $\Delta\mathbf{x}$ and $\Delta\mathbf{v}$:

$$\Delta\mathbf{x} = \begin{bmatrix} \Delta\mathbf{x}_m \\ \Delta\mathbf{x}_s \end{bmatrix}$$

and

$$\Delta\mathbf{v} = \begin{bmatrix} \Delta\mathbf{v}_m \\ \Delta\mathbf{v}_s \end{bmatrix}$$

In order to segregate the *movable* and *non-movable* points, we rewrite equation (5.5) as

$$f(\mathbf{x}_0 + \Delta\mathbf{x}, \mathbf{v}_0 + \Delta\mathbf{v}) = \mathbf{f}_0 + \frac{\partial f_m}{\partial x_m}\Delta\mathbf{x_m} + \frac{\partial f_m}{\partial v_m}\Delta\mathbf{v_m} + \frac{\partial f_m}{\partial x_s}\Delta\mathbf{x}_s + \frac{\partial f_m}{\partial v_s}\mathbf{v}_s \tag{5.6}$$

Notice that this equation is not the mathematically equivalent of equation (5.5) because we are omitting the terms $\frac{\partial f_s}{\partial x_m}$ and $\frac{\partial f_s}{\partial v_m}$. We choose this approximation in order to achieve our desired effects.

The first unknown in equation (5.4) is the vector $\mathbf{f}_0$, which represents the total amount of force acting on each nodal point at time t_0. Because VolumePoints and SurfacePoints' position displacement and change in velocity between t_0 and $t_0 + \Delta t$ depend strictly on the springs, we can write the force equation for $\mathbf{f}_0$ as

$$\mathbf{f}_{0,j} = \sum_{i=1}^{n} k_{i,j}(|\mathbf{x}_i - \mathbf{x}_j| - r_{i,j})\frac{\mathbf{x}_i - \mathbf{x}_j}{|\mathbf{x}_i - \mathbf{x}_j|} + \sum_{i=1}^{n} d_{i,j}(\mathbf{v}_i - \mathbf{v}_j)$$

where i iterates through all springs attached to the nodal point. $r_{i,j}$ represents the rest length of the spring that connects the two points i and j, $k_{i,j}$ the spring constant, $d_{i,j}$ the damping constant, $\mathbf{v}_i$ and $\mathbf{v}_j$ the velocities of the two nodal points connected by the spring i, j, and $\mathbf{x}_i$ and $\mathbf{x}_j$ the positions of the two nodal points.

To compute the elements in $\frac{\partial f_m}{\partial x_m}$ of equation (5.6), we differentiate the force equation with respect to the position of each nodal point.

$$\frac{\partial f_m}{\partial x_m} = \begin{bmatrix} \frac{\partial f_{1,x}}{\partial x_{1,x}} & \frac{\partial f_{1,x}}{\partial x_{1,y}} & \frac{\partial f_{1,x}}{\partial x_{1,z}} & \frac{\partial f_{1,x}}{\partial x_{2,x}} & \frac{\partial f_{1,x}}{\partial x_{2,y}} & \cdots \\ \frac{\partial f_{1,y}}{\partial x_{1,x}} & \frac{\partial f_{1,y}}{\partial x_{1,y}} & \frac{\partial f_{1,y}}{\partial x_{1,z}} & \frac{\partial f_{1,y}}{\partial x_{2,x}} & \frac{\partial f_{1,y}}{\partial x_{2,y}} & \cdots \\ \frac{\partial f_{1,z}}{\partial x_{1,x}} & \frac{\partial f_{1,z}}{\partial x_{1,y}} & \frac{\partial f_{1,z}}{\partial x_{1,z}} & \frac{\partial f_{1,z}}{\partial x_{2,x}} & \frac{\partial f_{1,z}}{\partial x_{2,y}} & \cdots \\ \frac{\partial f_{2,x}}{\partial x_{1,x}} & \frac{\partial f_{2,x}}{\partial x_{1,y}} & \frac{\partial f_{2,x}}{\partial x_{1,z}} & \frac{\partial f_{2,x}}{\partial x_{2,x}} & \frac{\partial f_{2,x}}{\partial x_{2,y}} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

$$\frac{\partial f_{a,x}}{\partial x_{a,x}} = \sum_{i=1}^{n} k_{ai}(r_{ai}\frac{1}{\sqrt{h}} - \frac{(x_i - x_a)^2}{\sqrt{h^3}} - 1)$$

$$\frac{\partial f_{a,x}}{\partial x_{a,y}} = -\sum_{i=1}^{n} k_{ai}(r_{ai}\frac{(x_i - x_a)(y_i - y_a)}{\sqrt{h^3}})$$

$$\frac{\partial f_{a,x}}{\partial x_{i,x}} = k_{ai}(r_{ai}\frac{(x_a - x_i)^2}{\sqrt{h^3}} - \frac{1}{\sqrt{h}} + 1) \qquad i \neq a$$

$$\frac{\partial f_{a,x}}{\partial x_{i,y}} = k_{ai}(r_{ai}\frac{(x_a - x_i)(y_a - y_b)}{\sqrt{h^3}}) \qquad i \neq a$$

where i iterates through all of nodal point a's neighboring nodal points. k_{ai} represents the the spring constant of the spring connecting the nodal point a and i, and r_{ai} the length of that spring. h represents the current spring length, and is calculated as

$$h = (x_a - x_i)^2 + (y_a - y_i)^2 + (z_a - z_i)^2$$

$\frac{\partial f_m}{\partial v_m}$ is derived similarly

$$\frac{\partial f_{a,x}}{\partial v_{a,x}} = \sum_{i=1}^{n} d_{ai}$$

$$\frac{\partial f_{a,x}}{\partial v_{a,y}} = 0$$

$$\frac{\partial f_{a,x}}{\partial v_{b,x}} = d_{ab}$$

$$\frac{\partial f_{a,x}}{\partial v_{b,y}} = 0$$

where d_{ab} is the damping constant of the spring between the two nodal points a, and b.

If a nodal point a is not directly connected to another nodal point b, the x, y, z, entries of $\frac{\partial f_a}{\partial x_a}$, $\frac{\partial f_a}{\partial x_b}$, $\frac{\partial f_a}{\partial v_a}$ and $\frac{\partial f_b}{\partial v_b}$ would all be zeros. In our system, the highest connectivity of any nodal point is 23, which makes both $\frac{\partial f_m}{\partial x_m}$ and $\frac{\partial f_m}{\partial v_m}$ symmetric and sparse matrices especially when the number of nodal points reaches 2,000 or above (as in the experiments described in the results section) and the matrices have the dimension of approximately $6,000 \times 6,000$.

The last unknowns in equation (5.6) are terms including SkeletonPoints. $\Delta\mathbf{x}_s$ and $\Delta\mathbf{v}_s$ are vectors representing the change in position and velocity of the SkeletonPoints. Computing $\frac{\partial f_m}{\partial x_s}$ and $\frac{\partial f_m}{\partial v_s}$ is similar to computing $\frac{\partial f_m}{\partial x_m}$ and $\frac{\partial f_m}{\partial v_m}$ except that $\frac{\partial f_m}{\partial x_s}$ and $\frac{\partial f_m}{\partial v_s}$ are not symmetric matrices. They have the dimension of $3n \times 3s$ where s is the number of SkeletonPoints and n is the number of movable points.

Substituting the force equation equation (5.6) into equation (5.4) yields:

$$\begin{pmatrix} \Delta\mathbf{x}_m \\ \Delta\mathbf{v}_m \end{pmatrix} = \Delta t \begin{pmatrix} \mathbf{v}_{0,m} + \Delta\mathbf{v}_m \\ M^{-1}(\mathbf{f}_0 + \frac{\partial f_m}{\partial x_m}\Delta\mathbf{x}_m + \frac{\partial f_m}{\partial v_m}\Delta\mathbf{v}_m + \frac{\partial f_m}{\partial x_s}\Delta\mathbf{x}_s + \frac{\partial f_m}{\partial v_s}\Delta\mathbf{v}_s) \end{pmatrix}$$

Taking the bottom row of this equation and substituting $\Delta\mathbf{x}_m$ with the top row of the same equation yields:

$$\Delta\mathbf{v}_m = \Delta t\, M^{-1}(\mathbf{f}_0 + \Delta t\frac{\partial f_m}{\partial x_m}(\mathbf{v}_{0,m} + \Delta\mathbf{v}_m) + \frac{\partial f_m}{\partial v_m}\Delta\mathbf{v}_m + \frac{\partial f_m}{\partial x_s}\Delta\mathbf{x}_s + \frac{\partial f_m}{\partial v_s}\Delta\mathbf{v}_s)$$

Let I denote the identity matrix, and regrouping:

$$(I - \Delta t M^{-1}\frac{\partial f_m}{\partial v_m} - \Delta t^2 M^{-1}\frac{\partial f_m}{\partial x_m})\Delta\mathbf{v}_m = \Delta t M^{-1}(\mathbf{f}_0 + \Delta t\frac{\partial f_m}{\partial x_m}\mathbf{v}_{0,m} + \frac{\partial f_m}{\partial x_s}\Delta\mathbf{x}_s + \frac{\partial f_m}{\partial v_s}\Delta\mathbf{v}_s)$$

In our implementation, every nodal point has unit mass, therefore, our mass matrix M is just the identity matrix. Substituting M^{-1} with I yields:

$$(I - \Delta t\frac{\partial f_m}{\partial v_m} - \Delta t^2\frac{\partial f_m}{\partial x_m})\Delta\mathbf{v}_m = \Delta t(\mathbf{f}_0 + \Delta t\frac{\partial f_m}{\partial x_m}\mathbf{v}_{0,m} + \frac{\partial f_m}{\partial x_s}\Delta\mathbf{x}_s + \frac{\partial f_m}{\partial v_s}\Delta\mathbf{v}_s)$$

which we then solve for $\Delta\mathbf{v}_m$ using the bi-conjugate gradient method described below. Given $\Delta\mathbf{v}_m$, we then trivially compute

$$\Delta\mathbf{x}_m = \Delta t(\mathbf{v}_m + \Delta\mathbf{v}_m)$$

5.3.1 Bi-Conjugate Gradient

If we substitute

A = left hand side of equation (5.3), and b = right hand side of the same equation, and $\mathbf{x}$ with $\Delta\mathbf{v}_m$, we arrive at the form

$$A\mathbf{x} = \mathbf{b}$$

where A denotes a matrix, $\mathbf{b}$ a known vector, and $\mathbf{x}$ the unknown vector. Instead of solving for $\mathbf{x}$ by inverting A, which is an expensive process, especially when the number of nodal points is large, we can use approximation algorithms. The bi-conjugate gradient function we adopted from [5] is a steepest decent method, and solves for the unknown $\mathbf{x}$ in a fraction of time compared to inverting matrices.

5.3.2 Memory Issue

The matrices $\frac{\partial f_m}{\partial v_m}$ and $\frac{\partial f_m}{\partial x_m}$ each has the dimension $3n \times 3n$. In our simulation, the number of nodal points n could range upwards to approximately 40,000. Assuming that the matrices are comprised of *doubles,* each matrix would require 56 gigabytes of memory. Thankfully, because these matrices are both sparse and symmetric, we implemented some data structures to store only the un-repeated non-zero elements. Although this compression slows down the computation due to the searches and retrievals required by these data structures, the trade off between speed and memory is necessary because of the limitation of hardware.

5.3.3 Speed-Ups

There are a few notable speedups that decrease processing time quite significantly. First of all, since the matrices $\frac{\partial f_m}{\partial x_m}$ and $\frac{\partial f_m}{\partial v_m}$ are symmetric, the matrix multiplication process is approximately halved. Second of all, $\frac{\partial f_m}{\partial v_m}$ and $\frac{\partial f_m}{\partial v_s}$ depend solely on the damping constant, which remains unchanged throughout the simulation. Therefore, $\frac{\partial f_m}{\partial v_m}$ and $\frac{\partial f_m}{\partial v_s}$ only need to be computed once during pre-processing.

Lastly, by breaking down the full character model into different body parts, we gain dramatically in both speed and memory. For example, assume that the character is comprised of 100 nodal points, and there are 10 body parts each of 10 nodal points, computing the character as a whole would require the $\frac{\partial f_m}{\partial v_m}$ and $\frac{\partial f_m}{\partial x_m}$ matrices be of size $300 \times 300 = 90,000$. However, if we compute each individual body part separately, the system would compute 10 matrices each of size $30 \times 30 = 900$, totaling 9,000, a savings of a factor of 10.

The downside to this optimization is that the effect generated by one nodal point's change in displacement or velocity does not immediately affect the entire system. Instead, it would require one extra time step for the effect to cross between boundaries of body parts. In other words, by moving the nodal points in the right hand of the character would require 7 time steps for the effect to travel through the upper arm, lower arm, torso, pelvis, upper leg, lower leg, and into the right foot. However, if the character is computed as a whole, the same move to the nodal point in the right hand would require only 1 time step to achieve the same effect.

5.3.4 Rendering

Unlike the reduced DOF model, FEM does not warp space. Therefore the rendering step requires the use of a mesh deformation technique such as tri-linear interpolation or low order deformation. Tri-linear interpolation requires that each vertex on the skin be enclosed in a

set of movable points that define a volume. Then the position of the vertex is parameterized within the volume and defined as a (u, v, w) Barycentric coordinate. When the positions of the movable points change, the position of the vertex would change with them but stay in the same (u, v, w) position within the volume.

We believe that using trilinear interpolation would create very high frequency deformation causing the possibility of self penetration or other artifacts. For the purpose of a pleasing and continuous mesh, we chose to use the low order deformation technique where we first collect the movements of all the nodal points in the system. Using that information, we build a function that describes the deformation process as follows

$$X(t + \Delta t) = p(\tilde{\mathbf{x}})\, R \tag{5.7}$$

where $X(t + \Delta t)$ is a $n \times 3$ matrix containing all deformed positions (in x, y, z) of all the nodal points:

$$X(t + \Delta t) = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ \vdots & \vdots & \vdots \end{bmatrix}$$

and $p(\tilde{\mathbf{x}})$ has the form of

$$\begin{pmatrix} \tilde{\mathbf{x}}_{1_x} & \tilde{\mathbf{x}}_{1_y} & \tilde{\mathbf{x}}_{1_z} & \tilde{\mathbf{x}}_{1_{xy}} & \tilde{\mathbf{x}}_{1_{yz}} & \tilde{\mathbf{x}}_{1_{xz}} & \tilde{\mathbf{x}}_{1_{x^2}} & \tilde{\mathbf{x}}_{1_{y^2}} & \tilde{\mathbf{x}}_{1_{z^2}} & 1 \\ \tilde{\mathbf{x}}_{2_x} & \tilde{\mathbf{x}}_{2_y} & \tilde{\mathbf{x}}_{2_z} & \tilde{\mathbf{x}}_{2_{xy}} & \tilde{\mathbf{x}}_{2_{yz}} & \tilde{\mathbf{x}}_{2_{xz}} & \tilde{\mathbf{x}}_{2_{x^2}} & \tilde{\mathbf{x}}_{2_{y^2}} & \tilde{\mathbf{x}}_{2_{z^2}} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

$p(\tilde{\mathbf{x}})$ is a $n \times 10$ matrix where n is the number of nodal points, and R is the deformation matrix. Assuming that the deformation is described with a 2nd order function, R becomes a 10×3 matrix.

We rewrite equation (5.7) as

$$\mathbf{x}(t + \Delta t) = p'(\tilde{\mathbf{x}}) * \mathbf{r} \tag{5.8}$$

where $\mathbf{x}(t + \Delta t)$ is a vector of size $3n \times 1$, $p'(\tilde{\mathbf{x}})$ is a matrix of size $3n \times 30$, and $\mathbf{r}$ is a vector of size 30×1. Transforming $\mathbf{x}(t + \Delta t)$ from $X(t + \Delta t)$ requires putting all $3n$ elements of $X(t + \Delta t)$ into a vector. Similarly, transforming $\mathbf{r}$ from R requires putting 30 elements in R into the vector $\mathbf{r}$. In order to multiply $p'(\tilde{\mathbf{x}})$ to a vector of size 30×1 resulting in another vector of size $3n \times 1$, $p'(\tilde{\mathbf{x}})$ must have the dimension of $3n \times 30$. Padding $p'(\tilde{\mathbf{x}})$ from a $n \times 10$ matrix into a $3n \times 30$ matrix, we add zeros into the original matrix as follows:

$$\begin{pmatrix} \tilde{\mathbf{x}}_{1_x} & 0 & 0 & \tilde{\mathbf{x}}_{1_y} & 0 & 0 & \vdots \\ 0 & \tilde{\mathbf{x}}_{1_x} & 0 & 0 & \tilde{\mathbf{x}}_{1_y} & 0 & \vdots \\ 0 & 0 & \tilde{\mathbf{x}}_{1_x} & 0 & 0 & \tilde{\mathbf{x}}_{1_y} & \vdots \\ \tilde{\mathbf{x}}_{2_x} & 0 & 0 & \tilde{\mathbf{x}}_{2_y} & 0 & 0 & \vdots \\ 0 & \tilde{\mathbf{x}}_{2_x} & 0 & 0 & \tilde{\mathbf{x}}_{2_y} & 0 & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

Equation (5.8) becomes a fitting problem where there are $3n$ number of equations but 30 unknowns. We use the least square fit approach to find $\mathbf{r}$.

$$\mathbf{r} = (p'(\tilde{\mathbf{x}})^T p'(\tilde{\mathbf{x}}))^{-1} p'(\tilde{\mathbf{x}})\, x(t + \Delta t)$$

Once the 30×1 vector $\mathbf{r}$ is found, we rewrite it as a 3×10 matrix R again, and substitute it into equation (5.1) to solve for the new positions of the vertices.

Chapter 6

Results

We tested the three systems: regular mass spring system, reduced DOF system, and the implicit integration system as described in Chapter 5 using different spring constants, damping constants, forces, time steps, and sample rates. The results are plotted with time on the x-axis, and *total deformation* on the y-axis. Total deformation is calculated as

$$d = \sum_{i=1}^{n} |r_i - l_i|$$

where n denotes the total number of springs in the system, and r_i and l_i denote the rest length and current length of spring i respectively.

6.1 Cube Tests - Critically Damped

The forces applied in all test cases in this section are applied to the skeleton in the single skeleton model for 0.05 seconds. After which time, the skeleton is stopped abruptly with its velocity set to 0, then the skeleton is held to the same position for the remainder of the simulation with a force in the y direction in order to counter gravity. Gravity in the system is assumed to be $-9.81 m/s^2$ in the y direction, and affects the skeleton as well as each individual nodal point at all times.

In this section, we critically damp all of our systems using the equation

$$d_c = \sqrt{k}$$

where d_c denotes the critical damping constant, and k denotes the spring constant. When we alter the spring constant, we also change the damping constant to ensure a critically damped system.

We first test the effects of modifying the stiffness of the system by fixing the sample rate to 7 (per side of the cube), time step to 0.0005 seconds, and force to 2000 $kg\,m/s^2$, then we change the spring constant from 100 (Figure (6.1)) to 225 (Figure (6.2)). From these two figures, we see that by increasing the stiffness by a factor of 2.25, the total deformation decreases by approximately 30% throughout the simulation, but the general *shape* of the deformation plots remains similar. Furthermore, although the three systems perform just about the same, the reduced DOF system has the least amount of deformation.

We increase the sampling rate to 13 (per side of the cube), and run the tests with the same parameters as described above. We get two similar graphs as figures (6.1) and (6.2) shown in figures (6.3) and (6.4) except that the total deformation is about twice as much as the ones with sampling rate of 7 for both the mass spring and implicit integration systems. The total deformation of the reduced DOF system, however, does not change much.

Repeating the parameters used in creating figures (6.1), (6.2), (6.3), and (6.4), we generate 4 more figures with the time step increased from 0.0005 to 0.005 (Figures (6.5), (6.6), (6.7), and (6.8)). We see that in the cases where the spring constant is 100 (Figures (6.5) (6.7)), the mass spring system and the implicit system are both stable, but the reduced DOF system begins to be *stressed out.* The spikes in the graphs are the results of oscillations of the volume similar to the reaction of a plasto-elastic object affected by an impulsive force. If we let the simulation run a little bit longer when the spring constant is 100, we see that eventually the reduced DOF system stabilizes (Figures (6.9) (6.10)). On the other hand, in both instances where the spring constant is 225, the mass spring system and the reduced DOF system both become unstable while the implicit integration system remains intact. Again, if we let the simulation run a little longer, we see that the mass spring system gains energy at an exponential rate, becoming unstable much faster than the reduced DOF system (Figures (6.11) (6.12)).

When we further increase the time step to 0.01, the mass spring system and the reduced DOF system continue to struggle, and the implicit integration model begins to reach its limitation. With the sample rate set at 7 and spring constant at 100, the implicit integration system seems to be stable as shown in figure (6.13). However, anywhere beyond that configuration, the implicit integration system begins to fall apart (Figures (6.14) (6.15) (6.16)).

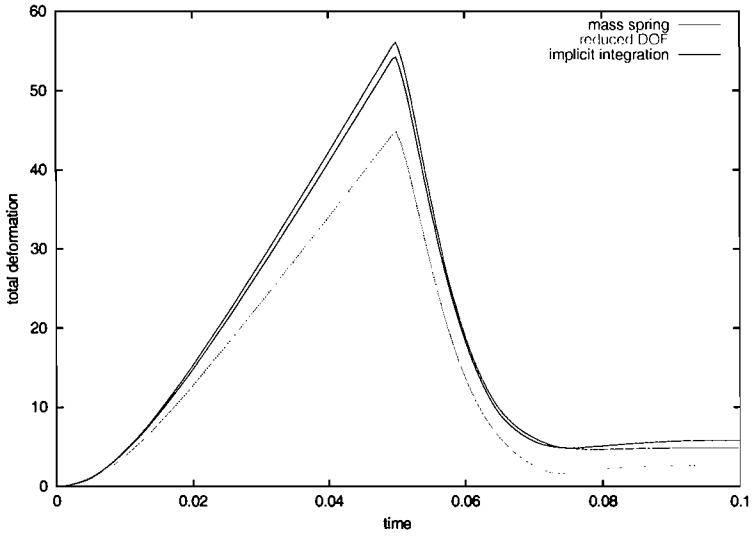


Figure 6.1: Sample Rate 7, Spring Constant 100, Damping Constant 10, Time Step 0.0005, Force 2000

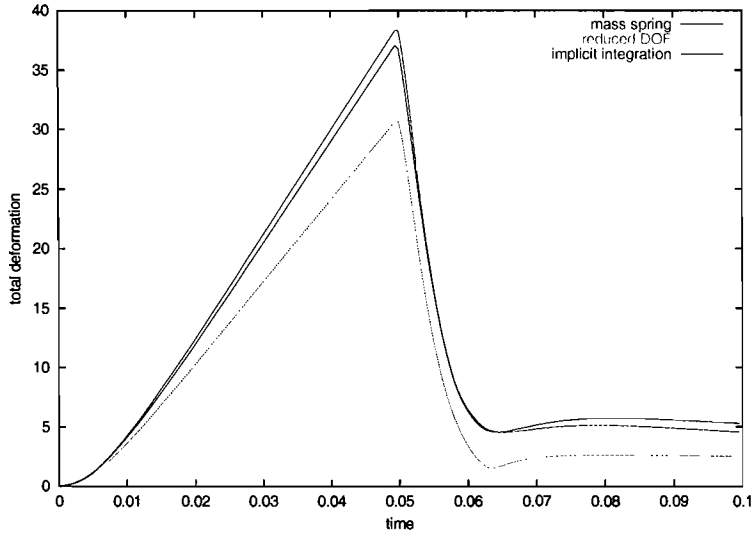


Figure 6.2: Sample Rate 7, Spring Constant 225, Damping Constant 15, Time Step 0.0005, Force 2000

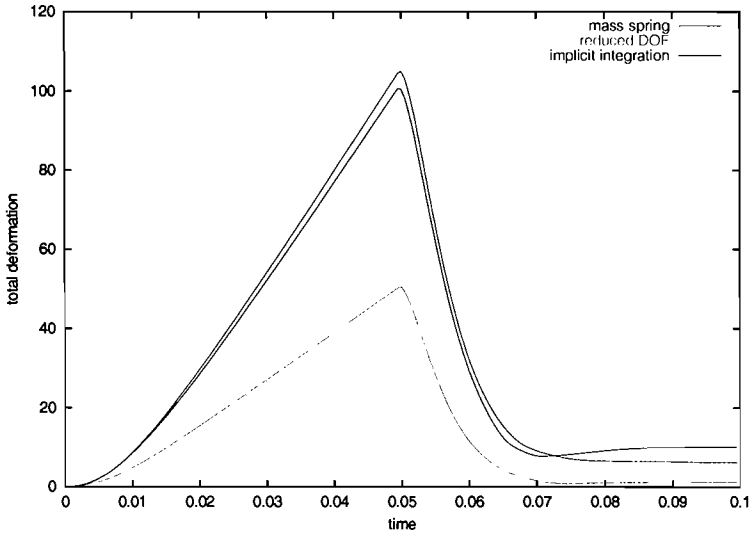


Figure 6.3: Sample Rate 13, Spring Constant 100, Damping Constant 10, Time Step 0.0005, Force 2000

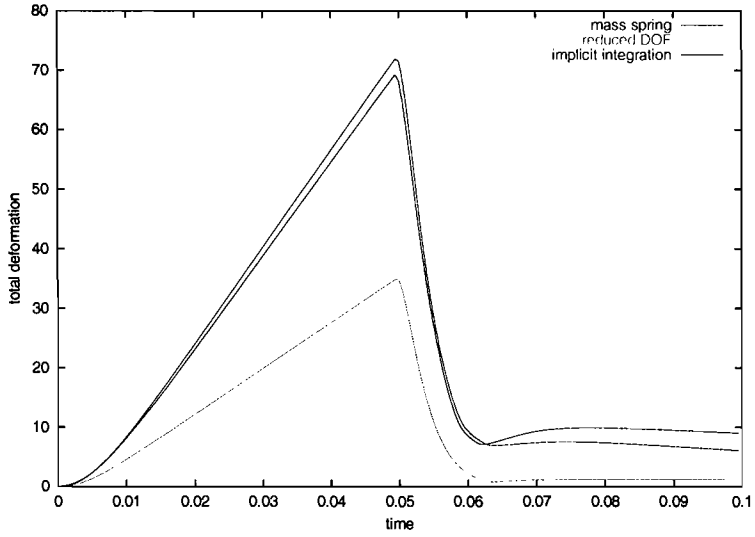


Figure 6.4: Sample Rate 13, Spring Constant 225, Damping Constant 15, Time Step 0.0005, Force 2000

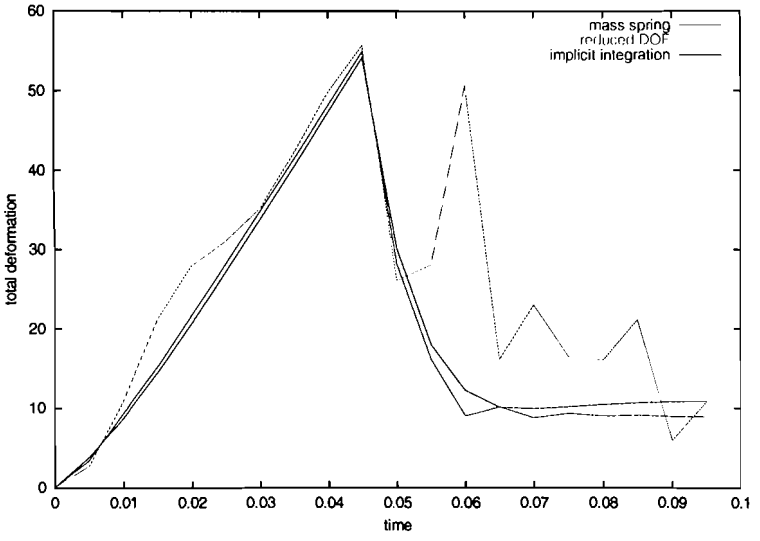


Figure 6.5: Sample Rate 7, Spring Constant 100, Damping Constant 10, Time Step 0.005, Force 2000

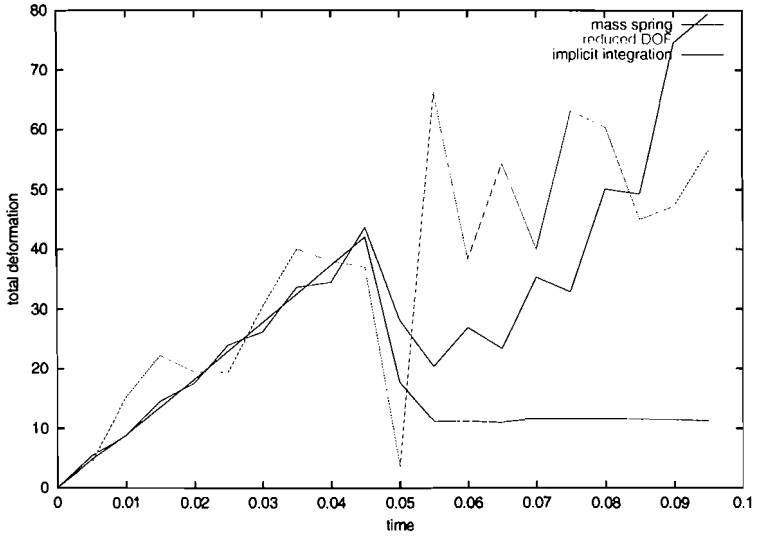


Figure 6.6: Sample Rate 7, Spring Constant 225, Damping Constant 15, Time Step 0.005, Force 2000

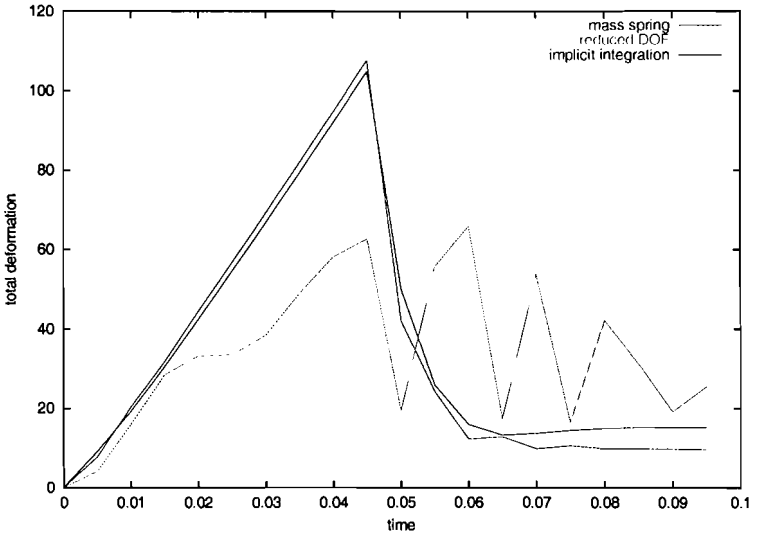


Figure 6.7: Sample Rate 13, Spring Constant 100, Damping Constant 10, Time Step 0.005, Force 2000

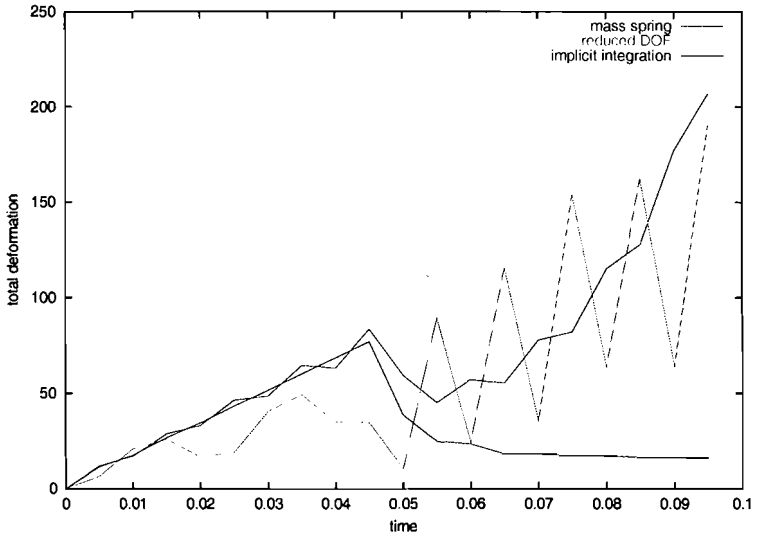


Figure 6.8: Sample Rate 13, Spring Constant 225, Damping Constant 15, Time Step 0.005, Force 2000

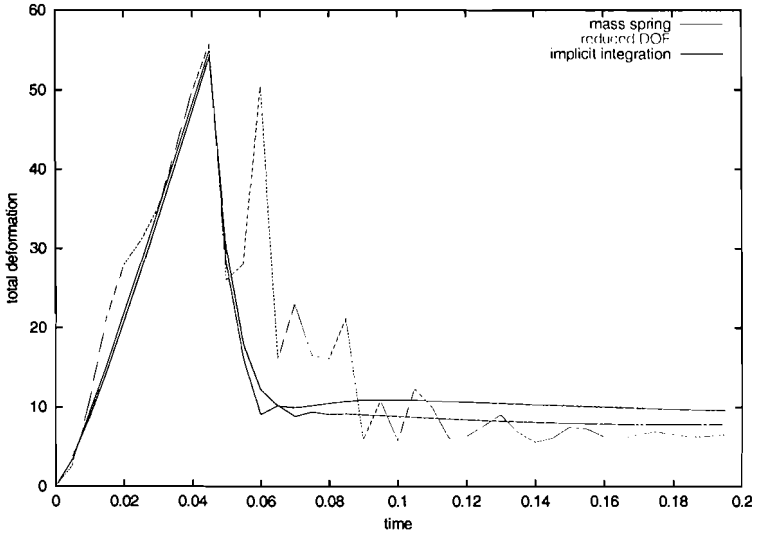


Figure 6.9: Sample Rate 7, Spring Constant 100, Damping Constant 10, Time Step 0.005, Force 2000 run for 2 seconds

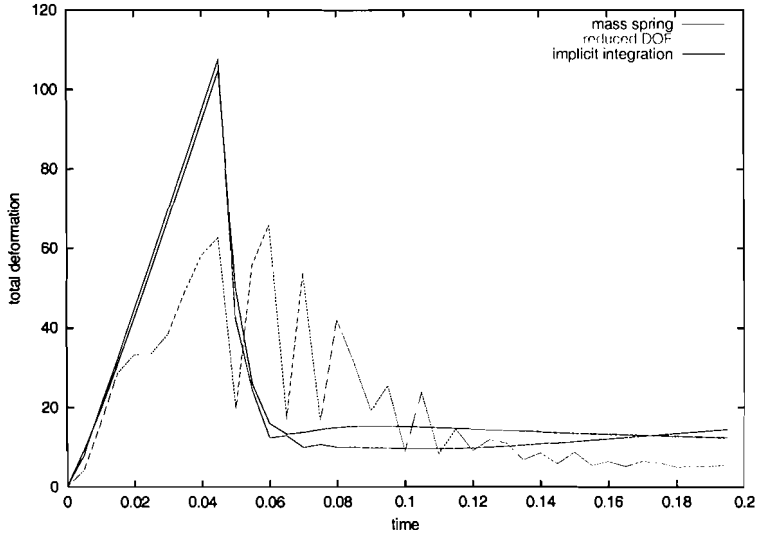


Figure 6.10: Sample Rate 13, Spring Constant 100, Damping Constant 10, Time Step 0.005, Force 2000 run for 2 seconds

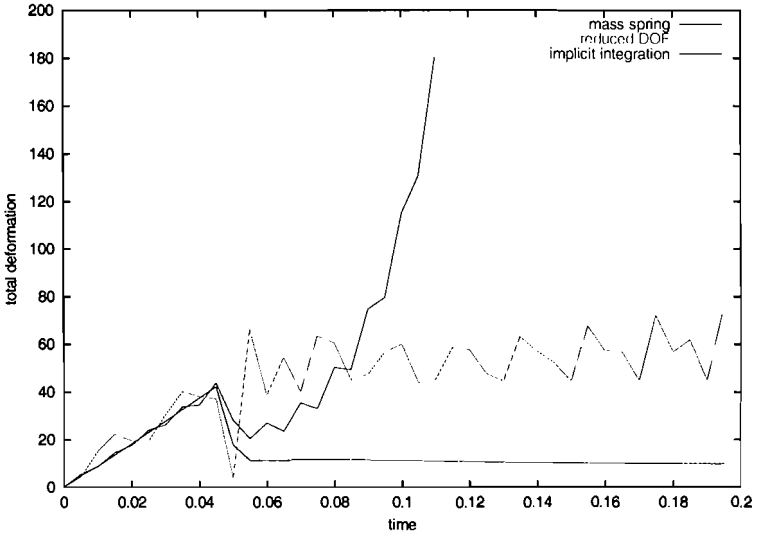


Figure 6.11: Sample Rate 7, Spring Constant 225, Damping Constant 15, Time Step 0.005, Force 2000 run for 2 seconds

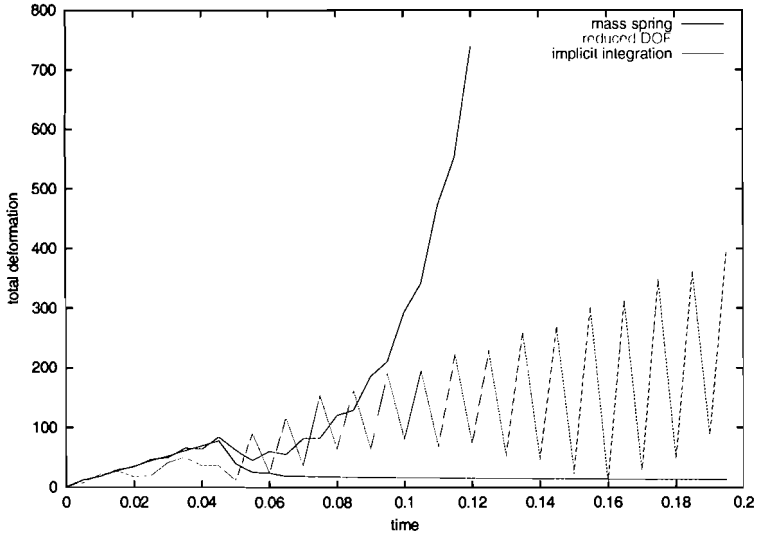


Figure 6.12: Sample Rate 7, Spring Constant 225, Damping Constant 15, Time Step 0.005, Force 2000 run for 2 seconds

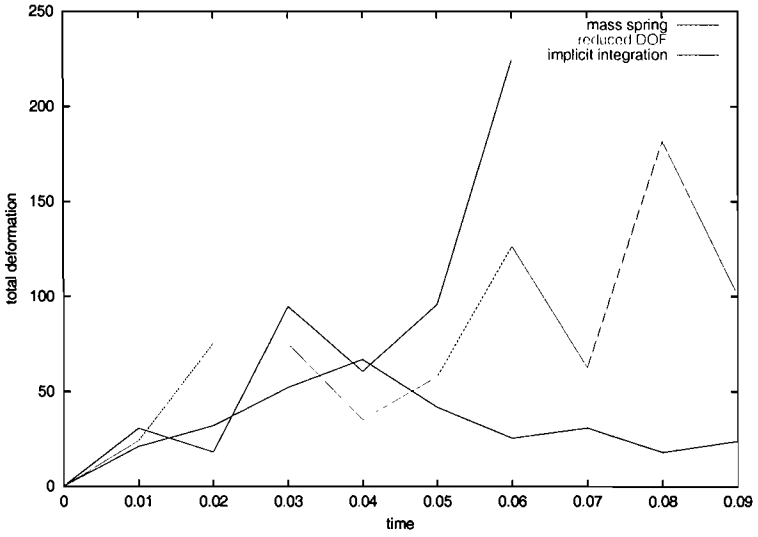


Figure 6.13: Sample Rate 7, Spring Constant 100, Damping Constant 10, Time Step 0.01, Force 2000

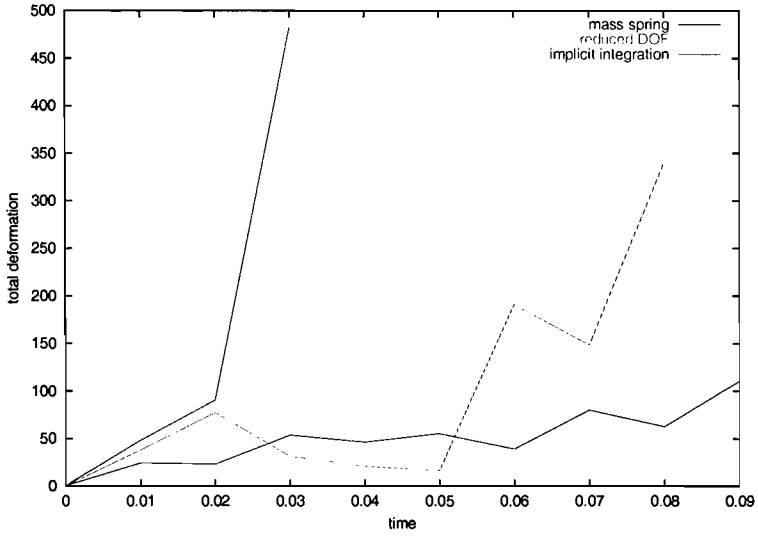


Figure 6.14: Sample Rate 7, Spring Constant 225, Damping Constant 15, Time Step 0.01, Force 2000

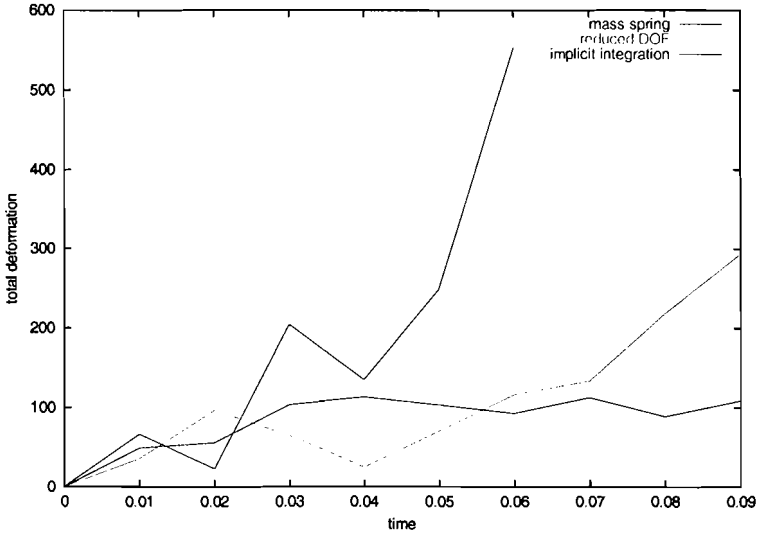


Figure 6.15: Sample Rate 13, Spring Constant 100, Damping Constant 10, Time Step 0.01, Force 2000

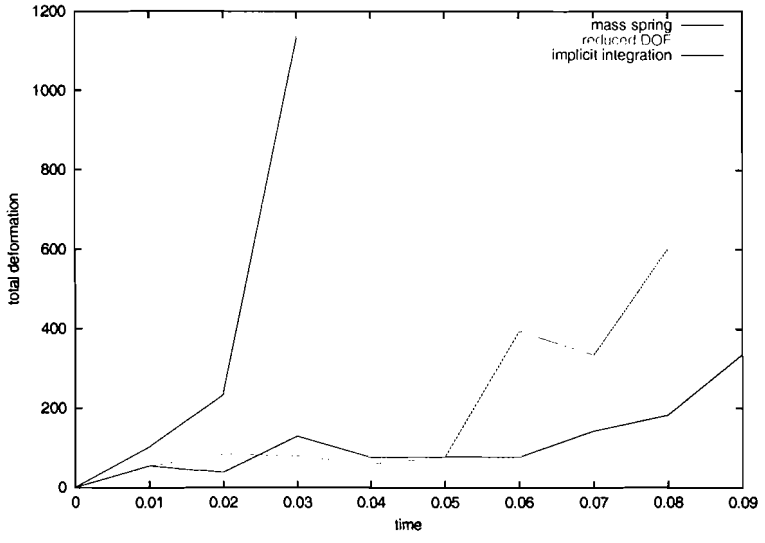


Figure 6.16: Sample Rate 13, Spring Constant 225, Damping Constant 15, Time Step 0.01, Force 2000

We also experiment with doubling the force from 2000 $kg\,m/s^2$ to 4000 $kg\,m/s^2$. This change does not alter the outcome of the tests except that the total deformation is approximately twice as high. The stability of each system does not seem to be affected by the increase in force.

6.2 Cube Tests - Underdamped

We attempt to explore some other properties of the implicit integration system used on underdamped models by gradually increasing the spring constant while holding the damping constant the same. We then apply large amount of forces to the skeleton in a short amount of time, simulating *impulses*. In the test cases that we created, we apply an impulse of 400,000 $kg\,m/s^2$ in the first 0.025 seconds of the simulation, then another impulse of -400,000 $kg\,m/s^2$ in the immediate following 0.025 seconds. After the impulses have been applied, the velocity of the skeleton is set to 0, and a force is applied to the skeleton to counter gravity.

Figures (6.17), (6.18), (6.19), and (6.20) show the increase in the spring constant from 225, 500, 1000, to 5000 while holding the damping constant at 3. The amount of oscillation increases in both the mass spring and reduced DOF systems as the spring constant increases. The implicit integration system, however, remains relatively stable throughout all the changes.

This test displays the property of implicit integration as described by Kass [6] where he relates the implicit integration method to a *filtering* process. The effect of filtering *smoothes out* the high frequency impulses and make the simulation more stable.

6.3 Speed

Although the implicit integration system out-performs the mass spring system and the reduced DOF system in stability, it certainly has its drawbacks. The most obvious is the amount of time required to compute each iteration of the simulation. Table [6.1] shows the amount of time required to generate 1 second of animation (Figure (6.21)) using each system in the single skeleton model with different sampling rate.

Table [6.2] shows the amount of time required to generate 1 second of animation using the reduced DOF and implicit integration systems on the full character model (Figures (6.22)(6.23)). Each system is sampled with 14,325 nodal points, and run with 0.00025 seconds as time step.

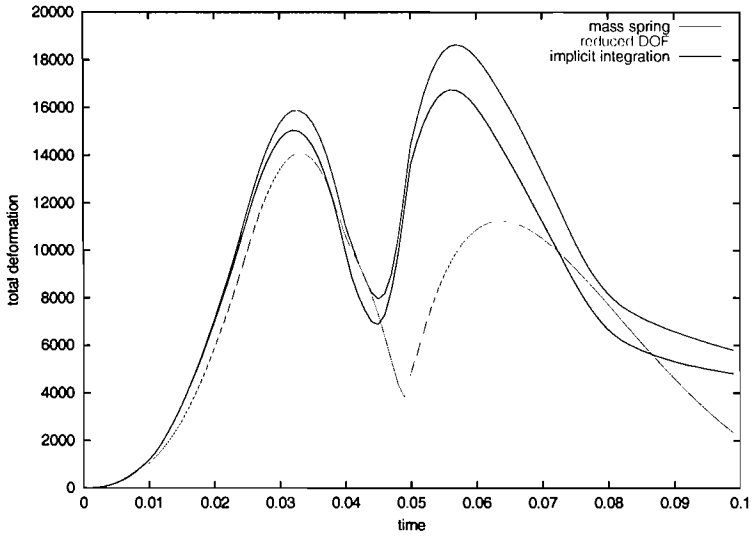


Figure 6.17: Sample Rate 7, Spring Constant 225, Damping Constant 3, Time Step 0.001

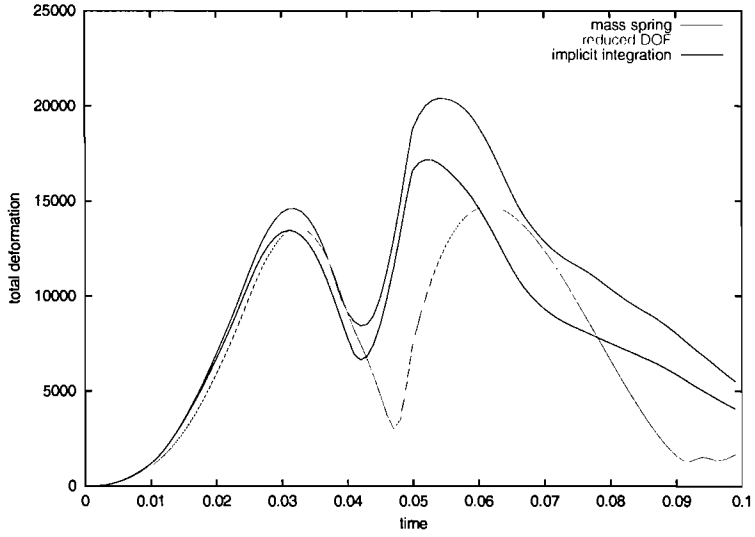


Figure 6.18: Sample Rate 7, Spring Constant 500, Damping Constant 3, Time Step 0.001

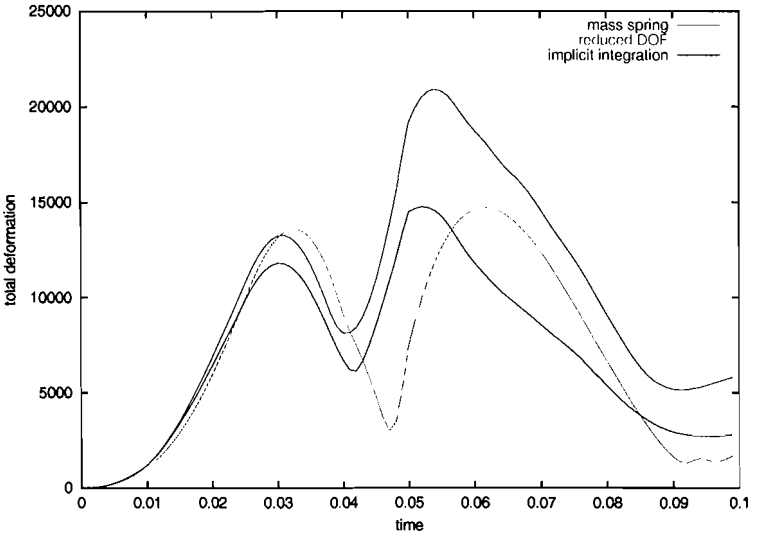


Figure 6.19: Sample Rate 7, Spring Constant 1000, Damping Constant 3, Time Step 0.001

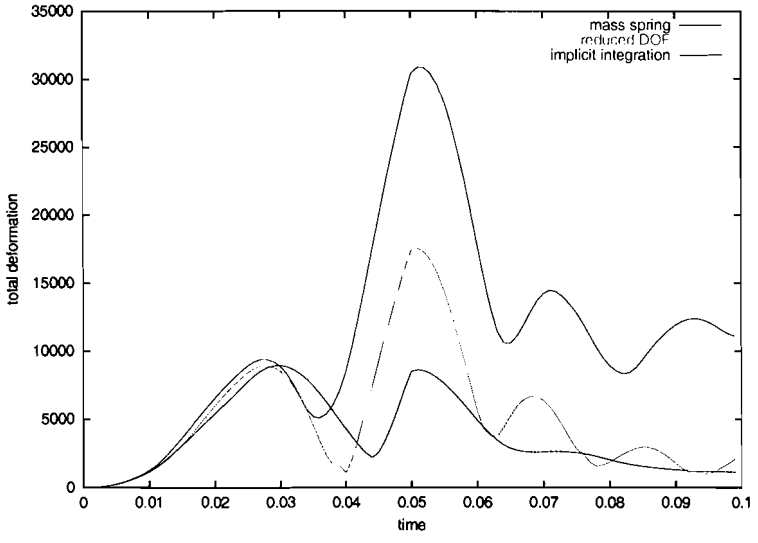


Figure 6.20: Sample Rate 7, Spring Constant 5000, Damping Constant 3, Time Step 0.001

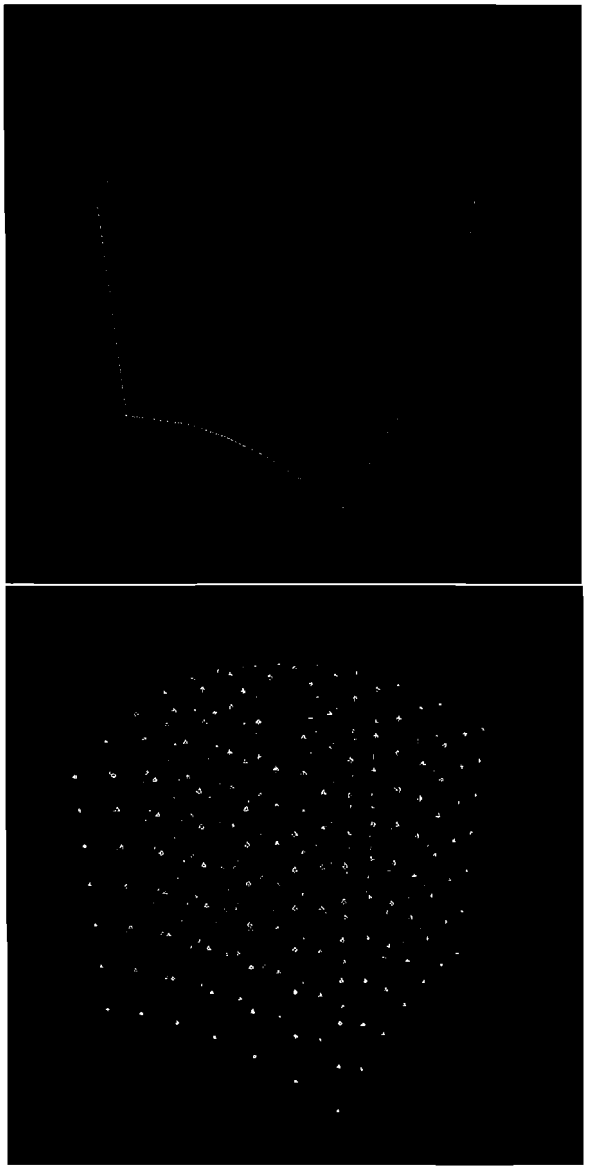


Figure 6.21: Deformed cube model (a) skin (b) skeleton and nodal points

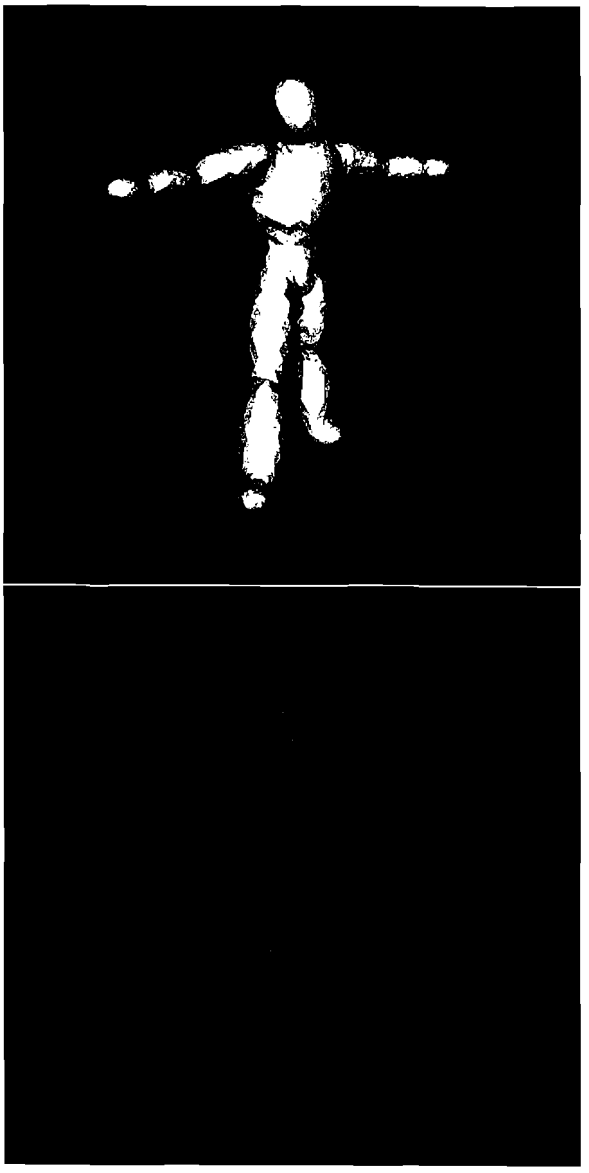


Figure 6.22: Deformed human character model frame 176 (a) skin (b) skeletons and nodal points

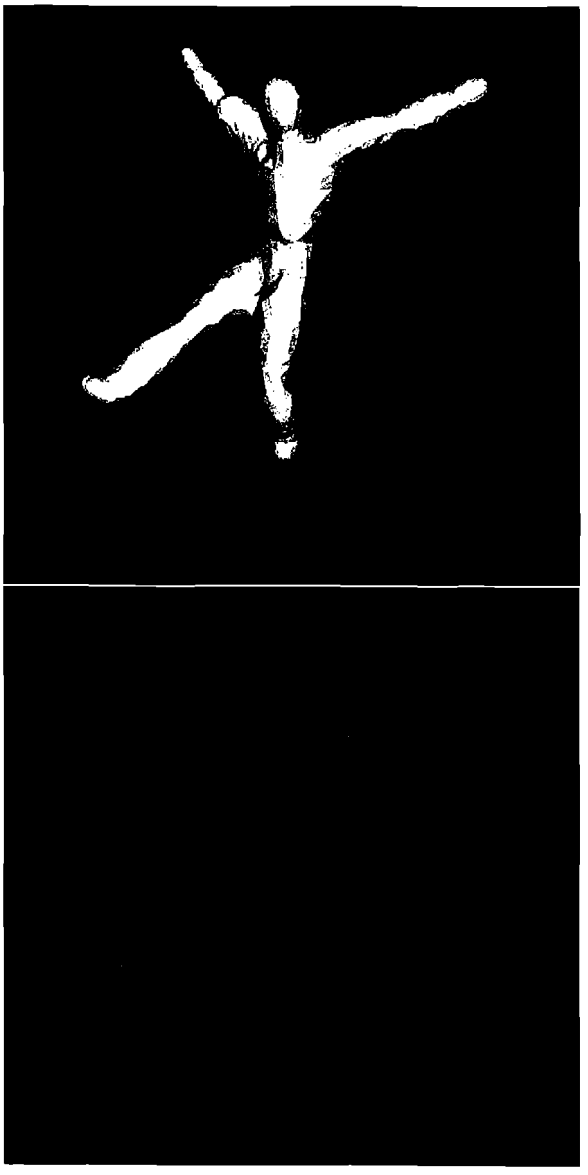


Figure 6.23: Deformed human character model frame 351 (a) skin (b) skeletons and nodal points

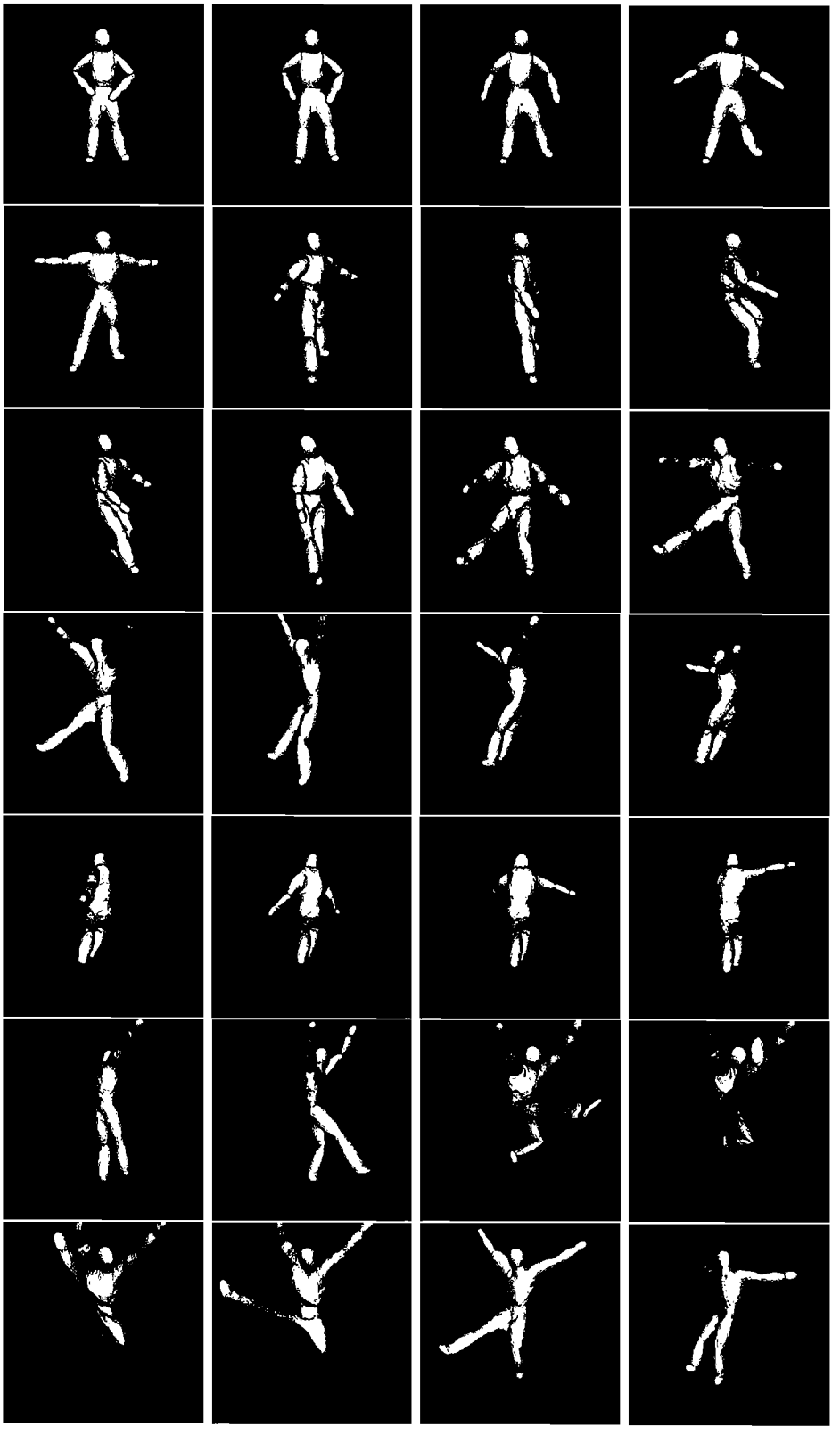


Figure 6.24: Sequence of animation

system	sample rate = 7	sample rate = 13
Mass Spring	4.06	32.0274
Reduced DOF	3.8412	18.03968
Implicit Integration	53.8828	2236.01

Table 6.1: number of seconds required to generate 1 second of animation using time step of 0.001 with the cube model

system	time
Reduced DOF	362.6972
Implicit Integration	35281.48

Table 6.2: number of seconds required to generate 1 second of animation using time step of 0.00025 with the full character model

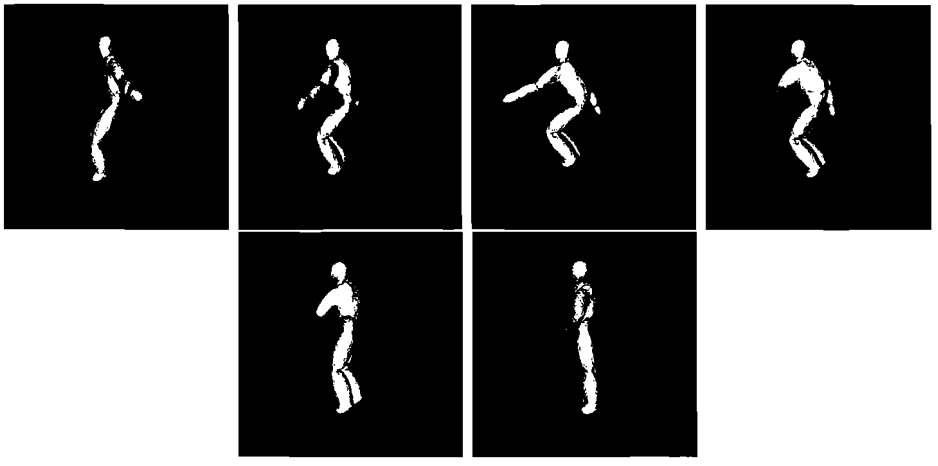


Figure 6.25: Sequence of animation continued

Chapter 7

Discussion

Although we have implemented three different deformation algorithms, our original goal of automating the process of creating an animation sequence of a viscoelastic model remained the same throughout. In our system, the only process that requires some amount of manual labor lies in creating the skeleton. The skeleton selection process plays a vital role to the stability of the simulation because it directly affects the amount of volume to be deformed. Decreasing the sizes of skeletons means that the number of nodal points between the skeletons and the surface is higher, which as the same effect as having higher sample rate, thus requiring the numerical integrator to take smaller time steps. On the other hand, if the skeletons are large, the number of movable points would decrease, limiting the amount of deformation. Therefore, given a continuous mesh as input, the user must first generate a suitable set of skeletons that is neither too small to potentially cause instability in the system, nor too big such that the amount of viscoelastic behavior becomes diminished. We approximated the skeletons as a collection of spheres scaled and stretched to resemble the character (Figure 3.2), and we have demonstrated that this approximation is adequate for accomplishing our goal.

7.1 Initialization

With the mesh and skeletons in place, the system takes the following as inputs:

- sample rate
- spring constant
- damping constant

- force description

Force description could either be motion capture data or key framed force information. With these data, the system begins the simulation and rendering automatically. We first sample the volume and identify the nodal points as VolumePoints, SurfacePoints, or SkeletonPoints. Then springs are added to connect the nodal points creating lattices with angular support[1]. Depending on the positions of the nodal points, each nodal point is assigned to one or more body parts based on its distances to the closest skeletons. Excess springs that span between body parts are removed according to the heuristics described in the section 3.6. At this point, the initialization step is complete, and the system is ready to start the animation and deformation process.

7.2 Stability Of The Deformation Models

The three deformation models that we implemented varied in stability and speed. From figures (6.1), (6.2), (6.3), and (6.4), we can see that with a small enough time step and a reasonable spring constant, all three models behave well. However, as the time step starts to increase, the mass spring model and the reduced DOF model begin to buckle under the stress. In figures (6.5) and (6.7), we notice that the reduced DOF model begin to show signs of destabilizing. The high spikes in the graphs are caused by the oscillation of the volume, which is similar to that of an underdamped system. The mass spring model, however, does not exhibit signs before becoming unstable. Between figures (6.5) and (6.6), the model transitions from being stable to exponentially gaining energy during the deformation. From this point on, any further stress on the mass spring model and reduced DOF model such as increasing time step or increasing spring constant only causes both models to destabilize in a shorter amount of time. The implicit integration model, on the other hand, shows great tolerance towards the changes in sample rate, time step, and spring constant. Only when the time step reaches 0.01 seconds does the model begin to deteriorate (Figures (6.13)(6.14) (6.15)(6.16)).

7.3 Speed Of The Deformation Models

The cost of having such stability in the implicit integration model is the speed of the simulation. Table [6.1] shows the amount of time required by each system to generate 1

[1]This is only for the full character model, the single skeleton model is connected differently as described in section 3.4.

second of animation using different sample rates with the cube model. The increase in the sample rate from 7 to 13 is approximately the same as increasing the number of nodal points 8 times because the sample rate represents the number of samples along each axis of the cube. The mass spring model shows that the computation time is $O(n)$ where n is the number of nodal points. The reduced DOF model, however, increases at a slower rate. Unlike the mass spring model where the numerical integrator computes the position and velocity of each individual nodal point (Section 5.1), the reduced DOF model computes the deformation matrix once, and applies the matrix to the nodal points for computing the positions and velocities making the simulation faster but less accurate (Section 5.2). From our experience, the amount of computation time in the reduced DOF model is dominated by the time required to generate the deformation matrix. Multiplying the matrix by n 3×1 position vectors requires much less time in comparison. In other words, although the complexity of the reduced DOF model is the same as the mass spring model at $O(n)$, reduced DOF model is faster because the constant associated with n is smaller. Lastly, the implicit integration model increases approximately 40 times in computation time as the sample rate increases from 7 to 13 due to the growth in the matrices required by the system. As described in section 5.3, such increase in the sample rate would increase the matrices from $(3 \times 7^3) \times (3 \times 7^3) = 1,058,841$ elements to $(3 \times 13^3) \times (3 \times 13^3) = 43,441,281$ elements. Because creating and multiplying these matrices is the primary cause of the computation time required by the implicit integration model, as the matrices grow in order of n^2, so does the complexity of the model ($O(n^2)$).

Table [6.2] shows the amount of time required to generate 1 second of animation with the character model. The reduced DOF model takes approximately 6 minutes, while the implicit integration model would require 588 minutes, or 9.8 hours. The animation sequence shown in section 6.3 lasts approximately 6 seconds, which translates to 36 minutes of simulation time using the reduced DOF model, and 58.8 hours with the implicit integration model.

7.4 Future Work

7.4.1 Deformation Models

There are areas in our algorithm that we believe could be improved upon. First and foremost, we have yet to test other deformation techniques that utilize the continuum model such as O'Brien's [23] finite element method that describes deformation based on internal energy. Using such continuum models might further increase the amount of computation

time, but could also increase the accuracy of the simulation.

7.4.2 Skeleton Generation

In order to make our entire system fully automated, we need to improve upon the skeleton generation process. Teichmann and Teller [17] explored the possibility of assisted articulation of polygonal models using *3D* Delaunay triangulation. In their system, a fair amount of user intervention is still required to create an accurate skeleton to represent the model, but it does offer the user the means to look at their mesh model while trying to generate the skeleton. We believe that we could adopt the same technique and principle and create a tool that would make our skeleton creation process equally interactive or even fully automated.

7.4.3 Collision Detection and Penetration Constraints

Our current system does not support collision detection or penetration constraints. These are necessary tools in making the system more robust and accurate in simulating complex environments. Although these features would further slow down the computation time, it is unavoidable for simulating the interaction between the character and other objects in the scene.

7.4.4 Sampling

The sampling technique used in our system currently relies on a *3D* grid. This creates many artifacts such as aliasing, under sampling, or over sampling. Ideally, the nodal points should be connected in not just tetrahedrons, but equilateral tetrahedrons where all springs have equal rest length and all neighboring nodal points are equally distanced apart. *3D* meshing of a volume using equilateral tetrahedron, however, is currently a difficult problem with no clear solution. In our system, we believe we could relax the constraint of equilateral tetrahedrons to approximate equilateral tetrahedrons using some relaxation technique on the placement of nodal points and not lose too much accuracy in the simulation.

7.4.5 User Defined Constants

The three user defined constants in our system that critically control the results of the deformation are the spring constant, the damping constant, and the time step. Currently, finding the correct values of these constants is purely based on trial and error. If a particular sequence of animation is not desirable in terms of stability or lack of viscoelastic behavior, the user has to change the constants based on intuition and restart the simulation again.

Although the definition of "lack of viscoelastic behavior" is quite subjective, we believe that we could still offer some tool that allows for a preview of the effects of some selected constants. Animators then would not have to spend hours rendering and re-rendering until the desired constants are found. Furthermore, we would also like give the option of assigning specific springs different behaviors or constants to simulate *layers* similar to that of the system created by Waters [18].

Chapter 8

Summary

The goal of this thesis is to animate a viscoelastic character model in an automated fashion. We have demonstrated that by supplying the character model with a set of skeletons, we were able to make the character dance in a physically plausible manner similar to that of flubber. Our approach represented the model using nodal points and springs, and applied three different deformation techniques for the animation process. The deformation techniques vary in speed, stability, and robustness, but each has its own strengths and weaknesses.

We tested the three deformation techniques using a simple cube model. The results demonstrated that the mass spring model, although fast and easy to construct, varied greatly in stability when the system parameters were modified slightly. The implicit integration model, on the other hand, was very tolerant towards changes in parameters, but suffered in performance as the number of nodal points increased. The reduced degree of freedom model fit between the previous two models, sacrificing some accuracy such that it is slightly faster than the mass spring model, but slightly less tolerant, and faster than the implicit integration model, but not quite as stable.

Based on the fact that not one technique had a clear advantage over the other two, we have essentially offered animators a set of tools for animating and simulating viscoelastic models. These systems could be alternatives for the traditional manual keyframing and space warping deformation methods because of their ease of use and speed. Depending on the needs of the animator, a system could be selected to assist in the process of creating a sequence of pleasing animation with the least amount of effort.

Bibliography

[1] D. Baraff and A. Witkin. Large steps in cloth simulation. In *Computer Graphics Proceedings, Annual Conference Series*, Proceedings of SIGGRAPH 98. ACM SIGGRAPH 1998.

[2] L. Markosian, J. Cohen, T. Crulli, and J. Hughes. Skin: a constructive approach to modeling free-form shapes. In *Computer Graphics Proceedings, Annual Conference Series*, Proceedings of SIGGRAPH 99. ACM SIGGRAPH 1999.

[3] Vasiliki Chatzi. Integer-coordinate crystalline meshes. PhD thesis, Brown University Department of Computer Science, 2000.

[4] George, P. L., Automatic mesh generation: Application to finite element methods, John Wiley & Sons, 1991.

[5] Numerical Recipes in C.

[6] Kass, Michael, An introduction to physically based modeling: an introduction to continuum dynamics for computer graphics, SIGGRAPH Course Notes, 1997.

[7] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10:350-355, September 1978.

[8] Sarah F. Gibson and Brian Mirtich. A survey of deformable models in computer graphics. Technical Report TR-97-19, Mitsubishi Electric Research Laboratories, Cambridge, MA, November 1997.

[9] A. Barr. Global and local deformations of solid primitives. In *Computer Graphics Proceedings, Annual Conference Series*, Proceedings of SIGGRAPH 84, pages 21-30. ACM SIGGRAPH 1984.

[10] T. Sederberg and S. Parry. Free-form deformation of solid geometric models. In *Computer Graphics Proceedings, Annual Conference Series*, Proceedings of SIGGRAPH 86, pages 151-160. ACM SIGGRAPH 1986.

[11] S. Coquillart. Extending free-form deformation: a sculpting tool for 3D geometric modeling. In *Computer Graphics Proceedings, Annual Conference Series*, Proceedings of SIGGRAPH 90, pages 187-196. ACM SIGGRAPH 1990.

[12] S. Coquillart and P. Jancene. Animated free-form deformation: an interactive animation technique. In *Computer Graphics Proceedings, Annual Conference Series*, Proceedings of SIGGRAPH 91, pages 23-26. ACM SIGGRAPH 1991.

[13] W. Hsu, J. Hughes, and H. Kaufman. Direct manipulation of free-form deformations. In *Computer Graphics Proceedings, Annual Conference Series*, Proceedings of SIGGRAPH 92, pages 177-184. ACM SIGGRAPH 1992.

[14] R. MacCracken and K. Joy. Free-form deformations with lattices of arbitrary topology. In *Computer Graphics Proceedings, Annual Conference Series*, Proceedings of SIGGRAPH 96, pages 181-188. ACM SIGGRAPH 1996.

[15] J. Chadwick, D. Haumann, and R. Parent. Layered construction for deformable animated characters. In *Computer Graphics Proceedings, Annual Conference Series*, Proceedings of SIGGRAPH 89, pages 243-252. ACM SIGGRAPH 1989.

[16] X. Tu and D. Terzopoulos. Artificial fishes: physics, locomotion, perception, behavior. In *Computer Graphics Proceedings, Annual Conference Series*, Proceedings of SIGGRAPH 94, pages 43-50. ACM SIGGRAPH 1994.

[17] M. Teichmann and S. Teller. Assisted articulation of closed polygonal models. in *Prof. 9th Eurographics Workshop on Animation and Simulation*, July 1998.

[18] K. Waters. A muscle model for animating three-dimensional facial expression. In *Computer Graphics Proceedings, Annual Conference Series*, Proceedings of SIGGRAPH 87, pages 17-24. ACM SIGGRAPH 1987.

[19] G. Celniker and D. Gossard. Deformable curve and surface finite elements for free-form shape design. In *Computer Graphics Proceedings, Annual Conference Series*, Proceedings of SIGGRAPH 91. ACM SIGGRAPH 1991.

[20] J.P. Gourret, N. Magnenat-Thalmann, and D. Thalmann. Simulation of object and human skin deformations in a grasping task. In *Computer Graphics Proceedings, Annual Conference Series*, Proceedings of SIGGRAPH 89, pages 21-30. ACM SIGGRAPH 1989.

[21] D. Chen and D. Zeltzer. Pump it up: computer animation of a biomechanically based model of muscle using the finite element method. In *Computer Graphics Proceedings, Annual Conference Series*, Proceedings of SIGGRAPH 92, pages 89-98. ACM SIGGRAPH 1992.

[22] M. Bro-Nielsen and S. Cotin. Real-time volumetric deformable models for surgery simulation using finite elements and condensation. In *Proceedings of Eurographics*, volume 15, pages 57-66, 1996.

[23] J. O'Brien and J. Hodgins. Graphical Modeling and Animation of Brittle Fracture. In *Computer Graphics Proceedings, Annual Conference Series*, Proceedings of SIGGRAPH 99. ACM SIGGRAPH 1999.

[24] P. Faloutsos, M. van de Panne, and D. Terzopoulos. Dynamic animation synthesis with free-form deformations. in *IEEE Transactions on Visualization and Computer Graphics*, Volume 3, Number 3, pages 201-214, July-September 1997.

[25] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: active contour models. *International J. Computer Vision*, 1(4):321-332, 1987.

[26] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. In *Computer Graphics Proceedings, Annual Conference Series*, Proceedings of SIGGRAPH 87, pages 205-214. ACM SIGGRAPH 1987.

[27] D. Terzopoulos and A. Witkin. Physically based models with rigid and deformable components. *IEEE Computer Graphics and Applications*, pages 41-51, November 1988.

[28] D. Terzopoulos and K. Fleischer. Modeling inelastic deformation: viscoelasticity, plasticity, fracture. In *Computer Graphics Proceedings, Annual Conference Series*, Proceedings of SIGGRAPH 88, pages 269-278. ACM SIGGRAPH 1988.

[29] A. Pentland and J. Williams. Good vibrations: modal dynamics for graphics and animation. In *Computer Graphics Proceedings, Annual Conference Series*, Proceedings of SIGGRAPH 89, pages 215-222. ACM SIGGRAPH 1989.

[30] A. Witkin and W. Welch. Fast animation and control of nonrigid structures. In *Computer Graphics Proceedings, Annual Conference Series*, Proceedings of SIGGRAPH 90, pages 243-252. ACM SIGGRAPH 1990.

[31] D. Baraff and A. Witkin. Dynamic simulation of non-penetrating flexible bodies. In *Computer Graphics Proceedings, Annual Conference Series*, Proceedings of SIGGRAPH 92, pages 303-308. ACM SIGGRAPH 1992.