Soft Body Deformation Dynamics Based on Shape Matching

Honey Sharma

Master of Science,

Computer Animation and Visual Effects

August, 2013

Contents

Table of contents i
List of figures iii
Abstract iv
Acknowledgements v

1 Introduction 1

2 Related work 4

3 Technical Background 7

3.1 Meshless Animation 7
3.2 Integration Schemes 9
3.2.1 Modified Euler Integration 9
3.3 The Algorithm 10
3.3.1 Shape Matching 11
3.3.2 Extensions 13

4 Design and Implementation 17

4.1 Design 17
4.1.1 Architecture Description 17
4.2 Collision Handling 22
4.2.1 Collision Detection 22
4.2.2 Collision Response 24
4.2.3 Spatial Partitioning 24

5 Applications and Results 25

5.1 Basic 25
5.2 Linear 26

5.3 Quadratic 27
5.4 Clustering 29

6 Conclusion **32**
6.1 Summary 32
6.2 Known bugs and issues 33
6.3 Future work 33

7 References **34**

References **35**

List of Figures

2.1 Two node mass spring system by Wikipedia 4

2.2 Free Form Deformation. 5

3.1 An explicit scheme integrating linear spring by Muller et al. (2005) 10

3.2 The original shape that is x_i^o is matched with the shape x_i and these are pulled towards the goal positions g_i by Muller et al. (2005) 11

3.3 The extensions presented by Muller et al. (2005) 13

3.4 Overlapping regions sharing the particles. 16

3.5 Clustering with different number of clusters (Muller et al. (2005) 16

3.6 Plasticity presented by Muller et al. (2005) 16

4.1 The complete class diagram. 18

4.2 The sphere plane collision. 22

4.3 The sphere box collision (Ericson, 2005). 23

Abstract

Soft body deformation has become one of the areas of intense research and recently has attracted a lot of attention specially towards the real time implementation which is not only computationally is cheap but easy and efficient to implement. One of the recent breakthroughs in simulation of soft bodies is "Meshless Deformation Based on Shape Matching".

This thesis will explore the concepts behind the implementation of this technique in detail along with handling of collisions. The results achieved includes the implementation of the basic algorithm along with four extensions mentioned in the paper.

Acknowledgements

I would first like to thank Dr. Jian Chang and Dr. Xiaosong Yang for helping me decide this project and guiding me throughout the course of the implementation. I would also like to thank Jonathan Macey and Mathieu Sanchez for always helping me with everything which also includes answering to my silly quesitons.

Chapter 1

Introduction

Soft body deformation is an area in computer graphics emphasising on visually convincing physical simulation of deformable entities. Unlike rigid objects, when an external force is applied on a deformable or a soft body it will change shape and it is expected to retain its original shape to a certain extent. Thus, soft objects in virtual environment should also exhibit this property to increase the authenticity and realism in various applications like games, films and even scientific visualization like virtual surgery. Due to the broad spectrum of soft body deformation it can be used to simulate quite different range of materials like skin, muscle, cloth and various other things.

Most of the methods for simulating soft bodies provide physically plausible results but there are other methods as well which are known to produce physically accurate simulation such as Finite Element Methods.

Problem statement

Recent breakthroughs in various fields of computer graphics be it hardware or rendering have enabled the development of realistic real time simulations. But inspite of these improvements most of the current ap-

plications are using the approach based on rigid bodies because of simplicity, ease of control and easily available libraries. Another reason can be the efficiency and cost of computation which can spike up while ensuring the stable solution.

In 2005 Muller et al. introduced **"Meshless Deformation Based on Shape Matching"** as a new approach to solve all the underlying issues by providing a geometrically motivated model which is simple enough to ensure the ensure the efficiency, stability and controllability.

Objectives and contributions

The following were the objective of the thesis and their accomplishment will be analysed in the end of the thesis.
1. Implement the basic algorithm to simulate a deformable object as presented in the paper by Muller et al. (2005).
2. Implement the other extensions given in the paper like:

- Rigid Body Dynamics.
- Linear Deformation.
- Quadratic Deformation.
- Cluster Based Deformation.
- Plasticity.

3. Implement collision detection and response for deformable bodies.

Structure

The following is the overview of the thesis:

Chapter 2 : Related Work - discusses the approaches introduced by various authors to simulate the deformation of soft bodies, particularly

those which have influenced this project.

Chapter 3: Technical Background - discusses the approach presented in the paper in detail along with all the extensions.

Chapter 4: Implementation - discusses the in depth implementation of the concepts presented in the paper including all the algorithms used to accomplish them along with problems or challenges faced during the course of implementation.

Chapter 5: Application and Results - discusses the results of the implementation and its applications in different forms.

Chapter 6: Conclusion - discusses the known bugs, limitations and issues along with future work to enhance the current tool.

Chapter 2

Related work

There are many approaches to realise the simulation of deformable bodies:
Rigid-body Dynamics can also be used to deform an object by using multiple connected rigid bodies with constraints and for rendering purpose a surface mesh can be generated, for example by using matrix-palette skinning as demonstrated in Havok physics engine.

Mass Spring System is represented by a set of nodes connected by an elastic spring following a modification of Hook‘s law. The set of nodes and springs can form an one dimensional (hair strand or rope) , two dimensional (cloth) or a three dimensional (jello cube) network (see Figure 2.1). By applying the Newton‘s second law of motion to a node or point mass comprising of external as well as spring forces a system of differential equations can be obtained which can be solved by methods for computing Ordinary Differential Equations (ODEs).

Finite Element Method is traditionally being used in serious engineering but it is also gradually finding its roots in feature films and

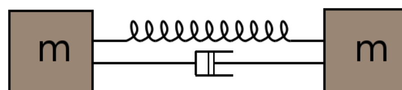


Figure 2.1: *Two node mass spring system by Wikipedia*

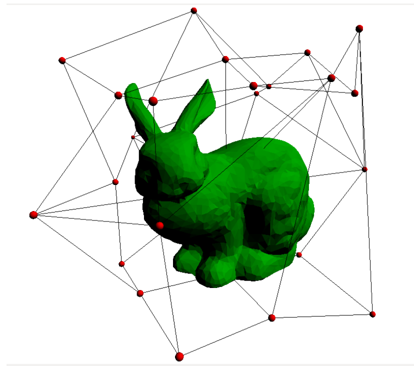


Figure 2.2: *Free Form Deformation.*

gaming industry. In 2001 Muller et al., presented a hybrid technique to simulate deformation and fracturing of materials in real time. They were calculating the effects of impact forces at discrete collision events. The deformations are computed based on internal stress tensor. The underlying technique uses finite elements to calculate static equilibrium response. In 2009 Parker et al., simulated real time deformation and fracturing using corotational tetrahedral finite element method. Using the underlying method they created an engine robust and fast enough to handle any situation at real time.

This approach is more physically accurate and solves partial differential equations to simulate an elastic material but it is computationally more expensive compared to the other less physically accurate models even with recent breakthroughs, which makes it less suitable for many real time applications.

Free Form Deformation or FFD was introduced by Serderberg and Parry in 1986 and since then it has become an important technique in computer graphics which allows to deform an object by deforming a parallelepiped region or lattice in which it is enclosed. The deformation of lattice is based on hyper patches that are similar to parametric curves like Bzier or NURBs (see Figure 2.2).

Shape Matching is a different approach on simulating deformable objects by managing the point based objects without bothering about con-

nectivity information unlike finite element methods. Here the target positions are approximated by generalizing shape matching of unmodified state with modified state of point cloud. According to approach forces are applied to the points or the particles to move them towards their original shape. Rotation of the deformable object must be calculated properly by using polar decomposition.

In 2007 Rivers et al., presented a lattice based shape matching technique to approximate volumetric large deformation dynamics for simulation. They introduced a fast lattice based shape matching (FastLSM) to simulate many soft bodies in real time on CPU.
The various approaches mentioned above can be divide into two categories, namely **Physically Based** and **Non-Physically Based**. Both approaches have their respective advantages and disadvantages. While physically based approaches are more accurate and parameters can be taken from real world objects for example Finite Element Method but they are computationally expensive and on the other hand non-physically based approaches are less accurate but cheap to compute but in this case parameters cannot be taken from the real world objects for example Mass Spring System and Meshless Shape Matching.

Chapter 3

Technical Background

The main concept behind meshless deformation based on shape matching is to provide a simple model represented as points with mass which is easy to implement and can be animated like a particle system without connectivity information which makes it ideal for real time applications like games as no volumetric structure is needed and each particle is updated individually which makes it efficient, unconditionally stable with less memory requirements.

3.1 Meshless Animation

Almost all the physically based simulations like rigid body dynamics, fluid simulation and others require the understanding of the Newton‘s second law of motion. It is important to have a clear understanding of the dependency among force, acceleration, velocity and distance. The forces are generated by derivations from equilibrium which accelerate the object back to its equilibrium state.

And in order to calculate the location of the object acceleration and velocity and are numerically integrated with respect to time and derivation is other way round, the velocity is the derivation of the location with respect to time and acceleration is the derivation of velocity with

respect to time. Mathematically it can be represented as:

$$v(t) = x(t) \tag{3.1}$$

$$a(t) = v(t) \tag{3.2}$$

where, x is position, v is velocity, a is acceleration and t is time.

Newton's law of motion gives the relation between the force, mass and acceleration of an object. It depicts that how much force should be applied to accelerate an object of certain mass.

$$F = ma \tag{3.3}$$

It is one of the most important equation which shows how the location and the velocity of an object will be affected with the change in the force. As mentioned above that acceleration is the derivation of velocity with respect to time and hence it will change when there is a change in force which in turn, will also change the location.

The gravitational acceleration or simply gravity is multiplied with mass to generate external force which is added to each particle as $h\frac{f_{ext}(t)}{m_i}$

where, $f_{ext}(t)$ is the external force which is same for every particle, m_i is the mass of each particle and h is the time step. Since gravity does not depend on the actual algorithm of shape matching so it can easily be added to each particle by increasing the velocity in each step.

3.2 Integration Schemes

The most important issues in numerical integration are efficiency and stability. Implicit integration is stable without depending on the time step but it is very expensive to compute because it is required to solve a system of equations that are expensive to calculate on the other hand explicit integration is faster and but it is not unconditionally stable.

3.2.1 Explicit Euler Integration

The explicit Euler integration or forward Euler integration is the simplest of all integrators solving ordinary differential equations. It takes into account past values $f(x)$ and its state derivatives $hf'(x)$.

$$f(t+h) = f(x) + hf'(x) \tag{3.4}$$

3.2.2 Implicit Euler Integration

The implicit Euler integration or backward Euler integration scheme solves the ordinary differential equations but requires to take one step ahead for integration $hf'(x+h)$.

$$f(t+h) = f(x) + hf'(x+h) \tag{3.5}$$

3.2.1 Modified Euler Integration

Consider an example of mass spring system where force is:

$$f = -k(x(t) - lo)$$

where, k is the spring constant, lo is the spring length and one point is fixed at the origin and the other point is free with mass m The velocity

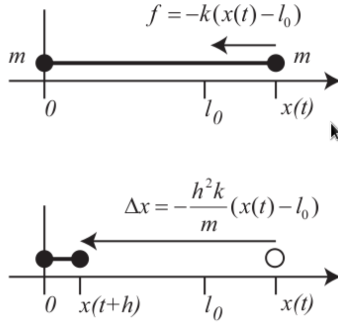


Figure 3.1: *An explicit scheme integrating linear spring by Muller et al. (2005)*

and position can be calculated with modified Euler integrator as:

$$v(t+h) = v(x) + h\frac{-k(x(t) - lo)}{m} \tag{3.6}$$

$$x(t+h) = x(t) + hv(t+h) \tag{3.7}$$

The velocity is calculated by explicit Euler step and position is calculated by implicit Euler step.

3.3 The Algorithm

As discussed briefly above that the main idea behind the given algorithm is to provide a simple model which is represented by a set of particles with initial positions x_i^0 and masses m_i without any interactions and connectivity among the them. But they do response to the external force and the collisions. At each time step the particles are moved towards their goal position and to calculate the goal positions each particle's original position x_i^0 is matched with the actual position x_i. And the velocity of the particle is changed and the object tries to retain its original shape based on the difference to the original shape (Figure 3.2).

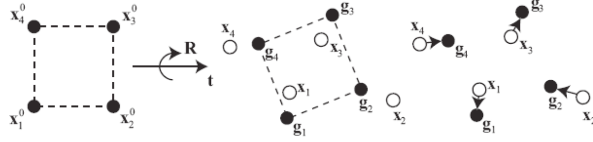


Figure 3.2: *The original shape that is x_i^o is matched with the shape x_i and these are pulled towards the goal positions g_i by Muller et al. (2005)*

3.3.1 Shape Matching

The main idea behind the algorithm is to take two set of positions, namely original position x_i^0 and actual position x_i and find translation vectors t_0 and t and the rotation matrix R which gives:

$$\sum_i w_i(R(x_i^0 - t_o) + t - x_i)^2 \tag{3.8}$$

where, w_i are weights but considered as masses ($w_i = m_i$). The translation vectors are in turn the center of mass of initial and actual shape which can be calculate by taking the partial derivate with respect to t_o and t, i.e.

$$t_o = x_{cm}^o = \frac{\sum_i m_i x_i^o}{\sum_i m_i} \tag{3.9}$$

$$t = x_{cm} = \frac{\sum_i m_i x_i}{\sum_i m_i} \tag{3.10}$$

It is simple to understand why it depends on the two translations and one rotaton, first translate the object to the origin and then perform rotation and then translate it back to its position.

In order to find the rotation which is a little bit more involved, we calculate the relative positions of particles with respect to its center of mass:

$$q_i = x_i^o - x_{cm}^o \tag{3.11}$$

$$p_i = x_i - x_{cm} \tag{3.12}$$

Now place the equations (1.11) and (1.12) in equation (1.8).

$$\sum_i m_i(Rq_i - p_i)^2 \tag{3.13}$$

The problem of finding the optimal rotation matrix has been reduced to finding the optimal linear transformation matrix **A**.

$$\sum_i m_i(Aq_i - p_i)^2 \tag{3.14}$$

To calculate linear transformation matrix, set the derivates with respect to all the enteries of matrix **A** to zero.

$$A = (\sum_i m_i p_i q_i^T)(\sum_i m_i q_i q_i^T)^{-1} = A_{pq}A_{qq} \tag{3.15}$$

The second term, that is A_{qq} gives a symmetric matrix which only has scaling information and no rotation information. The rotation part is contained in the term A_{pq} which can be found using polar decomposition.

$$A_{pq} = RS \tag{3.16}$$

where, $S = \sqrt{A_{pq}^T A_{pq}}$ gives the symmetrix part while $R = A_{pq}S^{-1}$. Hence, the goal position can be calculated as:

$$g_i = R(x_i^o - x_{cm}^o) + x_{cm} \tag{3.17}$$

The initial center of mass x_i^o, q_i and the matrix A_{qq} can be pre-computed. At each time step, we need to calculate the 3x3 matrix $A_{pq} = \sum_i m_i p_i q_i^T$. And the matrix **S** can be calculated by diagonalising the matrix $A_{pq}^T A_{pq}$ using Jacobi rotations.

3.3.2 Extended Integration The goal position g_i calculated in equation (1.17) can now be used to create an integrator which avoids overshooting:

$$v_i(t+h) = v(t) + \alpha \frac{g_i(t) - x_i(t))}{h} + h\frac{f_{ext}(t)}{m_i} \tag{3.18}$$

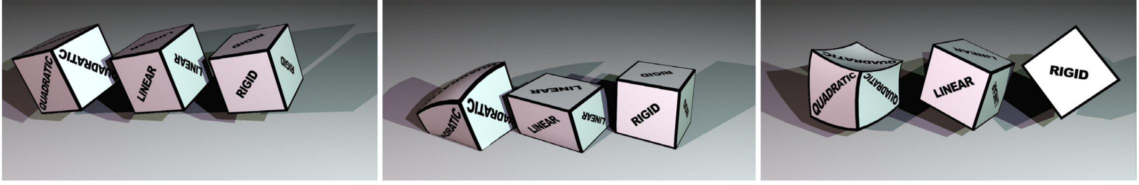


Figure 3.3: *The extensions presented by Muller et al. (2005)*

$$x_i(t+h) = x(t) + hv_i(t+h) \tag{3.19}$$

where $\alpha = [0...1]$ controls the stiffness of the deformable object. The differnce between this scheme and the one discussed earlier (equation (1.6) and (1.7)) is treatment of the internal elastic forces. When $\alpha = 1$ then particles move directly towards the goal position and when $\alpha < 1$, they will move towards the goal position.

Since the shape matching is done using center of mass which ensures that all the impulses are applied in equations (3.18) and (3.19) add up to zero and conserve the momentum. Even considering mass as the weights while calculating the matrix A_{pq} enforces the angular momentum.

3.3.2 Extensions

The authors have extended the basic algorithm discussed so far into five other extensions and out of those four have been implemented as the part of this thesis.

Rigid Body Dynamics

Rigid bodies can be simulated by assigning $\alpha = 1$ which ensures that the particles move to goal position instantly at each time step (Figure 3.3). In this case positions are translated and rotated with respect to initial position.

Linear Deformation

Linear deformations extends the range of deformations by allowing an object to shear and scale (Figure 3.3). So far only rotation was considered in calculating the goal position by in this case linear transformation matrix **A** calculated in equation (1.15) will be taken into account to perform the shape matching from initial position to the actual position. An additional parameter β will be added to the combination of linear transformation matrix **A** and rotation matrix **R** for calculating goal position equation.(**?**)

$$g_i = \beta A + (1 - \beta)R \tag{3.20}$$

where parameter $\beta = [0...1]$ controls the amount of linear deformation, the more value of beta means that shape will be deformed from its original shape or will not retain its original shape. To ensure that the volume remains conserved the matrix A is divided by the term $\sqrt[3]{det(A)}$ which makes sure that $det(A) = 1$.

Quadratic Deformation

Quadratic deformation further extends the range of motion by introducing twist and bends (Figure 3.3). Again the equation to calculate the goal position will be updated. The transformation matrices ($3x9$) for performing the quadratic deformation are quite larger when compared to the linear deformation ($3x3$) where the only transformations allowed were scaling, rotation, translation and shear. The equation for goal position is given as:

$$g_i = [AQM]\,\widetilde{q_i} \tag{3.21}$$

where $g_i \in \mathbb{R}^3$, $\widetilde{q_i} = \left[q_x, q_y, q_z, q_x^2, q_y^2, q_z^2, q_xq_y, q_yq_z, q_zq_x\right]^T \in \mathbb{R}^9$, $A \in \mathbb{R}^{3x3}$ which holds the coefficient of linear deformation, $Q \in \mathbb{R}^{3x3}$ which holds the coefficient of the quadratic deformation and $M \in \mathbb{R}^{3x3}$ holds the mixed terms. We can minimize the equation (1.13)

$$\sum_i m_i(\widetilde{A}q_i - p_i)^2 \tag{3.22}$$

where $\widetilde{A} = [AQM]\, \widetilde{q}_i \in \mathbb{R}^{3x9}$ The quadratic transformation will be:

$$\widetilde{A} = (\sum_i m_i p_i \widetilde{q}_i^T)(\sum_i m_i \widetilde{q}_i \widetilde{q}_i^T)^{-1} = \widetilde{A}_{pq} \widetilde{A}_{qq} \tag{3.23}$$

The symmetric matrix $\widetilde{A}_q q \in \mathbb{R}^{9x9}$ and $\widetilde{q}_i$ can again be pre computated. And the rotation matrix is $\widetilde{R} \in \mathbb{R}^{3x9} = [R00]$. Similar to linear deformation goal positions can be calculated as:

$$g_i = \beta \widetilde{A} + (1 - \beta)\widetilde{R} \tag{3.24}$$

Cluster Based Deformation

To further extend the spectrum of motion cluster based deformation is introducted where particles are divided among the overlapping clusters by subdividing the bounding space around mesh into overlapping bounding regions (Figure 3.4). For each cluster shape matching is performed to calculate the goal position and for each particle which has more than one goal position, the average of the goal position is taken before the integration to calculate velocity and position (Figure 3.5). Each cluster gives the following term

$$\triangle v_i = \alpha \frac{g_i^c(t) - x_i(t)}{h} \tag{3.25}$$

to all the particles contained in it, where $g_i^c(t)$ is the goal position of the particle for cluster c.

Plasticity

The idea behind plasticity is to retain the deformed shape when the applied force exceeds certain threshold. The algorithm to perform linear deformation can be extended to simulate plasticity. According to the equation (3.16) we obtained a symmetrix matrix $S = R^T A$ which stores the deformation of the particles before any rotation. The plastic deformation can be store in matrix S^p which is initialized to the identity

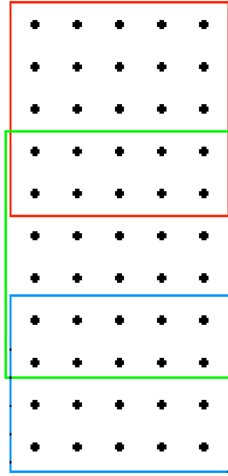


Figure 3.4: *Overlapping regions sharing the particles.*

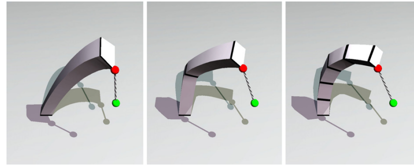


Figure 3.5: *Clustering with different number of clusters (Muller et al. (2005)*

matrix I. The plasticity state matrix is only changed when actual deformation $||S-I||_2$ (squared determinant) exceeds the given threshold c_{yield} which means lesser the value of c_{yield}, the lesser the amount of force is required for an object to undergo plasticity (Figure 3.6). The plasticity matrix S^p can be updated at each time step as:

$$S^p \leftarrow [I + hc_{creep}(S - I)]\, S^p \tag{3.26}$$

where c_{yield} and c_{creep} controls the plasticity and h is the time step.

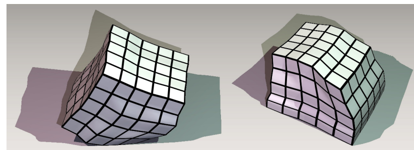


Figure 3.6: *Plasticity presented by Muller et al. (2005)*

Chapter 4

Design and Implementation

The main motive behind the development of this project was to create an efficient system which can simulate soft body deformation of an object efficiently and can be extended easily for furture improvements.

4.1 Design

The Figure 4.1 is the complete class diagram of the implementation.

4.1.1 Architecture Description

The important classes are as follows:

Mesh3D is a subclass of ngl::Obj class which is responsible for loading and drawing a wavefront .obj file. It takes the positions of the vertices of a mesh and assign particles at those positions. As there is no information in the paper on how to distribute the particles on a surface mesh this assumption of allocating the particles at the position of vertices was taken.
It calculates the bounding box and sphere around the particles for various purposes like clustering, collision detection and optimization.
It is also responsible for calculating the overlapping bounding boxes to

Figure 4.1: *The complete class diagram.*

implement clustering. For now clustering can only be done on individual axes like x, y and z.
Finally it redraws the mesh after every time step by updating the old vertex positions.

BoundingRegion is subclass of ngl::BBox which contains the functionalities to calculate the the closest point and squared distance for collision detection and response.
It is also has the functionality to determine whether it contains a particle or not.

Particle class is one of the most important classes. According to the paper an object is represented as a set of particles, it can be a point or can also be represented as a tiny sphere. To keep the implementation simple the particles are represented as small spheres. These particles not only have position (center of the sphere), velocity and radius but also have the original, goal and new position paramters along with mass which is essential in calculating the center of mass to perform shape matching. It can also store multiple goal positions in case it is shared among different clusters so that the average of the goal positions can be take before integration step as described in **section 3.3.3.4**
Every particle is updated individually at each time step. According to equations (3.18) and (3.19), the acceleration is calculated to get the velocity and position along with the external forces like gravity. And after every update, a check is performed to detect and response to collisions with the environment and other deformable objects. (Figure 5.4 showing particles of a mesh)

Cluster is the subclass of BoundingRegion class, it performs the check whether it contains the particles or not and if it does then they are stored to perform shape matching on them. All the pre computations mentioned earlier like x_i^o, q_i and the matrix A_{qq} are performed and at every time step it is updated to perform two important steps:

FOR EACH PARTICLE

Calculate external forces like gravity

Update goal position

DeformableBody is the class representing the soft body. It takes the parameters like emit position and mesh name to create and draw the input surface mesh. Just like cluster class it also needs to perform the pre computations for shape matching of the particles associate to it when clustering is not enabled and if clustering mode is enabled then it is responsible to update every cluster.
It is also responsible for performing integration on each particle at every time step and this step is independent of enabling and disabling of cluster mode.

FOR EACH PARTICLE

if *clustering is disabled,* ***then***

Calculate external forces like gravity

Update goal position

else

update cluster

integrate

update mesh

Mat3 is a simple $3x3$ matrix class for not only performing regular matrix opertations but also sligtly adavanced ones also. Mat3 is capable of calculating the rotation matrix $R = A_{pq}S^{-1}$ mentioned in equation (3.16) of Chapter 3 which is required at each step of the implementation. Calculation of rotation matrix is slightly complicated since it involves the computation of square root of the matrix $S = \sqrt{A_{pq}^T A_{pq}}$. So to get it, we need to calculate the polar decomposition by diagonalizing the matrix $A_{pq}^T A_{pq}$ using Jacobi rotations.
It also helps in computing the inverse of the matrix by calculating the eigen values and eigen vectors because the conventional way of calculating the inverse does work well in case of linear deformations of arbitary

meshes.
After calculating the diagonal matrix it is easy to calculate the matrix S by taking the square root of its values.

ShapeMatching is the class performing all necessary implementation explained in Chapter 3. It just requires a set of particles on which we want to perform the shape matching and it perform the required steps to calculate the goal position depending on the selected deformation mode. It calculates the original and actual center of masses x_{cm}^o and x_{cm} and relative coordinates p_i and q_i and matrices R, A_{pq}, A_{qq} and A for default and linear mode and $\widetilde{q_i}$, $\widetilde{R}$, $\tilde{A_{pq}}$, $\tilde{A_{qq}}$ and $\widetilde{A}$ for quadratic mode. As we know from section 3.3.3.3 of chapter 3 that quadratic deformation requires as large as $3x9$ and $9x9$ matrices. Matrices upto $3x9$ can be calulated using conventional two dimensional arrays as they do not involve any complicate calculation but the matrix $\tilde{A_{qq}} \in \mathbb{R}^{9x9}$ and we need to calculate it's inverse for quadratic deformation. In order to calulate the inverse of this matrix, boost matrix libray was used which does so by computing LU decomposition.

Utilities is the class responsible for storing the values from user interface events and also for passing them to the appropriate classes where this data is required. Considering the fact that this data is consistent throught the implementation or in order words it cannot be changed apart from user interface this class has been designed as a singleton.

Scene is the class which sets up the environment for simulation. It takes the user input to add an object randomly and maintains a list of objects on the scene it then updates each DeformableBody at each time step and checks whether it is colliding with another DeformableBody or not through DeformableBodyOctree. To make the system more efficient the each DeformableBody is added by a DeformableFactory.

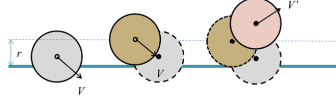


Figure 4.2: *The sphere plane collision.*

Interface class is mainly responsible for receiving the user interface events related to mesh and simulation and storing their values in the class Utitlities so that they can be made available.

Renderer class is responsible for handling the drawing routines which is quite evident from its name. It also update the Scene at each time step. The time step is also corrected using the approach suggested by Fiedler (2006).

4.2 Collision Handling

Collision handling has been the most trickest of all the implementations. It was the one which took a lot of time because of the fact that paper does not provide any information regarding the collision handling of an object deforming through a set of particles. As mentioned in the paper that there is no connectivity information among the particles which makes it more complicated to understand that how collision detection and importantly collision response should work.

4.2.1 Collision Detection

Collision detection with the environment such as walls is pretty straightforwad. A simple sphere to plane collision detection was performed. As we already know that the particles are assumed to be small spheres with certain position, velocity and radius. These particles are checked against a plane represented as wall(a, b, c, d) (Figure 4.2).

One of the most complicated part was to detect collisions among two

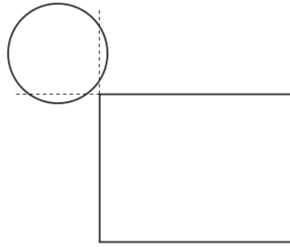


Figure 4.3: *The sphere box collision (Ericson, 2005).*

soft bodies. It was really difficult to understand how one can detect collisions when the particles do not have any connectivity information ruling out the option of using the conventional methods. Finally after a lot of considerations and trying out different options a very simple sphere to box collision detection method was used. Where collision among every particle is checked against the bounding box of other (Figure 4.3). It is not only simple to detect but it greatly improves the efficiency of the entire system compared to the particle against particle collision which was initially used and discarded because of these reasons:

(1) In order to detect proper particle to particle collisions the mesh should be sampled enough that the particles should not penetrate the other object from the space between two particles
(2) And if we sample the mesh enough so that there is no chance of missing the collisions then the system becomes too slow to be real time.
(3) And lastly if we think of detecting the collision among the particle of one object with the edge or the face of the other object then it is not only slow but also breaks the notion meshless deformation where there is no connectivity information among the particles.
Another important issue not mentioned in the paper is, what if two clusters of the same surface mesh collides with themselves as sphere to bounding box collision detection does not work efficiently may be because there are particles shared among the clusters.

4.2.2 Collision Response

Responding to the collisions was once again a very time consuming issue. Getting correct collision response for wall was simple.

But getting proper collision response for two colliding soft bodies was very challenging. Though still collision response is not efficient enough but when applied with correct parameters it does work quite well.

4.2.3 Spatial Partitioning

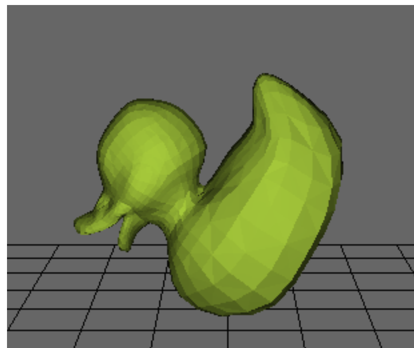
Even after taking a very simple approach for detecting the collision among the two soft bodies, still it was not that efficient to simulate objects more than a certain amount. To over come this issue Octree spatial partitioning was used which recursively subdivides the given three dimensional space into eight octants. There were certain issues which had to be considered before applying this as while detecting collisions among objects, the collision of a particle was checked against the bounding box, so to check the position of the particles in the octree bounding spheres of the objects were taken into account. The results of this implementation will be discussed later.

Chapter 5

Applications and Results

As mentioned earlier that apart from basic algorithm there are five other extensions and for the purpose of this thesis four out of five have been implemented.
Here are the few examples of the various implementations.

5.1 Basic



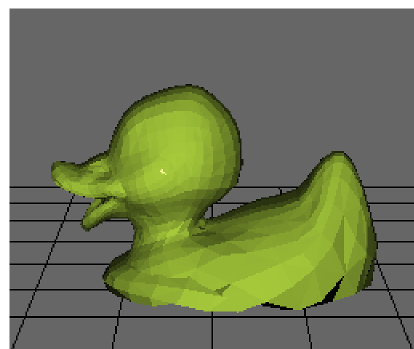


Figure 5.1: The images are showing the basic algorithm for two different alpha values. For the image above the $\alpha = 1$ and for the image below $\alpha = 0.6$.

And we already know that α controls the stiffness and $\alpha = 1$ gives almost rigid object.

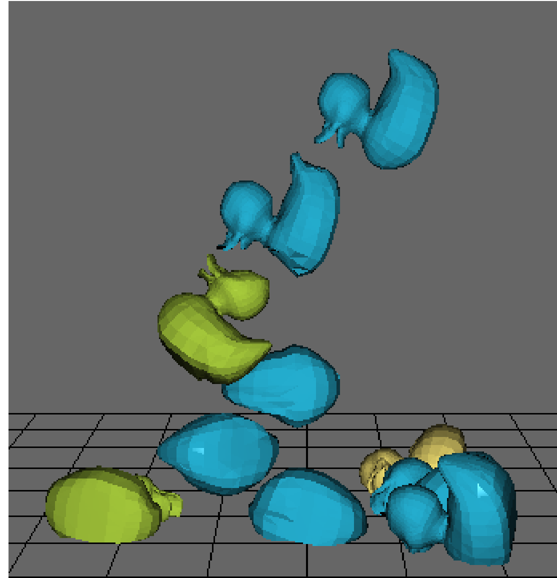


Figure 5.2: A total of ten duck models are simulated having 15,220 particles.

5.2 Linear

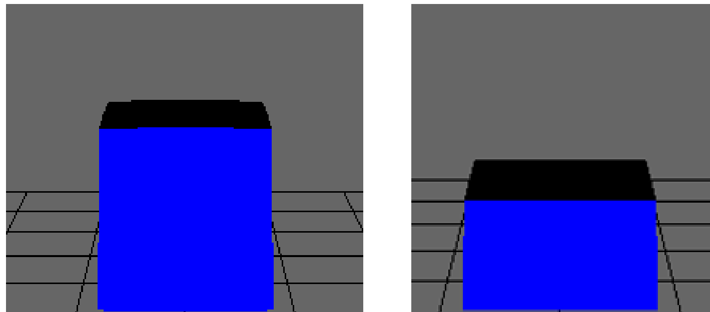


Figure 5.3: The linear deformation is shown using the a cube where left is the state before collision with the wall and right is the deformation obtained after it with is controlled by β.

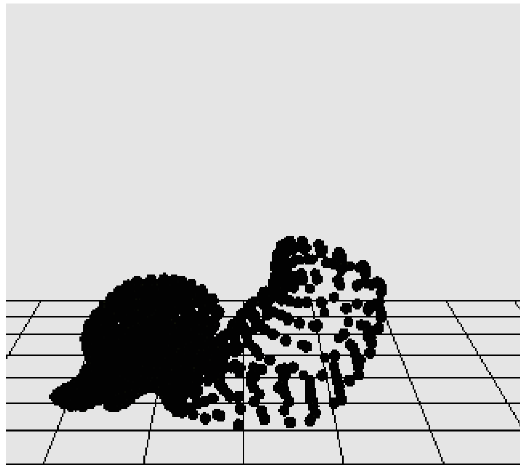


Figure 5.4: The linear deformation of a duck is represented by particles only.

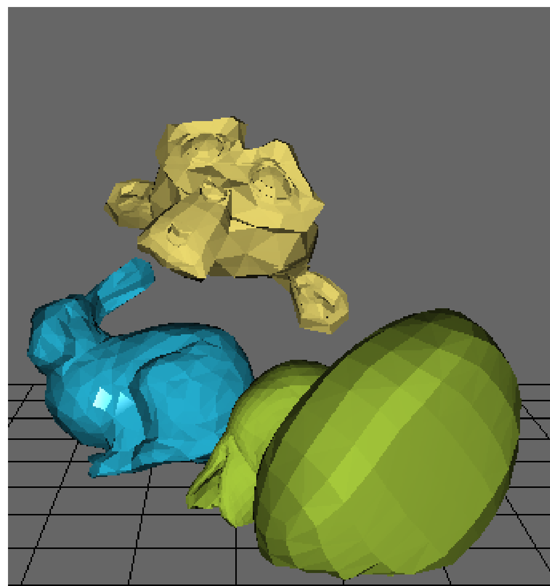


Figure 5.5: The deformation of different objects in linear mode.

5.3 Quadratic

Again the output of quadratic deformation can be best representated by particles on a cube.

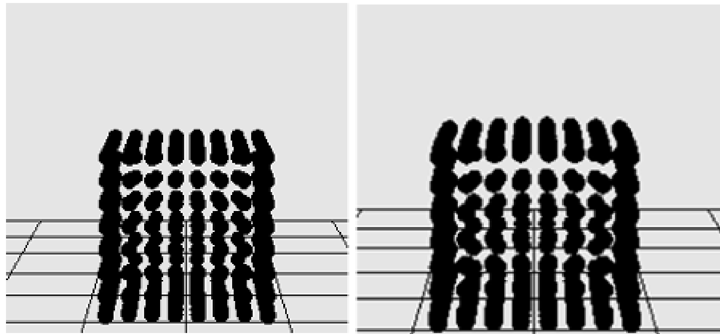


Figure 5.6: The left side of the image is showing the state before collision and right one is after collision.

It can be noted that the cube is slightly curved due to quadratic deformation.

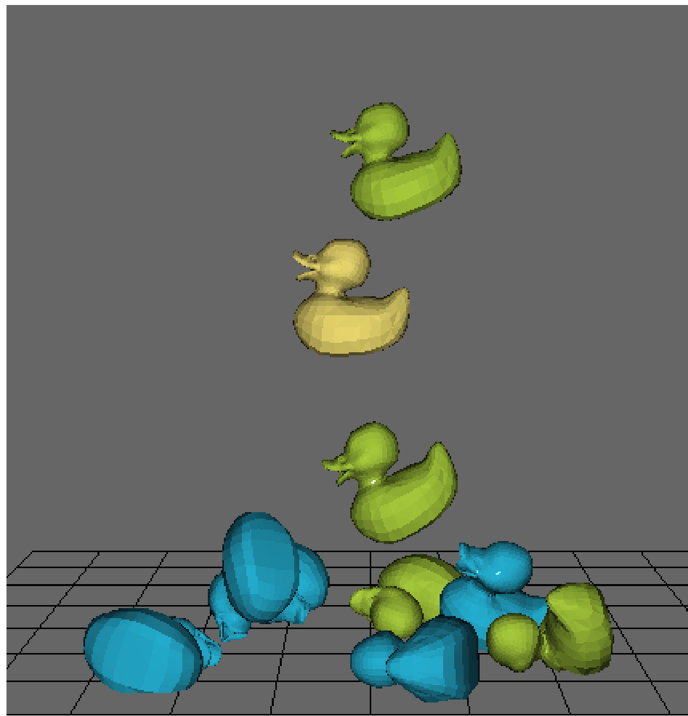


Figure 5.7: Multiple duck models are simulated using quadratic deformation.

5.4 Clustering

Clustering helps in increasing the freedom of deformation. Here are few results obtained through it.

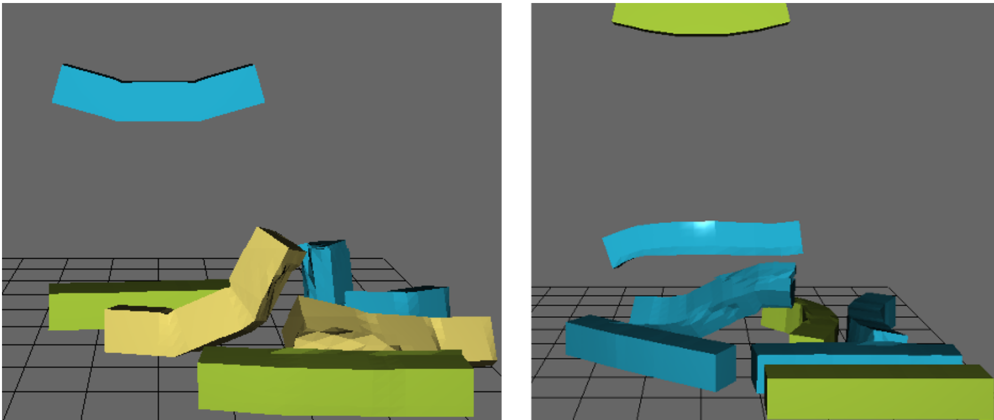


Figure 5.8: The image on the left deformes a bar using three clusters and the right one is using five clusters.

With the increase in the number of clusters the degree of deformation is also increase and it becomes more smooth.

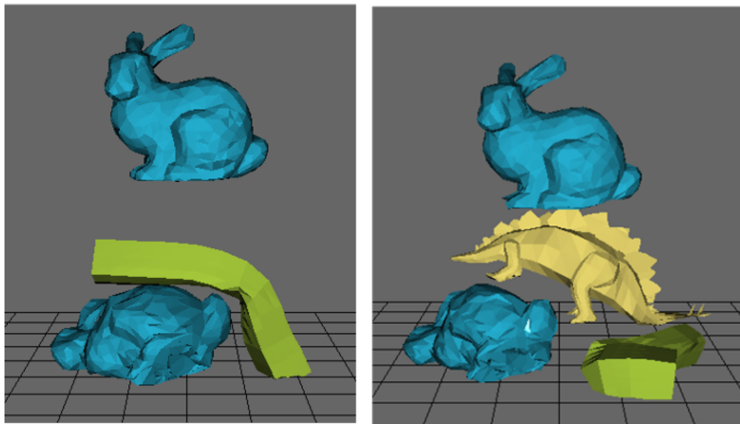


Figure 5.9: Another example showing the deformation using different number of clusters and different types of objects.

Below is the car model deformed with various modes and number of clusters.

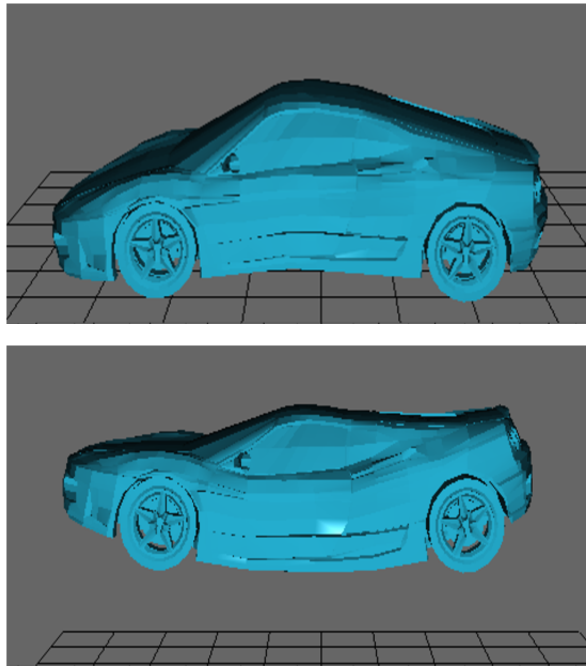


Figure 5.10: The above model is deformed with two clusters and the below one is deformed with five clusters. Both cases are deformed under default deformation.

Note that the above car model has more than 10,000 particles.
One more example of an old car deforming under linear deformation is shown below.

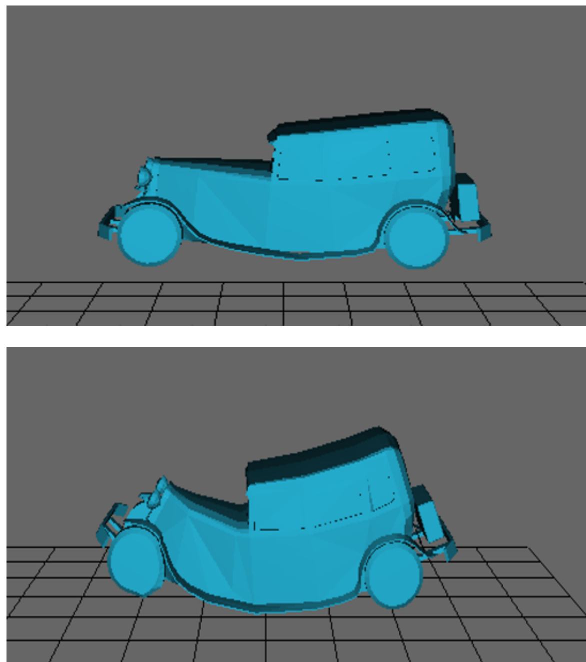


Figure 5.11: The above model is deformed with two clusters and the below one is deformed with five clusters. Both cases are deformed under linear deformation.

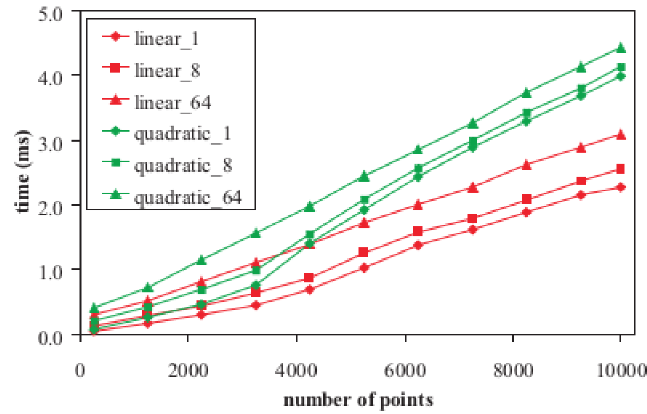


Figure 5.12: A comparision the time complexity of each model with the number of particles simulated by Muller et al. (2005).

Chapter 6

Conclusion

A lot of time was spent in researching about the various implementations followed to perform soft body deformation. Meshless shape matching is definitely one of the fastest algorithm to simulate soft bodies in real time. It can efficiently be included in almost all the environments requiring the real time interaction such as games and virtual surgery. This approach can successfully be implemented for a volumetric tetrahedral mesh.

6.1 Summary

Initially the goal of implementing the entire paper was set but unfortunately one could not be achieved due to the time constraints that it Plasticity. Rest of the achieved goals are as follows:

- Successfully implemented the basic algorithm.
- Successfully implemented four out of five extensions given in the paper.
- A very basic collision detection and response methods have been implemented to avoid collisions among soft bodies and environment and also among soft bodies themselves.
- A proper user interface is also provided.

6.2 Known bugs and issues

There are some known bugs which still need attention:

- Sometimes Quadratic deformation mode behaves unexpectedly and does not produce any simulation resulting the disappearing particles and in turn mesh.
- Because of unexpected behaviour of quadratic deformation the clustering can not be done with this mode. That is why there are only two modes available for clustering.

The most important limitation is due to the lack of efficient collision detection and spatial partitioning methods it is really difficult to simulate large amount of soft bodies. Soft bodies upto ten to twelve can be simulated efficiently with nearlly 15,000 to 20,000 particles. It is recommended to use low resolution mesh if more number of soft bodies are needed to be simulated.
There is one more drawback, if the emit position of the soft bodies is same then the octree spatial partitioning fails to optimize the system.

6.3 Future work

The following are the task which should be done to improve the current implementation.

- Implement efficient collision detection and response methods.
- Implement plasticity as mentioned in the paper.
- Implement a plugin or a tool for any 3D software package.
- GPU implementation can be done to improve the over all efficiency of the system.
- Algorithm can be manipulated to perform fracturing, tearing and cutting operations.

Chapter 7

References

anon, no date. Technical Game Development. http://web.cs.wpi.edu/~imgd4000/d07/slides/Physics.ppt [Accessesd 07 Aug 2013].

Ericson, C., 2005, Real - time Collision Detection. Second Edition. Burlington USA : Morgan Kaufmann Publishers.

Irving, G., Teran, J., and Fedkiw, R. 2004. Invertible finite element for robust simulation of large deformation. In Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2004, Aire-la-Ville, Switzerland, 131-140.

Muller, M., Dorsey, J., Macmillan, L., Jagnow, R., and Cutler, B. 2002. Stable Real-Time Deformations. In Proceedings of ACM SIGGRAPH Symposium on Computer Animation 2002, 49-54.

Muller, M., and Gross, M. 2004. Interactive virtual materials. In of Graphics Interface 2004, Ontario, Canada, 239-246.

Muller, M., Heidelberger, B., and Gross, M. 2005. Meshless Deformations Based on Shape Matching. In Proceedings of ACM SIGGRAPH 2005, New York, USA, 471-478.

Muller, M., McMillan, L., Dorsey, J., and Jagnow, R. 2001. Real-time simulation of deformation and fracture of stiff materials. In Proceedings of Eurographics Workshops, Eurographics Association 2002, 99-111.

Rivers, R. R., and James, D. L. 2007. FastLSM: fast lattice shape

matching for robust real-time deformation. In Proceedings of ACM SIGGRAPH 2007, New York, USA.

Parker, E. G, and O Brien, J. F. 2009. Real-time deformation and fracture in a game environment. In Proceedings of ACM SIGGRAPH 2004/Eurographics Symposium on Computer Animation 2009, 165-171.

Sederberg, T. W. and Parry, S. R. 1986. Free-form Deformation of Solid Geometric Models. In Proceedings of Communications of ACM 1986, 20(4) 151-160.

Voigt, H., Flower D., and Weisser, D. 2006. Group Report on Meshless Deformation, University of Auckland.