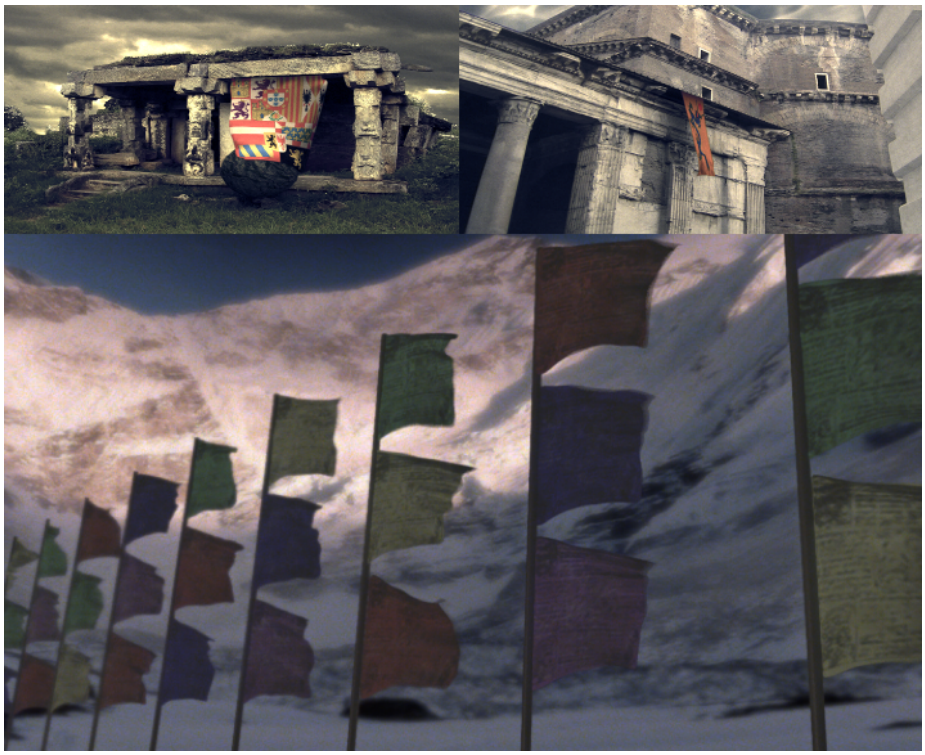
Master Project
C++ Cloth Simulation

Luis Pereira
MSc CAVE 09-10

August 15, 2010



Abstract

This thesis presents the concepts behind creating a successful cloth simulation tool completely integratable into a production pipeline. Its features include the self-collisions, collisions with static objects, multiple instances of cloth as well as the possibility of using any object as the focus of the simulation. All of the data behind this application is passed via texture maps to help artistic-minded users set different values along the mesh.

The results achieved include successful integration scenes, several materials accomplished by changing parameters such as stiffness and damping, and interaction of the cloth with OBJ file sequences.

Contents

List of Tables

List of Figures

1 Introduction

Cloth has been of the more researched subjects in the computer graphics field. Its use in motion pictures has been extensive varying from papers flying and towels hanging on wires, to highly detailed garments with folds and wrinkles.

The purpose for choosing this topic comes from a previous development of a cloth simulator that the author felt was unfinished. This project is set to refine the principles of the previous tool making it more robust and flexible allowing it to be integrated into a production pipeline with minimum effort for the user.

This thesis describes the process of implementation of the tool, the concepts behind it and the results achieved, providing the reader with enough knowledge to produce their own cloth simulator.

1.1 Objectives

The following objectives list was set at the beginning of the project and its success will be analysed at the end of the thesis.

- **Complete integratable tool** Perhaps the main objective of this tool was make it flexible enough to be integrated into any production pipeline. The application must be able to handle several input data and export the simulated files into a format recognized by most packages used in the industry. Creating a complete scene with simulated cloth integrated will sustain the success of the tool.
- **User defined cloth mesh** To achieve the previous item it is crucial that any object can be converted into cloth.
- **Cloth Self-Collisions** To create a realistic cloth simulation self collisions are imperative since they produce real life interactions that exist in cloth, therefore the simulation should mimic them.

1.2 Thesis Overview

This thesis is divided in the following way:

Chapter 2 - Previous Course Work Describes research done in this field on previous course work.

Chapter 3 - Previous Work Reviews the work of relevant authors in the field, with special focus on the authors that influenced this project. The reader is also referred to publications that survey the research on cloth simulation in the computer graphics field.

Chapter 4 - Technical Background Presents details on the theory behind this implementation.

Chapter 5 - Implementation Details on how the concepts from the previous chapter were implemented in the application. This section presents to the reader the algorithms used to achieve it. Includes as well a description of the interface created for a 3D package to allow the user to distance himself from the

code.

Chapter 6 - Results Presents the results achieved with this tool and the different possible usages of the simulate cloth.

Chapter 7 - Further Work Describes future implementations to improve the tool.

1.3 Applications Used

To produce this tool and its results the following applications were used:

- **QtCreator** - The IDE used to implement the application.
- **Doxygen** - Used to produce the documentation of the tool.
- **SideFX Houdini** - Main 3D package used to create the user/tool interaction as well as the rendered scenes.
- **Autodesk Maya 2008** - Given the author's background this was the prefered application used for modelling and to handle FBX file data.
- **Foundry NukeX** - Used to do the integration of the cloth into real scenes.
- **Gliffy** - On-line resource to produce the class diagrams.
- **Gimp** - Open-source 2D image editing software used to adjust some of the texture maps and the images seen in the thesis.
- **KDEnLive** - Open-source video editing software used to produce the final videos.

1.4 Libraries Used

This application could not have been achieved without the use of external libraries. The following list describe the ones used and what was its purpose:

- **NCCA Graphics Library** - Open-source library mainly developed by Jon Macey used as the backbone of the application, being used to hold several structures such as vectors, textures, amongst many others.
- **Boost Library** - Open-source c++ library used for parsing data, optimised for-loops and string handling.

2 Previous Course Work

The first introduction to cloth happened during the CGI Techniques course where a cloth simulation tool was developed. This tool can be considered as a proof of concept when compared to the one created as the Master's project therefore some of its characteristics were included.

Amongst other features, the previously developed cloth simulation included the following:

- Mass Spring Cloth Model
- Runge Kutta 4th Order Explicit Integration
- Collisions against multiple bounding spheres
- Multiple instances of cloth allowed
- User defined scene files

More important than the written code, the knowledge gained during the development proved itself as invaluable when approaching this project. The features of the tool were analysed so that the weak ones would reveal what needed to be changed and the positives what needed to be improved. The following table shows the result of this analysis:

To Keep	**To Discard**
Runge Kutta Integration	Cloth object mesh limitation
User defined scene files	Lack of export method
Collisions with static objects	Lack of self collisions
	Constraints defined by manually picking particles
	Lack of communication between 3D packaged and the tool

Table 2.0.1: Analysis on the features to keep and to discard.

By going back and studying the previous implementation, the pitfalls encountered while developing it were avoided and all of the good features were improved to produce a better tool.

3 Previous Work

The research on simulation of cloth in the computer graphics fields extends back to the 1980's with several authors focusing on this particular subject since then.

Most authors refer the reader to the paper (Ng and Grimsdale, 1996) and the book (House and Breen, 2000) because they provide a detailed history on the development of cloth simulation from its early beginnings until 1995 in the case of (Ng and Grimsdale, 1996) and 1999 in (House and Breen, 2000). These references explain the different techniques available in this field, such as the geometric and physical techniques, and analyse in detail the work of selected authors.

A brief history is now described mentioning the most relevant papers for the production of this project or the ones that were relevant to them.

The first source that most papers refer to is the work by (Terzopoulos et al., 1987) where cloth was treated as a rectangular mesh using a semi-implicit integration to update its position and using the theory of elasticity to animate it. (Breen et al., 1994) presented a new view, proposing a particle-based approach where the cloth was simulated using real life properties using the Kawabata measuring system. This research focused on producing static images. (Provot, 1995) introduced a massless-spring system to model cloth using a system of particles and updating them using the explicit Euler integration. In 1997, (Provot, 1997) focuses on collisions and self-collisions of the cloth.

Figure 3.0.1: The cloth created by Breen (Breen et al., 1994).

One of the most referenced papers is (Baraff and Witkin, 1998) since it introduced an implicit integration model allowing large time-steps while maintaining the stability of the cloth. The computational time of this algorithm is very fast however it became faster when in (Desbrun et al., 1999) it was extended increasing its speed while maintaining similar results. In (Volino and Magnenat-Thalmann, 2000) the simulation of cloth self collisions is extended using a implicit integration method and in (Volino and Magnenat-thalmann,

2001) several integration methods used in cloth simulation are compared.

Figure 3.0.2: (Baraff and Witkin, 1998) introduced an implicit method allowing large time-steps.

In 2002, (Bridson et al., 2002) focused on the simulation of self collisions of the cloth using an AABB tree to optimise the neighbour search. Their research continued and in (Bridson et al., 2005) the focus was on maintaining folds and wrinkles of the cloth when in contact with rigid objects.

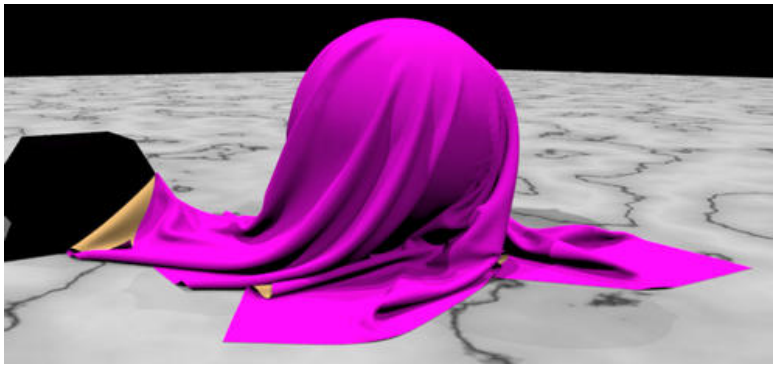


Figure 3.0.3: (Bridson et al., 2005) focused on maintaining folds and wrinkles of the cloth.

(Villard and Borouchaki, 2005) describes an adaptive-mesh method to simulate cloth allowing low definition meshes to be simulated and have it increase definition as needed. A quad-tree is used to subdivide the mesh. In 2009, (Selle et al., 2009) presented a new method to handle high resolution cloth objects while keeping a collision history. This method allows for simulations of cloth objects with up to two million particles.

These are some of the most relevant papers for the production of this tool. As mentioned, for a more in-depth look at the history of cloth simulation from 1986 until 2000 (Ng and Grimsdale, 1996) and (House and Breen, 2000) are the recommended sources as well as the articles mentioned above.

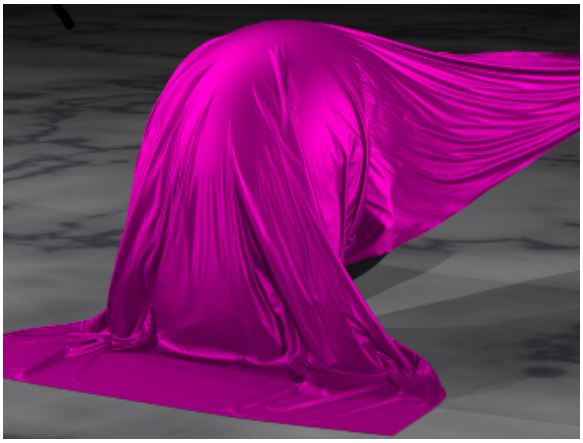


Figure 3.0.4: (Selle et al., 2009) allowed for very high resolution models up to two million polygons.

4 Technical Background

This section intends to provide the reader with theoretical information regarding the cloth simulation. Several techniques will be approached and most of them developed in the final tool.

4.1 Cloth Model

From the several methods to model cloth described in (Ng and Grimsdale, 1996), the one used in this application is the mass-spring model. Initially described in (Provot, 1995) it provides a robust way to simulate cloth and it is been widely used in several developments of cloth in computer graphics.

There are three main sections on this method: how the mesh is created, how many types of springs exist and how they are updated. All these methods are described below.

4.1.1 Mesh Generation

When creating cloth objects two choices are available: use a procedurally created quadrilateral grid as in (Provot, 1995) or allow for a user defined triangulated mesh as in (Selle et al., 2009).

Using the first method allows for quick generation times and to use the principle of a 2D grid in a 3D environment. The mesh is easily transformed from local to world space coordinates and therefore intersections with primitives such as spheres is much easier and faster. This method was used in the previous implementation of a cloth simulator.

The second method allows for any object to be converted into cloth. The spring generation, as it will be described in the next section, requires more computation time, but the final outcomes will be much easier to integrate into a production pipeline since any asset can be simulated.

4.1.2 Spring Generation

Similarly to the mesh generation there are two types of spring generation depending on the type of mesh, quadrilaterals or triangles.

If the mesh is composed by quadrilateral polygons then as shown in (Provot, 1995) three types of springs are available and will be defined in the following way:

Considering that i and j are particles, or masses, as (Provot, 1995) explains, there will be three types of springs created:

- spring linking masses $[i, j]$ and $[i+1, j]$, and masses $[i, j]$ and $[i, j+1]$, will be referred to as "structural springs"
- spring linking masses $[i, j]$ and $[i + 1, j + 1]$, and masses $[i + 1, j]$ and $[i, j + 1]$, will be referred to as "shear springs"
- spring linking masses $[i, j]$ and $[i+2, j]$, and masses $[i, j]$ and $[i, j+2]$, will be referred to as "flex springs"

Each of the springs have a different purpose that is described in (Kieran et al., 2005):

- The structural springs handle the extension and the compression of the cloth
- The shear springs handle shear stresses
- The flexion springs handle bending stresses

In this scenario every particle fully connected (with all the possible springs) has 12 springs. The particles on the edge of the cloth have 8 and the ones on the corners have 5. This means that when the simulation runs, if there are constraints in any of the particles on the edge of the cloth, especially the corners, parameters such as the stiffness will be very low. To compensate for those low values, the parameters of the particles on the edge are multiplied by 1.5 and on the ones on the corner of the cloth multiplied by 2.4. This way each particle will have the same amount of overall force making the cloth behave properly.

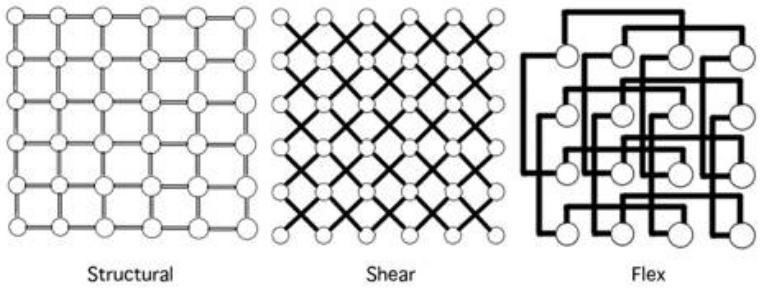


Figure 4.1.1: Different types of springs in procedurally generated (Liberatore, n.d.).

In case the cloth is generated from a modelled triangulated mesh or user-defined object, (Selle et al., 2009) suggests the use of two types of springs: edge springs and bend springs. The first ones would handle the extension and compressions of the cloth and would use all of the edges of the mesh. The second ones would handle bending stresses and connect non-connected particles of adjacent triangles.

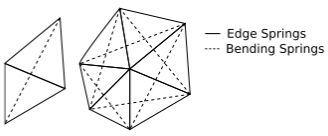


Figure 4.1.2: Different types of springs in procedurally generated (Selle et al., 2009).

4.1.3 Forces applied to springs

Particles in the cloth mesh will be connected by a series of springs and their oscillating motion will create the smooth movement that cloth is known for.

The model used to simulate the springs is a spring-damping system where the stiffness force will make the spring oscillate realistically, but endlessly, while the damping will make it stop in a natural and smooth way.

The stiffness of the spring is based on Hooke's Law, where k_s is the coefficient for the damping and $\vec{x}$ is the exceeding spring when compared to its natural equilibrium length:

$$\vec{F_{stiff}} = -k_s * \vec{x} \tag{4.1.1}$$

The damping is found by negating the velocity of the particle. In this example k_d is the damping coefficient and $\vec{v}$ is the difference between the velocities of the particles existing in the spring:

$$\vec{F_{damp}} = -k_d * \vec{v} \tag{4.1.2}$$

4.2 Integration Methods

Integration methods are used to calculate the position of objects from frame to frame. There are two types of integration methods, explicit or implicit.

Explicit integration methods will calculate the position of the next frame based on the position of the current one while the implicit integration methods will calculate the position on the current frame based on what position the object would have on the next frame.

The two methods available to the user belong on the explicit category: Euler Integration and Runge Kutta 4th Order Integration.

4.2.1 Euler Integration

Euler integration is one of the most basic integration methods to implement and if the time-step is small enough it will not have instability issues.

It is based in Newton's second law of motion and uses the notion of derivatives to find the velocity (whose derivative is acceleration) and the position (whose derivative is velocity).

To find the next position of an object the following algorithm is implemented:

$$\vec{a} = \vec{F}/m \tag{4.2.1}$$

$$\vec{v} = \vec{v_0} + \vec{a} * \Delta t \tag{4.2.2}$$

$$\vec{x} = \vec{x_0} + \vec{v} * \Delta t \tag{4.2.3}$$

This integration has some problems associated with it. First it is not completely accurate providing worst results as the simulation time goes by. Second it is quite unstable and a very small time-step has to be given to achieve relatively stable results. (VDocs, 2009) is a very good resource to compare different integration methods under different scenarios. Figure 4.2.1 and 4.2.2 show the stability and accuracy of Euler integration when compared with other integration methods.

"Reading left to right along the time-step increments, the RK4 integrator's results become unacceptable first at a time-step of 0.009 seconds (remember

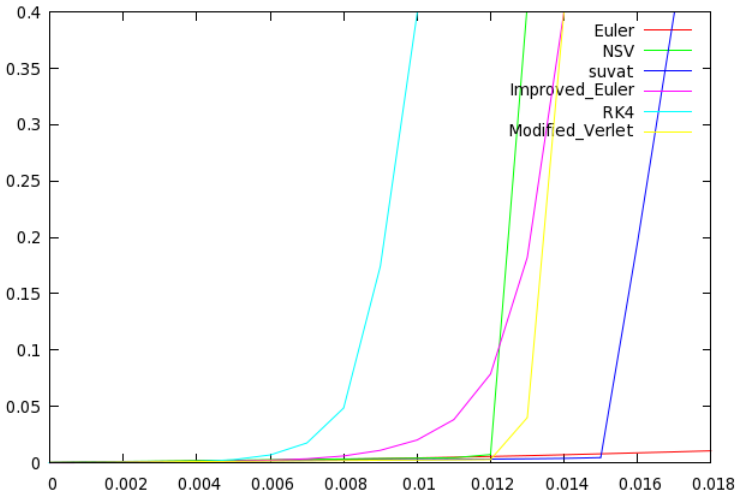


Figure 4.2.1: Stability of different integration methods (VDocs, 2009).

that this actually means a 0.036 second time-step due to the performance handicap of the RK4 method). The NSV is next, followed by the modified verlet and improved euler methods. The suvat method is second to last to become unacceptable and the euler method is acceptable even with a 0.020 second time-step.", (VDocs, 2009).

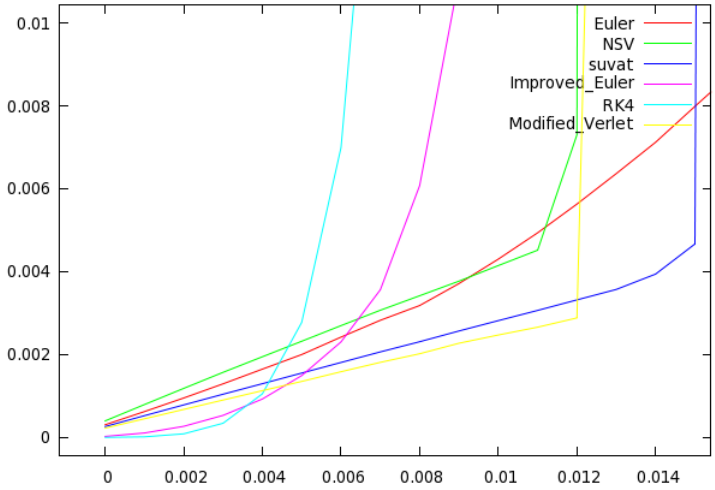


Figure 4.2.2: Accuracy of different integration methods (VDocs, 2009).

"Looking at a close-up of the results, it becomes apparent where the strengths of higher order integrators lie. At 0.002, the RK4 (actual time-step 0.008) and improved euler (actual time-step 0.004) methods beat out all of the other methods. Of the first order methods at 0.002, from most to least accurate they are: modified verlet, suvat, euler, and nsv.", (VDocs, 2009).

4.2.2 Runge Kutta 4th Order

Being one of the most stable and accurate integration methods, Runge Kutta 4th Order (RK4) was implemented. RK4 performs four Euler calculations throughout the time-step $\triangle t$ to find the final position and velocity of the particle. By using this method, the four calculations of Euler will make it a much more stable integration method.

Find the initial position and velocity of the particle:

$$k_1 = \triangle t f(x_n, y_n) \tag{4.2.4}$$

Find the position and velocity of the particle at $\frac{h}{2}$:

$$k_2 = \triangle t f(x_n + \frac{1}{2} \triangle t, y_n + \frac{1}{2} k_1) \tag{4.2.5}$$

$$k_3 = \triangle t f(x_n + \frac{1}{2} \triangle t, y_n + \frac{1}{2} k_2) \tag{4.2.6}$$

Find the position and velocity of the particle at h:

$$k_4 = \triangle t f(x_n + \triangle t, y_n + \triangle t) \tag{4.2.7}$$

Finally the calculated values will be averaged and the final position and velocity is found:

$$y_{n+1} = y_n + \frac{1}{6} k_1 + \frac{1}{3} k_2 + \frac{1}{3} k_3 + \frac{1}{6} k_4 + O(h^5) \tag{4.2.8}$$

4.3 Collision Detection

Collisions are vital to produce a realist cloth simulation since they give it a sense of reality than any shader or texture could never do. To find the collisions that exist in the cloth, a point-triangle intersection test is used as suggested in (Bridson et al., 2002). Note that according to the same author, the cloth will have a thickness h, so collisions will exist if the two primitives are closer than that thickness.

4.3.1 Point-Triangle Intersection

The cloth is composed by a series of triangles, so collision tests are executed between a particle and triangle (excluding triangles where this particle exist). The research to understand the math behind this method involved not only checking the formulas in (Bridson et al., 2002) but also decompose them with the help of (Ericson, 2005).

The complete math behind the following formulas can be found in appendix A.

Even though understanding the math behind this process is not vital to implement the algorithm, it is very helpful to find any problems that might be occurring.

In the examples shown below $\vec{x_{ij}}$ represent $\vec{x_i} - \vec{x_j}$ and h represents the thickness of the cloth.

To test if the point $\vec{P}$ is in danger of colliding with a triangle composed by the points $\vec{x_1}$, $\vec{x_2}$, $\vec{x_3}$, the first test is if the point is close enough to the triangle by checking if

$$|\vec{x_{P3}} \cdot \hat{n}| < h \tag{4.3.1}$$

If it is the collision point is calculated by finding the barycentric coordinates w_1, w_2, w_3 using Cramer's rule (more detail on how to use Cramer's rule can be found in appendix A).

$$\vec{e_0} = \vec{x_2} - \vec{x_1} \tag{4.3.2}$$

$$\vec{e_1} = \vec{x_3} - \vec{x_1} \tag{4.3.3}$$

$$\vec{e_2} = \vec{P} - \vec{x_1} \tag{4.3.4}$$

$$\begin{bmatrix} \vec{e_0}.\vec{e_0} & \vec{e_1}.\vec{e_0} \\ \vec{e_0}.\vec{e_1} & \vec{e_1}.\vec{e_1} \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} \vec{e_2}.\vec{e_0} \\ \vec{e_2}.\vec{e_1} \end{bmatrix} \tag{4.3.5}$$

$$u + v + w = 1 \tag{4.3.6}$$

The collision point P_c will be calculate using the following formula:

$$P_c = w_1\vec{x_1} + w_2\vec{x_2} + w_3\vec{x_3} \tag{4.3.7}$$

If the barycentric coordinates are all within the interval $[-\delta, 1 + \delta]$ where δ is h divided by the characteristic length of the triangle the point is close to collide. There was some doubt on what the author meant by the "characteristic length of the triangle" so an email was sent to him to which he clarified:

"It's not particularly critical what you choose the characteristic length to be. If all the triangles in your mesh are a similar size, just pick one number that scales with the triangle size, otherwise something like the sum of the edge lengths is fine. This is not a terribly exact way of working with collisions, but is adequate to avoid some problems with rounding error."

4.4 Collision Response

Generating a realistic contact response will be essential to produce good cloth simulations. Combining both works by Bridson, (Bridson et al., 2002) and (Bridson et al., 2005), the response to cloth-cloth and cloth-static objects are calculated. Before isolating each of these types of collisions it is necessary to explain how the velocities are assigned to all of the particles.

Let w_1, w_2 and w_3 be the barycentric coordinates of the collision point, I the impulse applied to the particles, i be the vertices of the tested triangle and P the point being tested.

$$\tilde{I} = \frac{2I}{1 + w_1^2 + w_2^2 + w_3^2} \tag{4.4.1}$$

$$\vec{v_i}^{new} = \vec{v_i} + w_i(\tilde{I}/m)\hat{n},\ i = 1, 2, 3 \tag{4.4.2}$$

$$\vec{P_v}^{new} = \vec{P_v} - (\tilde{I}/m)\hat{n} \tag{4.4.3}$$

These set of equations explain how the forces are distributed between the vertices of the triangle and the particle being tested.

4.4.1 Self-Collisions

Once determined that the cloth will collide with itself by using the previously mentioned point-triangle intersection test, the collision point PC_x will be calculated by:

$$\vec{PC}_x = w_1 * \vec{x_1} + w_2 * \vec{x_2} + w_3 * \vec{x_3} \tag{4.4.4}$$

Using a similar method, the collision point velocity PC_v is calculated by:

$$\vec{PC}_v = w_1 * \vec{v_1} + w_2 * \vec{v_2} + w_3 * \vec{v_3} \tag{4.4.5}$$

(Bridson et al., 2002) states that to find the impulse I that will be applied to the particles the relative velocity in the normal direction v_N is found using 4.4.6 and an inelastic impulse I_c in the normal direction is applied using 4.4.7:

$$v_N = \vec{PC}_v \cdot \vec{n} \tag{4.4.6}$$

$$I_c = \frac{mv_N}{2} \tag{4.4.7}$$

4.4.2 Static Objects Collisions

Static object collision response is based on the work of (Bridson et al., 2005) since this paper focuses on collisions between the cloth and external objects. While the cloth-cloth collisions are treated with an inelastic collision response, these are treated with a method described in (Bridson et al., 2005):

The collision point velocity, calculated in the same manner as before, will now be the velocity of the point since it is the only object moving:

$$\vec{PC}_v = \vec{P}_v \tag{4.4.8}$$

The velocity in the normal direction and in the tangential direction are calculated:

$$\vec{v_N} = \vec{PC}_v \cdot \vec{n} * \vec{n} \tag{4.4.9}$$

$$\vec{v_T} = \vec{PC}_v - \vec{v_N} \tag{4.4.10}$$

To produce realistic collisions friction, k_f and the coefficient of restitution $k_{coefRest}$ must be considered. The friction will determine if the cloth is going to try to glue itself to the object while the coefficient of restitution will determine how much of the collision velocity will be assigned to the particle.

$$\vec{v_T} = \vec{v_T} * (1 - k_f) \tag{4.4.11}$$

$$\vec{v_N} = \vec{v_T} * k_{coefRest} \tag{4.4.12}$$

Contrary to what happened in cloth-cloth collisions, the triangle will not be moved. To solve the collision the position of the particle is changed to a safe location and its velocity will be changed to the difference of the tangential and the normal velocity:

$$\vec{P_v} = \vec{v_T} - \vec{v_N} \tag{4.4.13}$$

$$\vec{P_x} = \vec{PC_x} - (\vec{n} * thickness) \tag{4.4.14}$$

4.5 Optimizations

Collision detection is a time-consuming process consuming most of the overall time of the application therefore it is usual to create an optimisation structure limiting the collision tests only against immediate neighbours instead of against the entire mesh.

Two methods are described, the first suggested by (Bridson et al., 2002) and the second by (Kelager, 2006), being this last one the approach used.

4.5.1 AABB Tree

(Bridson et al., 2002) describes the AABB tree as the method used to limit the number of neighbours against which collisions are tested. As described in the paper the structure "is built bottom-up once at the beginning of the simulation using the topology of the cloth mesh. In one sweep we greedily pair off adjacent triangles to get parent nodes, then in a sweep in the opposite direction pair off these to get the next level up, and so on alternating sweep directions until we are left with one root node".

Since real cloth has thickness associated with it, this method will have it as well. To do so, the bounding boxes placed around the triangles will have the its size plus a thickness variable producing more realistic collisions and the possibility to have thicker cloths.

4.5.2 Spatial Hash

The method used in this simulation is described by (Kelager, 2006). Even though the paper relates to a different field, Smooth Particle Hydrodynamics, the neighbour searching algorithm described is quite good, easy to implement and very to perform the neighbour search, decreasing the complexity from $O(n^2)$ to $O(nm)$ where n is the number of particles in the scene and m the number of averaged neighbours found by the algorithm.

The principle behind this algorithm is that a hash is created and neighbouring particles will be placed in the same index in the hash. Obviously these indexes have to be unique otherwise non neighbouring particles could have the same key so the use of prime numbers is suggested.

4.5.2.1 Inserting a particle into the hash Considering that n_H is the size of the hash table and $\vec{r}$ the position of the particle that is going to be inserted, the function to translate the 3D position into a 1D hash key is defined by:

$$hash(\hat{r}) = (\hat{r}_x p_1 \text{ xor } \hat{r}_y p_2 \text{ xor } \hat{r}_z p_3) \bmod n_H \tag{4.5.1}$$

and

$$\hat{r}(r) = (\lfloor r_x/l \rfloor, \lfloor r_y/l \rfloor, \lfloor r_z/l \rfloor)^T \tag{4.5.2}$$

defines the discretized 3D point with a cell size l.

The only unknown variables in the previous formulas are $p1$, $p2$ and $p3$ that are defined by the author as three large primes such as:

$$p_1 = 73,856,093 \tag{4.5.3}$$
$$p_2 = 19,349,663 \tag{4.5.4}$$
$$p_3 = 83,492,791 \tag{4.5.5}$$

(Kelager, 2006) suggests the size of the table to be

$$n_H = prime(2n) \tag{4.5.6}$$

where $prime(x)$, $x \in \mathbb{Z}$ returns the first prime number following x and n is the overall number of particles in the simulation. For a cloth simulation where there are multiple cloths with multiple thickness, the cell size is set as

$$l = max(cloth_{thickness}), \forall \text{ cloth objects in the simulation} \tag{4.5.7}$$

Inserting a particle into the hash is done by finding the index to place the particle and inserting it at that location:

$$hashTable[hash(\hat{r}(r_i))] = Particle_i \tag{4.5.8}$$

4.5.2.2 Finding the neighbours of a particle Finding the neighbours of a particle is the most important feature of this optimisation structure. Since the data that was saved linked to the hash using an unique index, the process of finding neighbours is simply achieved by querying the table for all of the particles inserted on a given index, having a theoretical complexity of $O(1)$.

However this method does not return the all of the correct neighbours of a particle. To find the missing ones surrounding positions to the particle are queried and verifying if the particles are neighbours or not.

A bounding box is created around the particle between BB_{min} and BB_{max} where

$$BB_{min} = \hat{r}(r_Q - (h,h,h)^T) \ , \ BB_{max} = hatr(r_Q + (h,h,h)^T) \tag{4.5.9}$$

and it will be iterated finding new surrounding positions pos_D finding the neighbours of this position. The result is an exceeding number of overall neighbours to the original particle but now all of the exact neighbours are included.

$$L = hashTable[hash(pos_D)] \tag{4.5.10}$$

Depending on the type of structures kept in the hash the exceeding neighbours can be removed by testing the distance between the particle and the neighbour and checking if it is smaller than the thickness.

This method is perfect for a particle based simulation however for this cloth structure that doesn't work particularly well since triangles are going to be tested instead of the particle's position so all of these new neighbours have to be accepted as a potential list of neighbours and let the collision detection algorithm determine if a collision will occur or not.

5 Implementation

5.1 Pipeline

One of the main goals of this implementation is to create a tool that can be easily integrated in a production pipeline. The user should be able to use it without any extra effort. The following sections describe how the pipeline is created.

5.1.1 Importing Meshes into the Tool

Importing any mesh into the tool was a requirement set at the beginning of the project, being one of the biggest flaws of the previous simulator. The file format chosen to transport meshes between 3D packages and between them and the tool is the OBJ file format.

This format is widely used in the industry since it is package independent containing text based information about vertices and faces. Its ASCII nature makes it a good format to use since it is very easy to parse.

The following table describes the keywords from the OBJ file format used by the application when importing the mesh:

Keyword	Parameters	Description
#	-	Comments
v	x y z	Vertex in position (x,y,z)
vn	x y z	Vertex normal with direction (x,y,z)
vt	u v (w)	Vertex UV coordinate (u,v). Some packages, such as SideFX Houdini, export a third coordinate.
vf	v1 v2 v3 (v4,...)	A face composed by three or more vertices. The parameters are the indexes of the vertices given their order on the vertex list.

5.1.2 Importing meshes into the application

At the beginning of the project a decision had to be made on how to handle the data from the OBJ file. Two possible choices were available: using the NCCA Graphics Library to read the file and then managing a way to export that data from the its structures converting it to cloth, or to convert the object directly into a cloth model object while parsing it. Since this second approach made the simulation faster, an OBJ file parser was created, based on the NCCA Graphics Library, that converted the cloth object directly while parsing the file. Details on how the data was transformed into a cloth object can be found in section 5.3.1.

5.1.3 Reading static objects and sequences

To increase the level of realism and the flexibility of the tool static objects are introduced into the application. The models accepted will go through the same translation as the cloth objects being the only difference the final structures they create.

Creating similar structures allows for a much smoother integration in the application mainly in the collision detection and response algorithms.

An extra important feature of the application is that not only single frame OBJ files are accepted but OBJ file sequences are accepted as well. Initially the intended file format to read animation was the native animation Houdini file format .clip, however the version available of this package no longer exported this format so considering the time left in development at the time the choice to use OBJ sequences was natural. The main disadvantage of using this format is that it holds no information whatsoever between the movement of vertices from one frame to another.

In most simulations this presents problems however, if the movement from one frame to the next is drastic, the cloth might not find the object to perform collisions. More of this subject will be addressed in a later section.

5.1.4 Restrictions applied to the meshes

The OBJ file format is very versatile allowing NURBS surfaces as well as Polygonal ones. Since the purpose of this tool was to allow the user to import virtually any mesh instead of any types of meshes, only triangulated polygonal surfaces are accepted.

The reason why the meshes need to be triangulated will be clear on a later section, but the user will have little problems converting any n-faced mesh into triangulated ones since all major 3D packages have a triangulate mesh function. This triangulation restriction applies to both cloth objects as well as static objects. Static objects have no further restrictions however there are two more applied to the objects that will be converted into cloth.

Since most of object specific parameters such as the texture, stiffness, thickness, amongst others are passed via texture images the objects that will be converted into cloth must have the UVs layed out correctly. This is a common practice in production since the meshes are to be textured and rendered so adding this restriction is a sensible one giving the user no extra work.

The final restriction is that the object should have normals. Failure to comply with this restriction will not affect the simulation but it won't allow the user to visualize the simulation since there are no normals to shade the mesh.

Most of meshes in production have all of these restrictions addressed so any user should be able to run this tool without any extra work.

5.1.5 Exporting simulated cloth into external applications

To finalize the pipeline, the cloth objects must be transported back into the 3D package so, the same format used to import meshes into the application was used. To output the mesh the files are written each frame into a location specified by the user.

5.1.6 Texture Based Parameters

A big objective when developing this tool was to make it extremely user friendly, allowing artists and technical users to use it without any problems. When using a cloth simulation tool there are a number of parameters that can be tweaked such as constraints, thickness of the cloth, stiffness and damping of the springs,

amongst others. Assigning specific values to the entire cloth is a fairly easy way of doing it however, most users want to create small changes within the mesh and the best method to do it is via texture maps.

In production, any mesh that will be rendered needs to have the UVs correctly layed out and that requirement was used in this tool to pass parameters into the simulation. The user will define minimum and maximum values for a parameter and provide with a grey-scale map. The tool will interpolate those values based on the information of the texture setting different values to different particles.

To do so, the NCCA Graphics Library was used however the Texture class needed to be tweaked to send colour information of the texture given UV coordinates. The algorithm used to set the parameters is described below:

for every particle p **do**
  for every texture tex **do**
    if tex exists **then**
      $offsetTex \leftarrow maxValueTex - minValueTex$
      $p.ParamTex \leftarrow maxValueTex + offsetTex * GetColour(p.u, p.v)$
    end if
  end for
end for

This method allows the users to define almost every parameter as they wish throughout the mesh giving more flexibility to the tool.

5.1.7 XML Parser

Finding a good method to pass parameters into the simulation wasn't enough to make it as user-friendly as it can be since the simulator still needs to parse a great amount of information. Choosing or creating a good file format to pass all the information about the scene to the simulation is also very important and given the nature of the XML file, how it is structured, how it is displayed to the user, how easily it is parsed and how easy it is for the user to change it, it was only natural that this would be the chosen file type.

These scene files are automatically created by the interface via a Python script so the user doesn't even need to see the file, however the grammar used is simple enough so artistic users can easily change it.

Tables 5.1.1, 5.1.2, 5.1.3 and 5.1.1 describe the different parameters that will be set in the XML file.

Static object parameters	
objPath	Location of the obj file

Table 5.1.1: The parameters of the static objects.

By creating this file, the user can run the simulation and tweak the scene files without the interface, something that also can be helpful since many times the user just wants to quickly run and export the simulated meshes.

FILE	
objPath	Location of the obj file
clothPath	Location of the cloth file
constraintsPath	Location of the constraints grey-scale texture map
texturePath	Location of the texture map
savePath	Path where the user desires the simulated files to be saved
saveName	The name of the simulated files to be saved
thicknessPath	Location of the thickness grey-scale texture map
massPath	Location of the mass grey-scale texture map
stiffnessPath	Location of the stiffness grey-scale texture map
dampingPath	Location of the damping grey-scale texture map

Table 5.1.2: The file section of the cloth objects define the locations of all the files related to that object in the simulation.

SETTINGS	
minThickness	Minimum thickness value
maxThickness	Maximum thickness value
minMass	Minimum mass value
maxMass	Maximum mass value
minStiffness	Minimum stiffness value
maxStiffness	Maximum stiffness value
minDamping	Minimum damping value
maxDamping	Maximum damping value
restitution	The coefficient of restitution of the collisions of this object
friction	The friction of the object

Table 5.1.3: The settings section of the cloth objects define the parameters of that cloth object in the simulation.

General simulation settings	
time-step	The time-step to use
gravity	The gravity force to be used
externalForces	The external forces
randomness	If the random forces are to be used
randomNegX	Limit random in the x axis only to negative forces
randomPosX	Limit random in the x axis only to positive forces
randomNegY	Limit random in the y axis only to negative forces
randomPosY	Limit random in the y axis only to positive forces
randomNegZ	Limit random in the z axis only to negative forces
randomPosZ	Limit random in the z axis only to positive forces
integration	The integration method: euler or runge kutta
StartFrame	The start frame of the simulation
EndFrame	The end frame of the simulation
Collisions	The number of extra iterations done to find more neighbours

Table 5.1.4: The general settings of the simulation that can be set.

5.2 Scene

The scene is created directly via the XMLParser class. At the beginning of the simulation the scene file is parsed creating and correctly placing cloth and static objects in their correspondent arrays, parsing and creating the necessary structures, saving the simulation parameters and it's only after the entire XML file is parsed and the variables initialised that the simulation starts. Figure 5.2.1 consists on the class diagram used for the Scene.

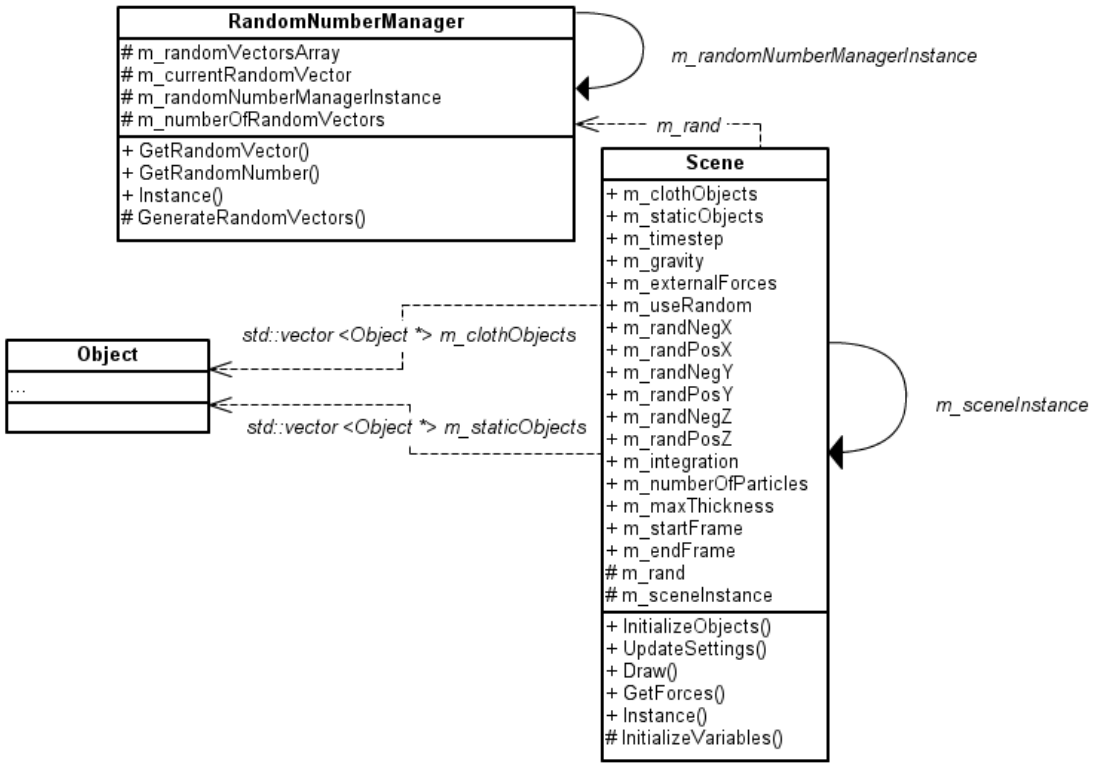


Figure 5.2.1: Class diagram for the scene.

5.3 Cloth Model

5.3.1 Loading the OBJ

The cloth model is represented by particles, springs and faces. The similarities between this model and the OBJ file format are easily identified therefore the translation between the parsed data and the cloth is made in an intuitive way. As mentioned the OBJ file format describes a list of vertices and which ones constitute each face.

The vertices are converted into particles and the face information is parsed in the following way: Let f be a face, v_i the vertices that constitute f and e_j the edge springs created:

$e_1 \leftarrow CreateEdgeSpring(v_1, v_2)$
$e_2 \leftarrow CreateEdgeSpring(v_1, v_3)$
$e_3 \leftarrow CreateEdgeSpring(v_2, v_3)$

Creating the bend springs is a method that has to be made after the file is parsed since no information is given in the OBJ file regarding the order or connection of the triangles. Every triangle has to be tested against each other, checking if they are adjacent and if they are create a bend spring between the vertices that are not connected already:

for every face f_i **do**
  for every vertex v_i in f_i **do**
    for every other face f_j **do**
      for every vertex v_j in f_j **do**
        if f_i and f_j share two edges **then**
          $v_i \leftarrow IsolatedVertex(f_i)$
          $v_j \leftarrow IsolatedVertex(f_j)$
          $CreateBendSpring(v_i, v_j)$
        end if
      end for
    end for
  end for
end for

This is a very time-consuming process therefore a method to improve it had to be created. The solution found was that the first time a cloth object is parsed all the triangles are tested to find the bend springs however a file is written with all the information of the springs so that the second time that object is loaded the springs are already pre-calculated. The following flowchart represents how the object is parsed:

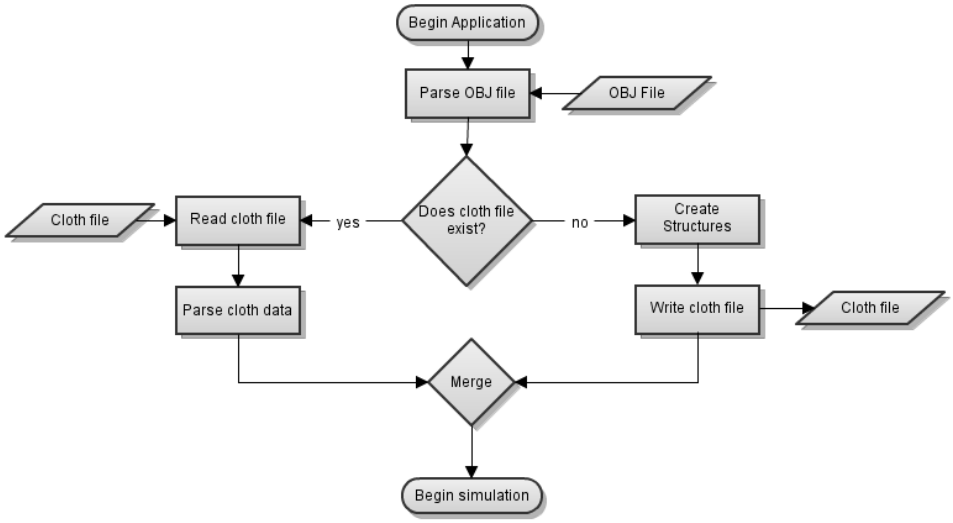


Figure 5.3.1: Flowchart representing how an object is parsed.

This is a very simple but effective approach but the results are significant

since a 100000 polygon object takes seven minutes to parse the first time it is loaded but only 0.25 seconds the subsequent times.

5.3.2 Cloth and Static Object Structures

The objects loaded in the simulation, both cloth and static ones, share the same class however the amount of data saved is different per object as well as the operations performed on each of them.

These structures exist mainly to save information and they will be extensively used throughout the application allowing access to the particles and faces in the cloth as well as the drawing and the saving functions that are called to display and export the cloth.

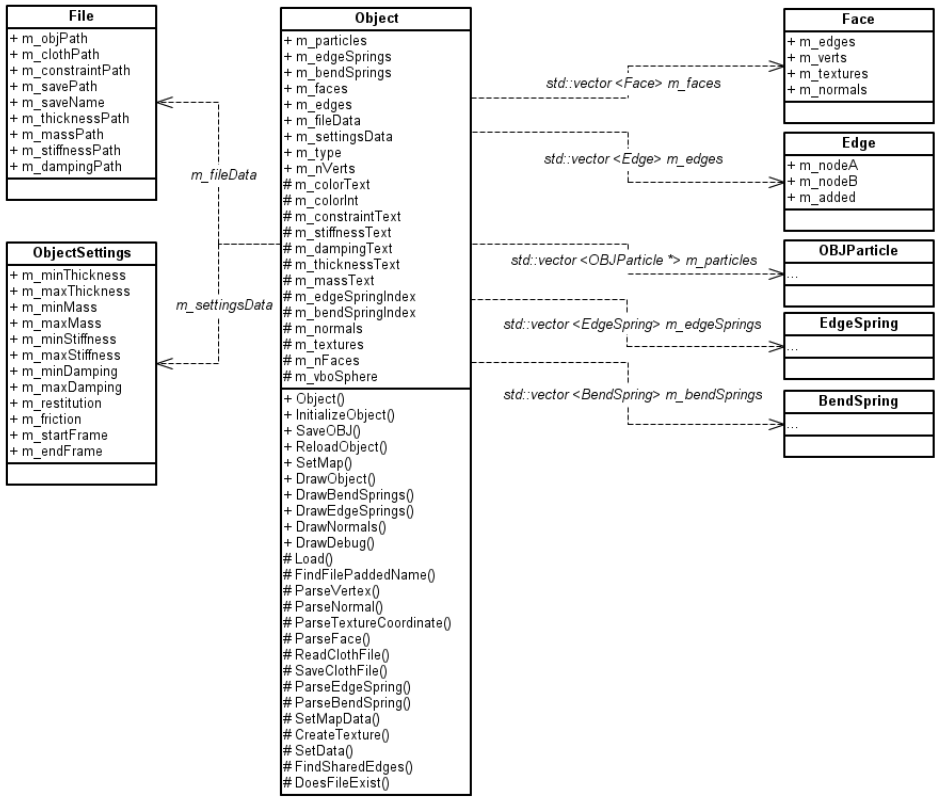


Figure 5.3.2: Class diagram for the Object structure.

5.3.2.1 Particles The particles of the cloth are the main object in the application since they are part of the other two structures that constitute it, faces and springs. The previous implementation developed for CGI Techniques had a good way of handling the particles, so a very similar structure was used. A base class named MovingObject is created and the OBJParticle will extend it.

The advantages of using such method is that the base class contains generic parameters such as position and velocity, and the particle class will only declare variables that are exclusive to it such as damping and stiffness of that particle. This way any object can be created, extending this base class and methods such as the integration that only deal with MovingObjects don't need to be changed when more objects are introduced to the simulation.

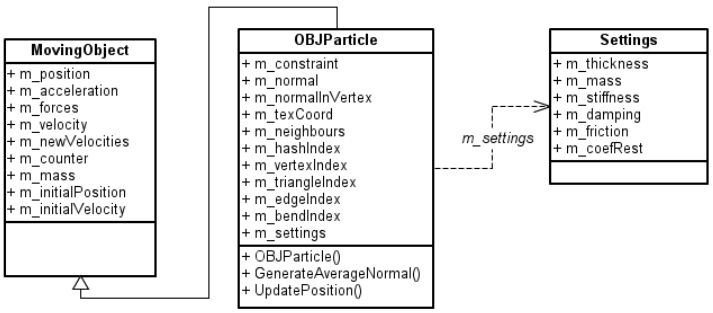


Figure 5.3.3: Class diagram of the OBJParticle.

5.3.2.2 Springs The principle behind creating the springs was the same as the particle: creating a base class and all of the specific classes would extend from it. So a Spring class was created with generic functions and both EdgeSpring and BendSpring extend from it. Even in this case this method is useful since there are several methods to implement how the bendsprings behave so the developer only need to change the code of the update function of the BendSpring class without ruining the EdgeSpring. Figure 5.3.4 show the class diagram for this object.

5.4 Integration

As described, the user can choose one of two methods to update the position of the particle: Euler integration or Runge-Kutta 4th Order. The implementation behind these methods is divided into two categories: an isolated class called Integration and the function calls in the UpdateCloth function.

The integration class is an abstract one that only deals with the generic MovingObjects allowing the user to update any type of object as long as it extends its base class. The methods were created using the theory in section 4.2.

The UpdateCloth function will call either euler integration or runge kutta depending on the user's choice and follows this algorithm:

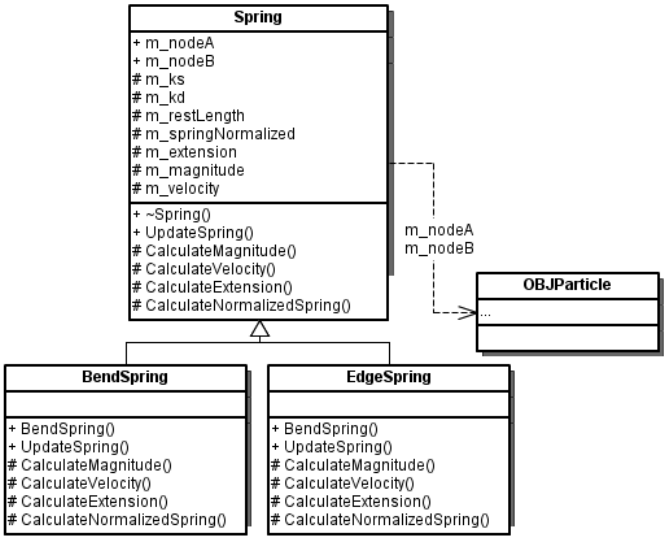


Figure 5.3.4: Class diagram for the Spring.

if Euler Integration **then**
  for every cloth object *cloth* **do**
    for every particle p **do**
      $p \leftarrow EulerIntegration(p)$
    end for
  end for
else
  for every cloth object *cloth* **do**
    $p \leftarrow RungeKuttaIntegration(cloth)$
  end for
end if

where $RungeKuttaIntegration(cloth)$ is defined by:

$h \leftarrow time - step$
$UpdateForces(cloth)$
$k1 \leftarrow RungeKuttaStep(cloth, 0)$
$UpdateForces(cloth)$
$k2 \leftarrow RungeKuttaStep(cloth, 0.5 * h)$
$UpdateForces(cloth)$
$k3 \leftarrow RungeKuttaStep(cloth, 0.5 * h)$
$UpdateForces(cloth)$
$k4 \leftarrow RungeKuttaStep(cloth, 0.5 * h)$
for every particle p **do**
  $RungeKuttaIntegration(cloth, k1, k2, k3, k4, h)$
end for

To implement this integration method, initially (Fiedler, 2006) was consulted. It provides all the necessary code to develop the RK4 method however, the sample code it provides is based on a single variable without any connection

with other objects and no dependencies. For this cloth simulation, as it will be described later on, each particle is connected in multiple springs that will be updated and influence the acceleration of such particle.

5.5 Collisions

The collision detection and response was the biggest task in the development of this application taking most its time. To produce a realistic cloth simulation not only the movement must look realistic but collisions between cloth and external objects must exist as well as collisions within the cloth.

The collisions detection phase was straightforward once the math was understood but what took most of the time was collision response because if it doesn't look realistic the user will immediately notice it.

The methods used to create these collisions will be described below.

5.5.1 Collision Detection

Once the theory behind collision detection, as it is described in the Technical Background chapter, is understood, the development is quite straightforward.

All that is needed is to determine if a particle collides with any of the triangles where its neighbour particles exist. The algorithm to do so is the following:

```
for every update to the cloth do
  UPDATE THE CLOTH
  UPDATE THE HASH
  for every cloth object in the scene do
    for every particle p in that cloth object do
      if p.constraint == false then
        for every neighbour neighbour of the particle do
          if neighbour.type == OBJECT then
            for every triangle t where that particle exists do
              PointTriangleCollision(p, t)
            end for
          else
            for every triangle t where that particle exists do
              if p is not in t then
                thick ← CalculateAverageThickness(p, t)
                PointTriangleCollision(thick, p, t.p1, t.p2, t.p3)
              end if
            end for
          end if
        end for
      end if
    end for
  end for
  CALCULATE NEW VELOCITIES
end for
```

As mentioned, the point-triangle test was used to detect collisions as proposed by (Bridson et al., 2002), however the author also states that edge-edge collisions had to be verified. Once implemented no difference was noted with edge-edge collisions since all of them were caught by the first method. (Cheng,

2009) implemented both (Bridson et al., 2002) and (Bridson et al., 2005) and also faced the same issue so no edge-edge collision detection was implemented.

Analysing this situation, it was decided that this test would be removed since it slowed the application and its results were not visible.

5.5.2 Collision Response

As mentioned before, the collisions were the most time-consuming process in the development of this tool. Once the math was behind the collision detection was understood the implementation was direct.

Not mentioned in the referenced paper is that multiple collisions in the cloth will occur so the particles will have forces applied to them several times overriding the previous ones. The method found to overcome this difficulty is that the velocities must be averaged creating this way good results without allowing self collisions.

The only exception is when the collision occurs with a static object. In this case all the other collisions are overridden so the cloth is moved to a safe location.

5.5.3 Spatial Hash

As mentioned, collision detection is a high time-consuming process. Unless there is an optimisation system the every particle in the cloth will test for a collision with every triangle in every object. To find the closest neighbours to a particle is imperative to limit the number of tests performed and as a consequence making the simulation faster. In this simulation the method proposed by (Kelager, 2006) was used.

The spatial hashing optimisation was created using a C++ structure called a multi-map. This structure, similar to a hash, allows the user to specify the two parameters, the first is the location where the data is going to be saved and the second is the data itself.

The method with which a particle is placed into the multi-map and how the neighbours are found were previously described, however the structure used to keep particle related data was not. After careful consideration it was noted that not only the position of the particle was necessary to keep in the hash but in what object it was inserted and where that object existed in the main object vector was also important, so a structure called SpatialVertexInfo was created keeping all the relevant data of a particle.

On every update the hash will be repopulated and the neighbours recalculated. Below is the algorithm used on updating the cloth objects:

for every update to the cloth **do**
  UPDATE THE CLOTH
  $hash.CleanHash()$
  for every object in the scene **do**
    for every particle p in the object **do**
      $hash.AddParticle()$
    end for

end for
for every cloth object in the scene **do**
for every particle p in the object **do**
$hash.FindNeighbours()$
end for
end for
DO COLLISION HANDLING
CALCULATE NEW VELOCITIES
end for

Apart from its easy implementation, this method worked very well and it is possible to specify how exhaustive we want the neighbour lookup to be, allowing users to run quick simulations, missing some of the collisions, or by increasing this neighbour lookup refinement variable, run slower simulations but with an almost perfect collision detection.

A few modifications had to be made to the original algorithm to allow for the structures used in this application but most of the method remains the same.

The class diagram for this optimisation structure is described below:

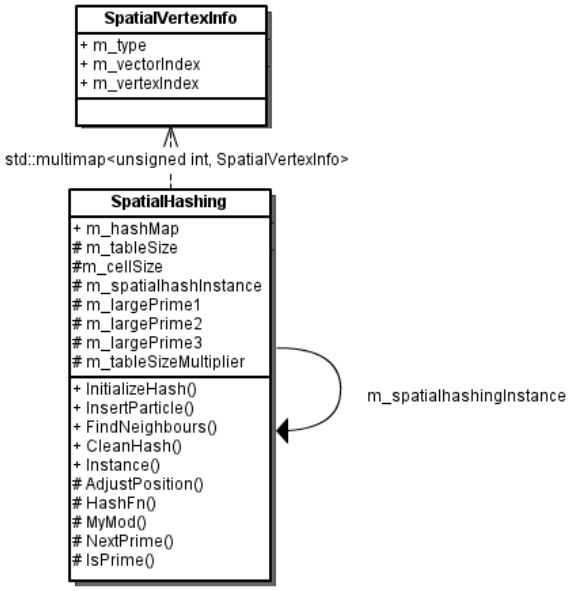


Figure 5.5.1: Diagram for the Spatial Hash class.

5.6 Houdini Interface

To create a way for an inexperienced user be able to use the application an interface was created in Houdini. This interface can be a bit complex at a first look because it has a lot of variables but after it is used for the first time it is quite intuitive and straightforward.

The interface automatically exports the OBJs for the user, and all it needs to be specified are the locations where the files are supposed to be save. To avoid rewriting the files every time the tool in ran there is an "Overwrite" toggle

button allowing the user to specify if the files are going to be exported again or not.

Apart from defining file locations, the other settings, when it comes to the cloth objects, are meant to define values for parameters such as stiffness, damping, amongst others. The user has two choices: disable the "Use Texture" toggle button and only define a value that will be spread throughout the entire mesh, or enable that button, set a minimum and a maximum and use a texture file to interpolate the values.

When it comes to the overall setting of the simulation there are two sections: Forces and General Settings.

Forces will define what are the values for the gravity and for other external forces. The external forces can be either constant or randomised allowing the user, in case he chooses this last option, to clamp them to positive or negative on all the axis.

General Settings will define the integration method, Euler or Runge-Kutta 4th Order, and the time-step used for the simulation.

There is an abstract layer to the user where, when the "Run Simulation" button is clicked, the application is executed. At this time a network is created in Houdini that holds the newly simulated cloth. A Python script was created to handle all of the data creating a XML file, as mentioned, running the simulation and creating the network.

By using this interface any user, technical or artistic, is able to use the application and given how user-friendly this method is quickly test different settings and analyse different results.

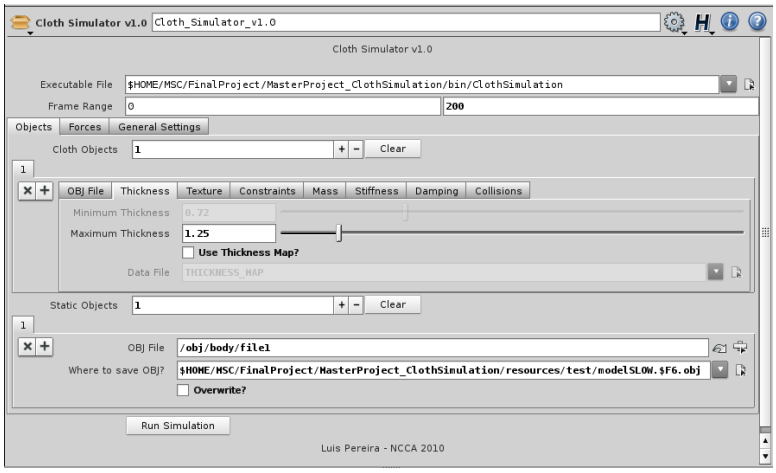


Figure 5.6.1: Interface created for Houdini.

6 Results

There are several results that were achieved with this tool from simple material tests to complete integrations into real scenes.

6.1 Collision Handling

Figure 6.1.1: Collision detection and response of the cloth when interacting with a animated static object.

One of the objectives of this tool was to allow for the cloth to be simulated into a real character. Initially the desired animation format to use would be the native Houdini animation .clip file however the current version of Houdini only allows the user to export binary versions of that file making the ASCII parser obsolete.

To overcome this difficulty the tool accepts sequences of OBJs and it allows the cloth to interact with them. The image above shows the results achieved by roughly modelling a dress in Maya and running the simulation. Even though some problems arose with the simulation such as the dress falling down or sometimes, when the arm of the model intersects the torso, the cloth behaving

strangely since it does not know where to place itself in order to avoid the collisions.

The model used is from (Kieran et al., 2005) as well as the FBX animation.

6.2 Modelling Tool

Figure 6.2.1: Initial model of the dress on the left and the simulated one on the right.

One possible use of this simulation is to model garment onto characters. Modelling high detailed wrinkles and folds that naturally hang from a model is a very time-consuming process and even then the final outcome might not look as realistic as possible.

This example shows that by modelling a very rough dress on the character, and running the simulation, the final outcome is a highly detailed dress with folds and wrinkles that are adjusted to the model. This mesh of this resultant dress can still be sculpted by the artist providing it with even more detail. This technique is recommended for static images where a quick first version of clothing is desired.

The model used is taken from (Kieran et al., 2005).

6.3 Different Materials

Another test made with the application is to achieve different types of materials. By simply changing the stiffness and damping of the springs, as well as the thickness of the cloth the user can create a range of materials that go from a very latex/rubber kind of cloth, to a proper satin cloth, very thin and light.

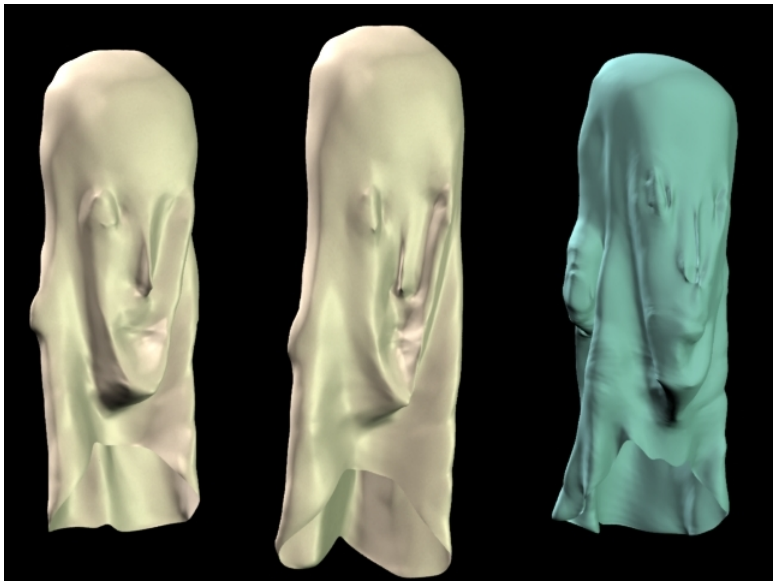


Figure 6.3.1: Three different materials created with the tool. From left to right the rubber feeling of the cloth diminishes.

6.4 Backplate Integration

Finally some tests were made to the possibility of integration of the tool into a production pipeline attempting to create a complete scene.

Four static images were downloaded from the internet, two to serve as background and two to replace the sky. The camera was roughly matched, dummy geometry was created to allow interaction with the cloth and the simulation was executed.

After taking the files back to the 3D package, every item was rendered and composited in Nuke to produce the two images below.

Static images were taken from (Vargas, n.d.), (Farm, n.d.), (Derrich, n.d.) and (Xinhua, n.d.).

Figure 6.4.1: Integration of a cloth object into a backplate.

Figure 6.4.2: Another integration of the cloth into a backplate. The animated object collides with the cloth.

6.5 Multiple Objects

The possibility of simulating multiple objects is very important in a cloth simulator. This application handles multiple objects in the following two ways:

6.5.1 Simulating several cloth objects

The first method to simulate several cloth objects is to create separate OBJs and simulate them. The tool will create two objects and simulate them independently. In the figure 6.5.1 the two cloth objects are the jacket and the trunks. They were created as two separate objects, each with their own texture, constraints and parameters. The troll model used was created by Ritchie Moore.

6.5.2 Simulating several objects with one OBJ

The second method takes full advantage of the flexibility of the application. In this example all of the flags were written in a single OBJ file, having all of the parameters being written into a single file texture per parameter. The

Figure 6.5.1: Simulating several cloth objects.

simulator will interpret this as a single cloth object but single every single flag has their own stiffness, damping, amongst other parameters, the final simulation will produce different animations for each. Figure 6.5.2 show such an example as 24 flags were simulated. The backplate was taken from (Thundafunda, n.d.).

Figure 6.5.2: Simulating several cloth objects using a single OBJ.

7 Further Work

To further improve the application some new features can be implemented allowing it to produce larger range of results and to be better integrated into a production pipeline:

- **Implicit integration methods** Creating implicit integration methods allows for larger time-steps of the simulation and even though the calculation per iteration is slower, less iterations are required to write a frame. Critics made to this method were that the cloth would become very stiff, however techniques mixing explicit and implicit integration methods as proposed by (Bridson et al., 2005) claim to solve those stiffness issues.
- **Animation file format** A very important new feature to be implemented is the inclusion of ASCII animation data import. The method should be as such to allow rigging data so the cloth could be attached to certain parts of the mesh.
- **Folds and wrinkles upon static object impact** (Bridson et al., 2005) proposes a model to maintain folds and wrinkles when cloth hit static objects. This would be an interesting optimisations since currently the position of the particles is changed to a safe position ignoring folds and wrinkles.
- **Tearing** Allowing the cloth to tear was one of the initial objectives of this application however due to time restrictions it was not implemented. Cloth is allowed to tear if the springs are stretched more than a predetermined value. If it occurs a new particle is created and springs reconnected in order to split the cloth. To implement this feature the development has to be directed to allow the insertion and deletion of particles at any given time otherwise the amount of code that would have to be changed can be very high.
- **More Collision Optimization Methods** Apart from the spatial hash implemented, other methods could be implemented such as kd-trees or AABB trees. This would give the user the possibility to choose the method he wants to use.
- **Adaptive meshing** This technique is presented by (Villard and Borouchaki, 2005) and the idea behind it is very interesting and useful. Whenever the cloth is bending more than a predetermined value, each face will be subdivided allowing the creation of more mesh wherever it is needed. The number of subdivisions can be set by the user allowing the resolution of the cloth to become very high. According to the author, the results using this method and a very dense mesh are the same however the time needed to simulate this adaptive mesh is half of the dense one.
- **GPU calculations** The computational power of the GPU is much higher than of the CPU, even with multiple cores. Cloth simulations are a perfect candidate to split up calculations since the particles are isolated. Even though the learning curve for GPU programming can be quite steep the final results are very good decreasing the calculation times significantly.

- **Parallelizable calculations** Another method to optimise the calculation time of the simulation is to split the computation using several computers. Aside from using different computers, several cores of each computer can be used. This approach could be better than using the GPU to insert the tool into a proper production pipeline since a render-farm already exists and the costs of using the GPU (air-conditioning and electricity) would be very high.

8 Conclusion

Approximately three months were spent working, the initial two to three weeks consisting of research, and the last two writing this thesis, producing more videos and tweaking the code where it was necessary, leaving around six to seven weeks to implement the tool.

Considering the goals set at the beginning of the project the following is an analysis on its success:

- **Complete integratable tool** As shown in the Results chapter two scenes were successfully integrated allowing the user to completely go through its pipeline inserting this tool before rendering the final images.
- **User defined cloth mesh** Any mesh is accepted in the tool as long as some restrictions are met, as previously described. These restrictions are common practice in a production pipeline so the user should have no extra work in meeting them.
- **Cloth Self Collisions** Lacking from the previous development, self collisions were a great addition to this tool. They gave the cloth a high degree of realism.

Another focus of the implementation was to produce a tool that was flexible enough to allow any new developer to continue where this project ends without being forced to change any significant amount of code, and that was refined enough so that the features that are implemented are done in a robust and optimised way.

In general, this project was approached as a new learning experience, using the knowledge gained during the CGI Techniques course and producing a tool that is much more refined and flexible when compared to the previous one. The pitfalls of the previous tool were avoided but naturally new ones surfaced, such as producing visually appealing collision responses, taking much more time than initially expected and therefore increasing the implementation time.

The Results chapter, and videos that are handed along with this thesis, reveal that this tool was successfully integrated into a backplate, creating integrated scenes, as well as producing a range of different outcomes such as different types of materials. By implementing some or all of the features included in the Further Work chapter this tool could become a very good cloth tool to have in a production pipeline.

The overall feeling, as the project reaches its end, is of success. Even though there are many topics not approached during this project, mainly due to the high degree of research made on this computer graphics field, the ones that were produced realistic results without removing any control over the simulation the final outcome from the artist.

References

Baraff, D. and Witkin, A. (1998). Large steps in cloth simulation, *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, pp. 43–54.
URL: *http://doi.acm.org/10.1145/280814.280821*

Breen, D. E., House, D. H. and Wozny, M. J. (1994). Predicting the drape of woven cloth using interacting particles, *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, pp. 365–372.
URL: *http://doi.acm.org/10.1145/192161.192259*

Bridson, R., Fedkiw, R. and Anderson, J. (2002). Robust treatment of collisions, contact and friction for cloth animation, *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, pp. 594–603.
URL: *http://doi.acm.org/10.1145/566570.566623*

Bridson, R., Marino, S. and Fedkiw, R. (2005). Simulation of clothing with folds and wrinkles, *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, ACM, New York, NY, USA, p. 3.
URL: *http://doi.acm.org/10.1145/1198555.1198573*

Carignan, M., Yang, Y., Magnenat-Thalmann, N. and Thalmann, D. (1992). Dressing animated synthetic actors with complex deformable clothes, *Computer Graphics (Proceedings of ACM SIGGRAPH 92)*, ACM Press, pp. 99–104.

Cheng, H.-H. (2009). *Interactive cloth simulation*, Master's thesis, Computer Science Department, University of California, Los Angeles, USA.
URL: *http://www.scribd.com/doc/24601463/Interactive-Cloth-Simulation-Stella-Cheng*

Derrich (n.d.). Derrich.com, `http://www.derrich.com/tag/san-antonio/`. Last accessed, 30 July 2010.

Desbrun, M., Schröder, P. and Barr, A. (1999). Interactive animation of structured deformable objects, *Proceedings of Graphics Interface (GI 1999)*, Canadian Computer-Human Communications Society, pp. 1–8.
URL: *http://www-grail.usc.edu/pubs/DSB_GI99.pdf*

Ericson, C. (2005). *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology)*, Morgan Kaufmann.
URL: *http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/1558607323*

Farm, B. B. (n.d.). Bamboo banks guest house and farm, `http://bamboobanks.in/fac/AT/`. Last accessed, 30 July 2010.

Fiedler, G. (2006). Gaffer on games: Integration basics, `http://gafferongames.com/game-physics/integration-basics/`. Last accessed, April and May 2010.

House, D. H. and Breen, D. E. (eds) (2000). *Cloth modeling and animation*, A. K. Peters, Ltd., Natick, MA, USA.

Kelager, M. (2006). *Lagrangian fluid dynamics using smoothed particle hydrodynamics*, Master's thesis, Department of Computer Science, University of Copenhagen.
URL: *http://image.diku.dk/projects/graphics.php*

Kieran, E., Harrison, G. and Openshaw, L. (2005). *Cloth simulation*, Master's thesis, Bournemouth University, Poole, UK.
URL: *http://nccastaff.bournemouth.ac.uk/jmacey/MastersProjects/Msc05/cloth_simulation.pdf*

Liberatore, M. (n.d.). Comparing numerical integration methods in a simulator for the draping behavior of cloth, `http://src.acm.org/liberatore/liberatore.html`. Last accessed August 2010.

Macri, D. (2010). Simulating cloth for 3d games. Last accessed, April 2010.
URL: *http://software.intel.com/en-us/articles/simulating-cloth-for-3d-games/*

Müller, M., Heidelberger, B., Hennix, M. and Ratcliff, J. (2007). Position based dynamics, *J. Vis. Comun. Image Represent.* **18**(2): 109–118.
URL: *http://dx.doi.org/10.1016/j.jvcir.2007.01.005*

Ng, H. N. and Grimsdale, R. L. (1996). Computer graphics techniques for modeling cloth, *IEEE Comput. Graph. Appl.* **16**(5): 28–41.
URL: *http://dx.doi.org/10.1109/38.536273*

NVIDIA (2007). Cloth simulation, *Technical report*, NVIDIA.

Provot, X. (1995). Deformation constraints in a mass-spring model to describe rigid cloth behavior, *Graphics Interface '95*, pp. 147–154.

Provot, X. (1997). Collision and self-collision handling in cloth model dedicated to design garments, *Proceedings of the Eurographics Workshop on Computer Animation and Simulation (CAS 1997)*, Springer-Verlag, pp. 177–189.
URL: *http://www-rocq.inria.fr/mirages/SYNTIM_OLD/textes/Collisions_vetements.ps.gz*

Selle, A., Su, J., Irving, G. and Fedkiw, R. (2009). Robust high-resolution cloth using parallelism, history-based collisions, and accurate friction, *IEEE Transactions on Visualization and Computer Graphics* **15**(2): 339–350.
URL: *http://dx.doi.org/10.1109/TVCG.2008.79*

Terzopoulos, D., Platt, J., Barr, A. and Fleischer, K. (1987). Elastically deformable models, *Computer Graphics (Proceedings of ACM SIGGRAPH 87)*, ACM Press, pp. 205–214.

Thundafunda (n.d.). Mountain pictures, `http://thundafunda.com/5/Mountains/`. Last accessed August 2010.

Treuille, A. (n.d.). Simulation of natural phenomena. Last accessed, July 2010.
URL: *http://graphics.cs.cmu.edu/courses/15-869/*

Vargas, J. (n.d.). http://www.jlvargas.com/italy.html. Last accessed, 28 July 2010.

VDocs (2009). Numerical integration, http://wiki.vdrift.net/Numerical_Integration. Last accessed, 20 May 2010.

Villard, J. and Borouchaki, H. (2005). Adaptive meshing for cloth animation, *Eng. with Comput.* **20**(4): 333–341.
URL: *http://dx.doi.org/10.1007/s00366-005-0302-1*

Volino, P. and Magnenat-Thalmann, N. (2000). Implementing fast cloth simulation with collision response, *Proceedings of Computer Graphics International (CGI 2000)*, IEEE Computer Society, pp. 257–268.
URL: *http://www.miralab.unige.ch/papers/47.pdf*

Volino, P. and Magnenat-thalmann, N. (2001). Comparing efficiency of integration methods for cloth simulation, *Computer Graphics International Proceedings*, IEEE Computer Society, pp. 265–274.

Xinhua (n.d.). Window of china, http://rss.xinhuanet.com/newsc/english/2008-08/23/content_9666071.htm. Last accessed, 28 July 2010.

A Math

A.1 Cramer's Rule

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} s \\ t \end{bmatrix} = \begin{bmatrix} E \\ F \end{bmatrix} \tag{A.1.1}$$

$$s = \frac{E * D - B * F}{A * D - B * C} \tag{A.1.2}$$

$$t = \frac{A * F - E * C}{A * D - B * C} \tag{A.1.3}$$

A.2 Solving Point Triangle Intersection Test

$\vec{P_c}$ - Collision Point
u, v, w - Barycentric Coordinates

$$u + v + w = 1 \tag{A.2.1}$$
$$0 \leq u, v, w \leq 1 \tag{A.2.2}$$
$$\vec{P_c} = u\vec{x_1} + v\vec{x_2} + w\vec{x_3} \tag{A.2.3}$$
$$\vec{P_c} = \vec{x_1} + v(\vec{x_2} - \vec{x_1}) + w(\vec{x_3} - \vec{x_1}) \tag{A.2.4}$$
$$\vec{P_c} - \vec{x_1} = v(\vec{x_2} - \vec{x_1}) + w(\vec{x_3} - \vec{x_1}) \tag{A.2.5}$$

$$V_0 = \vec{x_2} - \vec{x_1} \tag{A.2.6}$$
$$V_1 = \vec{x_3} - \vec{x_1} \tag{A.2.7}$$
$$V_2 = \vec{P_c} - \vec{x_1} \tag{A.2.8}$$

From A.2.5:

$$vV_0 + wV_1 = V_2 \tag{A.2.9}$$

$$\begin{cases} (vV_0 + wV_1) \cdot V_0 = V2 \cdot V_0 \\ (vV_0 + wV_1) \cdot V_1 = V2 \cdot V_1 \end{cases} \tag{A.2.10}$$

$$\begin{cases} v(V_0 \cdot V_0) + w(V_1 \cdot V_0) = V2 \cdot V_0 \\ v(V_0 \cdot V_1) + w(V_1 \cdot V_1) = V2 \cdot V_0 \end{cases} \tag{A.2.11}$$

$$\begin{bmatrix} V_0 \cdot V_0 & V_1 \cdot V_0 \\ V_0 \cdot V_1 & V_1 \cdot V_1 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} V_2 \cdot V_0 \\ V_2 \cdot V_1 \end{bmatrix} \tag{A.2.12}$$