Cloth Simulation Report

Octavian Mihai Vasilovici

9 May 2013

Bournemouth University

1. Summary

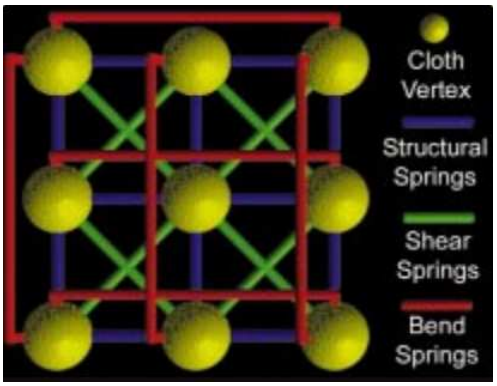
This document aims at explaining the methods decided to be used in creating a Cloth simulation program with the help of the NGL library. It will focus on showing the methods, design, algorithms and decisions that were made. It will consist of research, the theoretical algorithms and methods selected for the implementation, how certain drawbacks are planned to be resolved and the features that are going to be added later.

2. Introduction

The principle of simulating cloth was first introduced in 1930 (Pierce, 1937), but only in 1980's the graphics community took an interest in the subject (House, 1990). From this point onward the subject of simulating cloth has become a popular topic of research. The main scope is to have proper looking cloth that deforms and reacts to the forces applied to it. Cloth simulation it is currently used both in the movie industry and computer games industry (real time simulation). Currently in the game industry cloth simulation is used a lot. Currently there are a lot of physically-based algorithms to produce nice looking result, but their use in the video games industry is low due to the fact that they are fairly complicated and computationally expensive. Precise simulation methods exist and can be borrowed from the engineering, however in video games and interactive use, accuracy is not the main goal, but rather the believability of the simulation (Jakobsen, 2001). The methods described further are meant for interactive use and they do not try to simulate how a piece of cloth would perform in the real world, instead it focuses on the approach selected to simulate a believable piece of cloth.

3. The Spring Mass model

The most common system of simulating cloth is the Spring Mass Model. This model uses the concept of masses that are simulated by using particles, which are connected together in a grid by springs. The typical representation of such a system is composed of: three type of springs (or constraints) namely stretch, shear and bend that connect each mass point to its neighbors. All these constraints give the cloth its behavior.



The interconnection of cloth springs - *Jeff Lander, 1999*

Therefore, the simulation of the cloth comes down to move these masses in a realistic way. The forces that are applied to the cloth fall into two categories: **internal** and **external**.

The internal force appears due to the tensions between the constraints and can be expressed using Hooke's law of elasticity:

$$F= -k*x$$

- ***k*** *is spring constant.*
- ***x*** *is the displacement in the spring from its rest length (the distance between two neighbor particles in rest state).*

Another component of the internal force is the damping coefficient. This coefficient is responsible for "bringing" the system back to the rest state. Without it, the springs would oscillate infinitely.

$$F = -c*v$$

-c *is the damping coefficient.*
-v *is the velocity of the particle.*

The final equation of **internal forces** can then be written as:

$$F = -k*x –c*v$$

The **external forces** are basically composed of two forces: gravity and another external force such as wind.
In order to represent gravity we must apply Newton's second law:

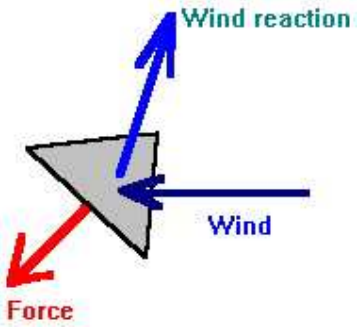
$$G= m*g$$

- ***m*** *is the mass*
- ***g*** *is gravitational acceleration*

Wind is another external force that can be applied to the cloth. Rather than being applied to each particle that is part of the cloth, the wind force is applied to a face created by three neighbor particles:

F= c*(n dot d)*n

- ***c*** *is the wind strength*
- ***d*** *is the wind direction*
- ***n*** *is the normal of the face*



Wind simulation of a mesh triangle - *Hugo Elias*

Adding all these forces will create the motion of the particles in the cloth, which in their turn will start adding tension in the constraints by creating expansions and contractions of the springs.

In order to have a simulation we must make the particle move. In order to calculate where a particle will be located in time we need to use an **integration** method.

As an integration method the so-called "**Verlet**" method was selected. Since a piece of cloth is basically composed of particles and constraints, the typical implementation would be for each particle to have two main variables: position **x** and velocity **v** (Jakobsen, 2001).
The integration method would then state that the new position and velocity would be:

$$x' = x + v \cdot \Delta t$$

$$v' = v + a \cdot \Delta t$$

- Δt *is the timestep*
-*a is the acceleration computed from Newton's law f=m*a (a=f/m)*

(Euler Integration scheme)

In the above representation the velocity vector can be removed altogether to give birth to the "Verlet" integration scheme. By not representing each particle's position and velocity, but instead using for each particle a current position **x** and a previous position **x'**, while keeping a fixed time step, the integration becomes:

$$x' = 2x - x' + a \cdot \Delta t^2$$

$$x^* = x$$

(Verlet Integration scheme)

The main benefit of this method is its stability (Jakobsen, 2001) since the velocity is implicitly given and therefore harder for the velocity and position to get out of sync. It works since:

$$2x\text{-}x' = x + (x\text{-}x')$$

- *x is the current position*
- *x' is the previous position*
- *x-x' is the approximation of the current velocity (distance traveled from last time step)*
- x^* *is the new position*

4. Implementation

The selected implementation was first described by Thomas Jakobsen in the article Advanced Character Physics in 2001. The algorithm was developed by IO Interactive and was first time used in the video game Hitman: Codename 47.

The most important thing about this algorithm is the way springs (constrains) are implemented. Instead of using springs with spring coefficients the algorithm uses stiff constrains that can be seen as really hard springs (Jakobsen, 2001).

The algorithm follows these steps:

- For each vertex a particle is created, with the constraint rest length being the initial distance between two particles
- The constraints are then simply handled by using the **relaxation** method over all the springs.

While most of the time only one iteration over the constrains is necessary to obtain good looking animations (Jakobsen, 2001) in the implanted code each constraint is actually satisfied 3 times to try and prevent some of the problems that could appear in some cases.

The time usage depends mostly on the N square root operations and the N divisions performed, where N is the number of edges in the cloth mesh:

```
Vector3 delta = x2-x1;
float deltalength = sqrt(delta*delta);
float diff=(deltalength-c.restlength)/deltalength;
x1 += delta*0.5*diff;
x2 -= delta*0.5*diff;
```

(Pseudo-code for constraint calculation - Thomas Jakobsen, 2001)

The algorithm can be further optimized by removing the square root operation. Since we already know that the result of the square root operation in a certain constraint should be the rest length of that particular constraint, we can use this fact to approximate the square root function. This is done by using the 1st order Taylor expansion on all the particles that are surrounding the square rest length, r*r (Jakobsen, 2001).

The algorithm than becomes:

```
Vector3 delta = x2-x1;
Delta *= restlength * restlength / (delta* delta + restlength * restlength) -0.5
x1 += delta*0.5*diff;
x2 -= delta*0.5*diff;
```

(Pseudo-code for constraint calculation - Thomas Jakobsen, 2001)

For each frame the operations are now down to N divisions which hugely increase the algorithm efficiency.

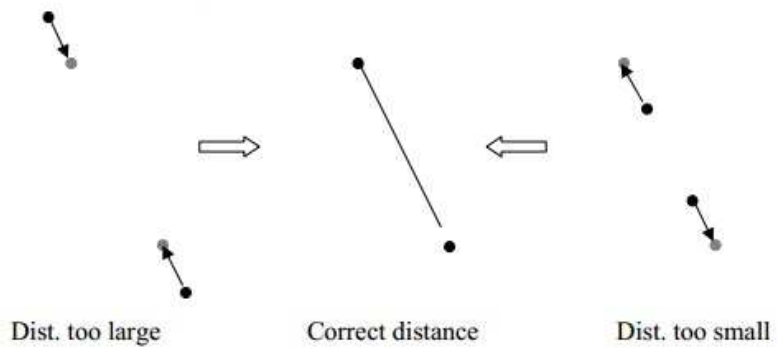
4.1Relaxation

The most common model approach for cloth is based on using a simple system of interconnected constraints and particles. However it is not always trivial to solve the corresponding differential equations: strong springs lead to stiff systems of equations that in their turn can lead to system instability while weak springs lead to rubber looking cloth.

However if the springs are left to be stiff to infinity the system suddenly becomes solvable in a stable, easy and fast way (Jakobsen, 2001). Even if the particles might be placed correctly in the initial phase after some integration steps the distance between them might become invalid. In order to obtain the correct distance again, the particles are moved by using a secondary set of solutions:

$$|x2 - x1| = rest\_length$$

This is done pulling or pushing the particles away from each other.



Fixing invalid distance by moving the particle -

In order to obtain a good representation in the collision detection the initial set of solutions (like moving the particles on the boundaries of a sphere) and the described secondary set of solutions must be solved more than once in order for it to converge to the expected solution. This method is called **relaxation** or Jacobi or Gauss-Seidel iteration, based on implementation (Jakobsen, 2001). The number of iterations depends on the

simulation and the amount of motion in the system. Based on testing, the number of iterations over each constraint found to suffice a good result is 3, as mentioned above.

4.2 Collisions

Four methods for collisions were used in the simulation:

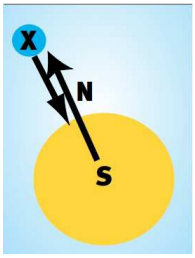
- Plane collision
- Ball collision
- Box collision
- Brute force self collision check using an octree of bounding boxes.

4.2.1 Plane collision

The plane collision is the easiest collision detection method of all: Every time a particle moves a check is being made on the particle position against the position of the plane. If the particle position at t+1 is intersecting with the plane then the particle is moved at the position declared by the plane (in order to stick on the plane surface).

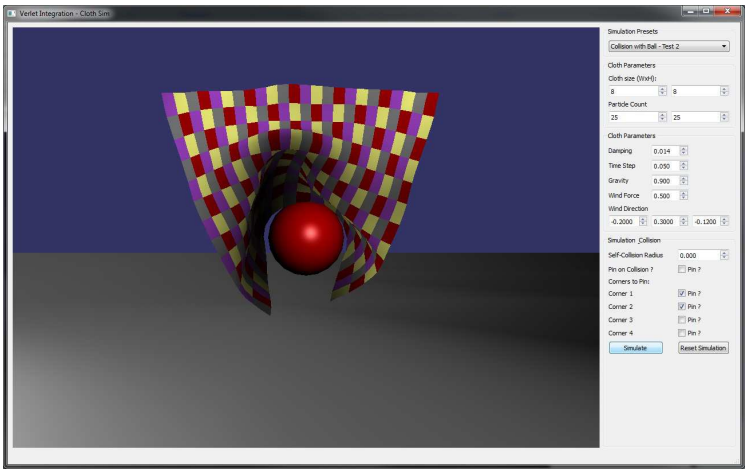
4.2.2 Ball collision

The method consists of checking every time a particle moves against the radius of the ball.



Ball collision - *Jeff Lander, 1999*

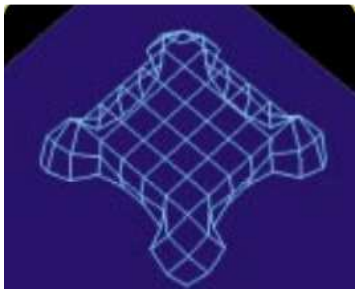
If the particle intersects the ball the collision is solved by moving the particle out of the ball along the vector from the ball's center to the particle taking into consideration that the distance is equal with the ball's radius.(Mosegaard, 2009)



Ball collision.

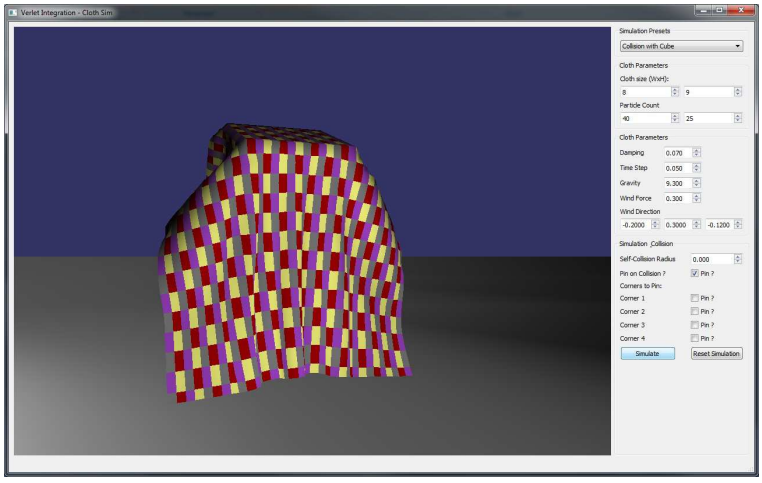
4.2.3 Box collision

Handling collisions with a box is a bit trickier. A box basically is created by 4 planes. The implementation used is as follows: Every time a particle moves it is checked to see if it is inside the box. If it is then we need to detect on which side of the box the collision happens. Once detected, the checks with the other sides are not done since a particle can only intersect one face at a time and the particle is moved on the surface of the box.



Box collision - *Jeff Lander, 1999*

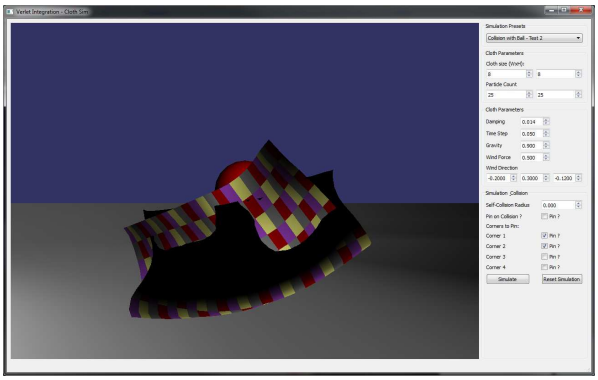
In practice, this implementation presented some problems. Using the above mentioned methods of satisfying constraints using relaxation moving the particle to the plane's position doesn't work since the string is relaxed afterwards resulting in the cloth simply passing through the box. Two quick fixed have been found: the first fix is to make the make the particle unmovable therefore stopping the integration process for it. The second one consists of stopping the integration just one step, when the collision is detected. However, the second method is highly dependent on the selected time step. If a high value is selected for the time step the particle tends to "pop" producing some visually unpleasing effects.



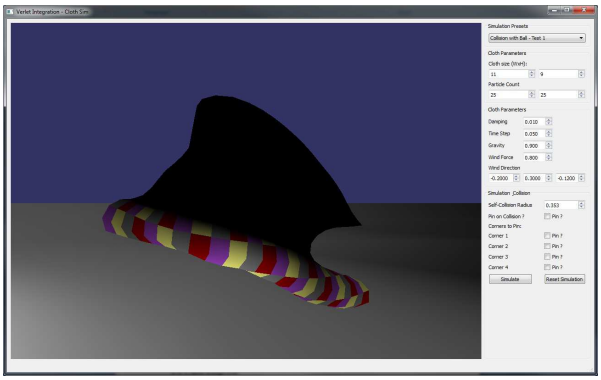
Cloth on top of a box.

4.2.4 Self collision using an octree

While this is more of a brute force method and definitely not the recommended method this method was selected from pure curiosity. Partitioning the space in eight parts is also problematic especially when opposite corners want to collide, in which case it will not work. Another problem with this implementation is the sphere radius allocated for each particle. If a radius is too big the cloth will simply do not move since the cloth is in "collision" with itself even when the simulation starts. I did manage to obtain some nice results with it however.



Self-collision disabled.

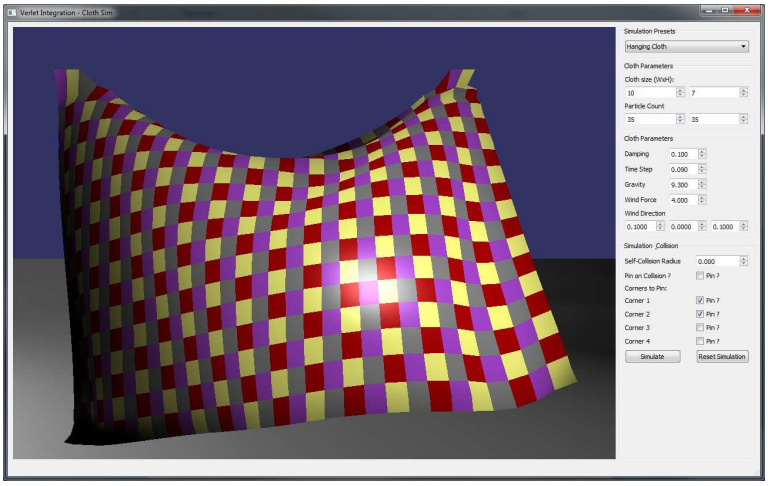


Self-collision enabled.

Initially this method was meant to be used for creating an Axis Aligned Bounding Box tree for importing objects, but was repurposed later for this purpose.

4.3 Pinning points

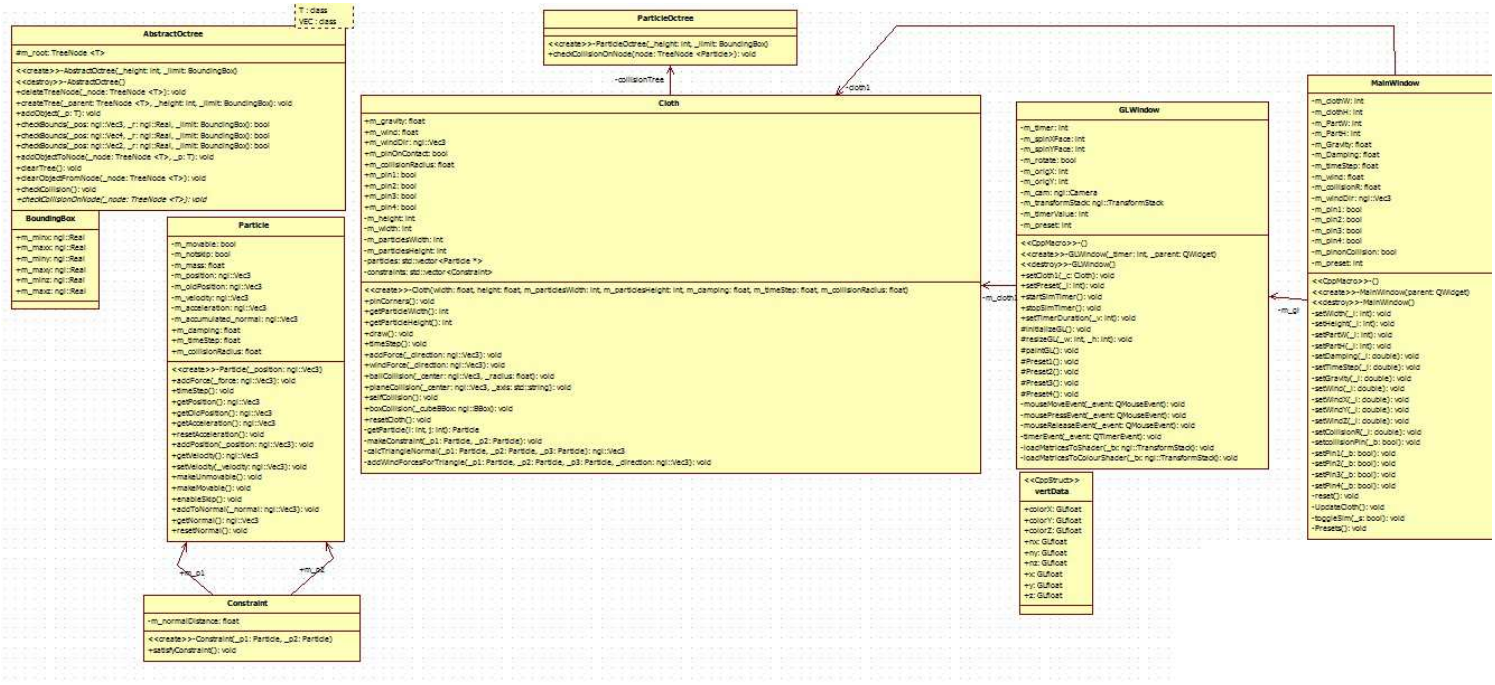
Pinning points is a trivial matter. When the cloth is being created the particles that we don't want to integrate (move in time) are set. The result is a "hanging" cloth.



Cloth pined at two corners. Simulation with wind.

5. Class diagram

The code was created using the NGL library for graphical manipulation. The class diagram is as follows:



The implementation of the described algorithm was made taking advantage of NGL for hardware accelerated graphics, Qt for the GUI interface in order to allow customizations. While the simulator offers some defined presets for various types of collision simulation, each preset can be manually tweaked in order to achieve different behaviors. Based on the above described algorithm the simulator offers some very interesting features: it supports a big number of particles and is meant for real-time rendering. While the results are very interesting and appealing visually, the simulator can be further improved by optimizing the current methods and adding new features:

- Currently a framebuffer object is created every second and destroyed after the rendering. This could be improved but creating and populating the framebuffer with date during initialization and later just manipulating the data from the framebuffer.
- Fixing a problem when sometimes the face normals are not calculated correctly for shading.
- Implement a method for subdiving the mesh geometry to obtain even more realistic results.
- Implement cloth to cloth collision
- Implement collision with custom loaded mesh using an AABB tree
- Implement adding textures to the cloth.
- Implement a friction method for making the cloth not fall from the collided object rather than pin the particles on contact.

6. References

- Baraff D., Witkin A., *Large Steps in Cloth Simulation*, COMPUTER GRAPHICS Proceedings, Annual Conference Series, 1998 (SIGGRAPH 98, Orlando, July 19–24).

- Elias H., *Cloths*. Available from: http://freespace.virgin.net/hugo.elias/models/m_cloth.htm [Last accessed 09.05.2013]

- Fiedler G., *Spring Physics* Available from: http://gafferongames.com/game-physics/spring-physics/ [Last accessed 09.05.2013]

- House, D.H. and Breen, D.E., 2000. *Cloth Modelling and Animation.* A K Peters.

- Jakobsen T., *Advanced Character Physics.* Available from:http://www.pagines.ma1.upc.edu /~susin/files/AdvancedCharacterPhysics.pdf [Last accessed 09.05.2013]

- Jakobsen T., *The making of "Hitman: Codename 47"* . Available from: http://www-scf.usc.edu/~gamedev/assets/AdvancedPhysics.ppt [Last accessed 09.05.2013]

- Lander J., *Devil in the Blue Faceted Dress: Real-time Cloth Animation.* Available from: http://www.darwin3d.com/gamedev/articles/col0599.pdf[Last accessed 09.05.2013]

- Pierce, F. T. *On the Geometry of Cloth Structure*. In Journal of the Textile Institute, 28: T45 – T97, 1937.