Cloth Animation with Collision Detection

Mara Guimarães da Silva*

Figure 1: Cloth blowing in the wind.

Abstract

This document reports the techniques and steps used to implemented a physically based animation of cloth, which included collision with objects and self collision. We also comment on the difficulties encountered, and present some results.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realist—Animation I.3.5 [Computer Graphics]: Computation Geometry and Object Modeling—Physically based modeling

Keywords: cloth,collision detection, collision response, physically based animation

1 Introduction

Physically based simulations are an important tool in creating realistic animations. In the animation of cloth one of the main issues is detecting and handling collisions. Since cloth is a deformable body and all points are on the surface, each point has to potencial to collide with the environment and with other points on the cloth at any time step. Wrong collisions handling can result in cloth sticking out from the wrong side.

For this project I had two goals. The first goal was to have a nice cloth model, which would allow wrinkles. The second goal was to properly handle collision with the environment, self-collision and friction.

*e-mail: mara.silva@gmail.com

2 Overview

This report is organized as following. First I discuss the cloth model used, as well as another option that was explored. Then issues about limiting cloth strain are discussed. Third we describe the acceleration structure used during collion detection. We conclude describing the collision algorithm.

3 Cloth Model

For choosing a cloth model I look into two models: the spring-mass model proposed by [Provot 1995], and the resposive cloth described in [Choi and Ko 2002]. Following I briefly describe both models.

3.1 Provot's model

This mass-spring model is composed of particles connected through springs. Figure 2 shows how these are layout.

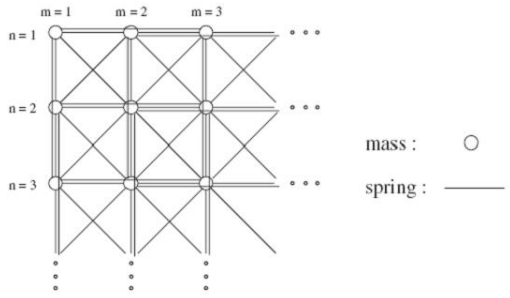

Figure 2: Mass-spring model.

Particles have mass and are the recipient of forces. Particles also have position and velocity. At each iteration new positions and

velocities are computed based on the forces applied to the particle.

Particles are connected through three different kinds of springs:

Structural springs. Deal with stretching. Connect neighbooring particles in the horizontal and vertical direction. Structural springs link particle [*i*,*j*] to particles [*i*+1,*j*] and [*i*,*j*+1].

Shear springs. Deal with shear. Connect neighbooring particles in the diagonal direction. Shear springs link particle [*i*,*j*] to particle [*i*+1,*j*+1], and particle [*i*+1,*j*] to particle [*i*,*j*+1].

Flexion springs. Deal with bending. Connect every other particles in the horizontal direction and vertical directions. Flexion springs link particle [*i*,*j*] to particles [*i*+2,*j*] and [*i*,*j*+2].

Each spring have constants which are used to compute the spring and damper forces from the springs. The integration method proposed is Euler integration.

3.2 Responsive model

This responsive model is described by [Choi and Ko 2002]. At first this model looks very similar to Provot's model since it is also composed by particles and springs. The differences are the way the springs connect the particles, and that different forces are applied to different types of springs.

Type 1 connections connect all neighbooring particles. It corresponds to structural and shear springs on Provot's model. These springs are responsible for stretching and shearing.

Type 2 connections connect every other particle. It corresponds to the flexion springs on Provot's model, but it also include particles in the diagonal direction. These springs are responsible for flexural and compression resistance.

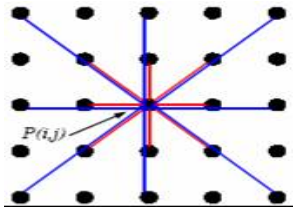

Figure 3: Responsive model. Type 1 connections are red. Type 2 connections are blue.

Figure 3 illustrates the different connections. The responsive model uses implicit integration.

3.3 Comparison

Provot's model is fairly simple to implement, so it was my starting choice. It gives nice results, but the Euler integration scheme dictates small times steps. Also there are some strain issues (discussed in section 4). I could resolve the time step issue using a implicit integration method, as proposed by [Baraff and Witkin 1998], but I decided to further investigate [Choi and Ko 2002] for their promise of allowing buckling without losing stability.

In the mass-spring model, since the forces are applied to all the springs in the same manner, trying to get buckling effects results in instability and the cloth explodes. Many times damping forces are applied in order to mantain stability. What I like about the responsive cloth model is that it applies different types of forces to different types of spring. Also, the only damping force applied is the intrisic damping property of fabrics. The result is a cloth with a not-so-rubbery look, with many wrinkles.

The responsive cloth model also makes use of implicit integration. I found their integration very close to that describe in [Baraff and Witkin 1998], the difference being that they use a second-order difference formula.

3.4 Implementation: Final choice

My final choice was implementing responsive cloth. I should quote [Baraff and Witkin 1998] saying that I *"wholeheartedly refer the reader to [Shewchuk 1994] for information on the CG method*". Unfortunately I could not get this model to work properly. I believe there is an implementatoin error in the computation of the type 2 interactions, but I could not solve it on time. The results I obtained were not stable, and also a hanging cloth shrinks in the bottom (where there are no constraints). Even though I prefered the responsive model, my final implementation uses the mass-spring model.

4 Limiting Strain

Sometimes a spring is compressed or stretched more than 10% of its rest length, which is not a realistic cloth behavior. To deal with this issue, two methods were investigated.

4.1 Position method

[Provot 1995] proposed a simple an efficient method to limit strain. After an iteration all the springs are checked for deformation. If a spring length exceeds its natural length by more than 10%, the position of the particles linked by this spring are adjusted so that the adjusted length is within this 10% limit.

4.2 Velocity method

[Bridson et al. 2002] method is similar to Provot's method. The difference it that instead of changing particles positions, momentum-conserving corrective impulses are applied.

4.3 Comparison

Using the velocity method allows to have correct positions and corresponding velocities. Comparing a non-limited cloth and a position-limited cloth leads to two conclusions: the limited cloth conserves its size but it has a rubbery look, as shown in figure 4.

The velocity method seems a promising choice. [Bridson et al. 2002] proposes that this limiting procedure should be applied using a Jacobi iteration approach. Unfortunately I could not make it work well for my final implementation.

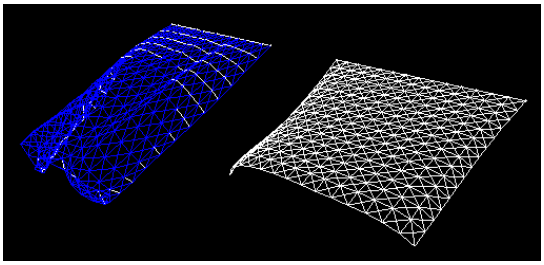

Figure 4: Unlimited vs Limited-strain cloth. Cloth on the right is limited using position method. Cloth on the left is not limited (blue represent springs that are stretched more than 10% from their rest length.

The state of the cloth after this limiting strain stage is the input for the collision algorithm. Since the velocities and positions using Provot's method do not correspond, I could not use this method together with the collision algorithm (the cloth explodes after some iterations).

5 Acceleration Structure

Collision detection is known as the major bottleneck in cloth simulations. To check for potential cloth collisions one should check all points in the cloth with all objects in the environment, and also each points in the clock with all other points in itself. This process is extremelly slow.

A acceleration structure is used to eliminate unnecessary testing. I implemented an axis-aligned bounding boxes tree. Figure 6 show how only a small part of the tree was tested during the collision test between the cloth and the gray triangle. This image shows the boxes that did not need to be tested.

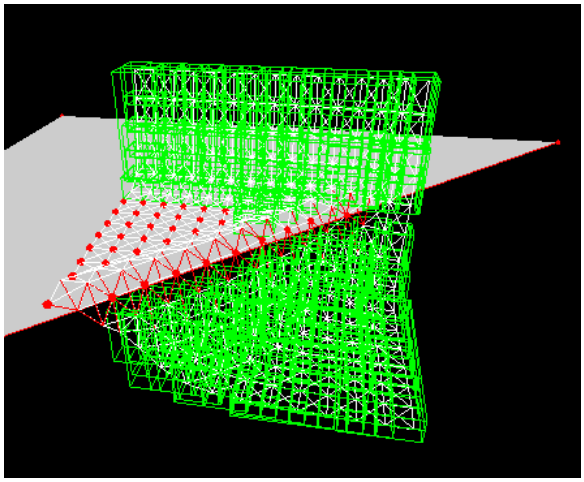

Figure 5: AABB tree in use. Green boxes represent the tree nodes that were not tested during the time step.

5.1 AABB tree for cloth.

The tree is built bottom-up in the beginning of the simulation. It starts by wraping each triangle with a bounding box. Then I took advantage of the uniform distribution of my cloth implementation and pair up triangles that have the same bounding box and compute their parent node. Next I pair up neighbooring boxes in the horizontal direction, and then in the vertical direction, successively, until I was left with only one node.

At each iteration, since the triangles move, the structure needs to be recomputed. It is not necessary to build the tree again, only update it. The update process is made bottom-up, updating the leafs, then its parents, and so on.

5.2 AABB tree for meshes.

The objects in the environment are represented by meshes, which can have many triangles. I decided to apply an AABB hierarchy to the rigid objects in the scene in order to efficiently test for collision.

The AABB tree for rigid objects are build top-down, since I do not have information about the triangles distribution. I constructed the tree following the description in [van den Bergen 1997]. Update are necessary only if the rigid object is moving, and should be done in the same way as the update for cloth.

6 Intersection Testing

Before explaining the collision algorithm, I am going to comment on the collision tests I implemented for this project. Collision were detected using edge-edge and point-triangle proximity tests, and also a trajectory projection test.

6.1 Point-Triangle proximity test.

To test if a point p is closer to a triangle $t_1t_2t_3$ than a certain distance h, I first compute the distance of the point from the triangle plane. If the distance d is smaller than h I project the point into the triangle plane and compute the barycentric coordinates. If the barycentric coordinates are inside the triangle than a collision is recorded.

6.2 Edge-Edge proximity test.

To check if an edge e_1e_2 is closer to an edge f_1f_2 than a certain distance h, I implemented the distance between segments test described in [Sunday].

6.3 Geometric collision test.

Given four points and their velocities, the geometric collision test consist in determing the time t when the four points are coplanar. If t is inside the current time step, a proximity test is performed using the position of the points at t.

Given points $\vec{x}_1,\vec{x}_2,\vec{x}_3,\vec{x}_4$, their velocities $\vec{v}_1,\vec{v}_2,\vec{v}_3,\vec{v}_4$, and defining $\vec{x}_{ij}=\vec{x}_i-\vec{x}_j$, the time t when the points will be coplanar are the roots of the cubic equation [Provot 1997],

$$(\vec{x}_{21}+t\vec{v}_{21})\times(\vec{x}_{31}+t\vec{v}_{31})\cdot(\vec{x}_{41}+t\vec{v}_{41})=0$$

To solve the cubic equation

$$a_3 t^3 + a_2 t^2 + a_1 t + a_0 = 0$$

where

$$\begin{aligned} a_3 &= f_x k + f_y l + f_z m \\ a_2 &= e_x k + e_y l + e_z m + f_x n + f_y o + f_z p \\ a_1 &= e_x n + e_y o + e_z p + f_x q + f_y r + f_z s \\ a_0 &= e_x q + e_y r + e_z s \end{aligned}$$

and

$$\begin{aligned} a &= \vec{x}_{21} \\ b &= \vec{v}_{21} \\ c &= \vec{x}_{31} \\ d &= \vec{v}_{31} \\ e &= \vec{x}_{41} \\ f &= \vec{v}_{41} \\ k &= b_y d_z - b_z d_y \\ l &= b_z d_x - b_x d_z \\ m &= b_x d_y - b_y d_x \\ n &= a_y d_z - a_z d_y + b_y c_z - b_z c_y \\ o &= a_z d_x - a_x d_z + b_z c_x - b_x c_z \\ p &= a_x d_y - a_y d_x + b_x c_y - b_y c_x \\ n &= a_y c_z - a_z c_y \\ o &= a_z c_x - a_x c_z \\ p &= a_x c_y - a_y c_x \end{aligned}$$

I used the Cardano's formula described in [Weisstein].

7 Collision Algorithm

The collision algorithm I implemented is similar to what as proposed by [Bridson et al. 2002], so following is just a brief description. It is composed of two stages. The first stage try to prevent collision from happening, and the second stage deal with actual geometric collisions. This two-stage approach has the advantage of elliminating almost all collision in the first stage, leaving only a few collision for the second stage, which is more expensive. Following is a brief description.

The velocity used for the collision computation is the average velocity, which is

$$\bar{\mathbf{v}}^{n+1/2} = (\bar{\mathbf{x}}^{n+1} - \mathbf{x}^n)/\Delta t$$

where $\mathbf{x}^n$ is the position of the particle at the beginning of the time step, and $\mathbf{x}^{n+1}$ is the candidate particle position at the end of the time step, computed by the cloth internal dynamics.

7.1 Proximity detection and response

To detect proximity we first update the AABB tree, expanding the bounding box to consider the thickness of the cloth. Then we check for collision between cloth AABB tree and the other AABB trees in the scene. The leafs of intersecting bounding boxes are checked for proximity. Following the algorithm looks for self intersection, looking for intersecting bounding boxes inside the cloth tree. When a proximity is detected two kinds of response are applied:

Inelastic impulse. If a proximity is detected, and if the points are approaching, I stop the imminent collision applying a inelastic impulse in the normal direction.

Spring based repulsion force. A impulse in the normal direction is applied to close points. This impulse is proportional to the overlap of the objects.

7.2 Geometric collision detection and response

To detect geometric collision the AABB tree is updated in order to have the bounding boxes wraping the volume cointaining a triangle at the beginning and at the end of the time step. The geometric collision test is performed for intersecting bounding boxes, and the time t of the possible collision is computed.

The positions are projected for t, using the average velocity. The proximity tests are performed for these points, using a small thickness to account for rounding error.

8 Results

Overall I was very happy with the results I obtained from this project. Following are some images from the simulation.

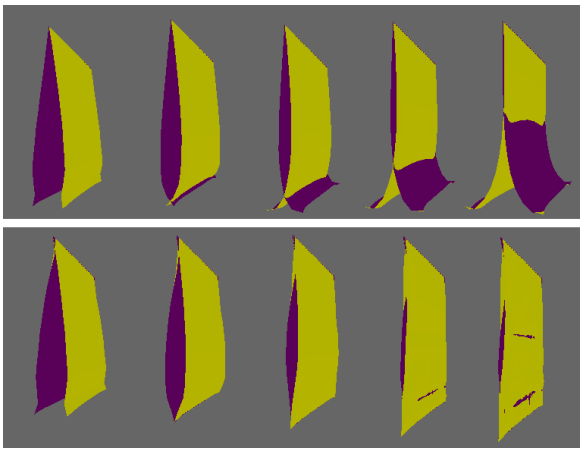

Figure 6: No self collision *(top)* and self collision*(bottom)*.

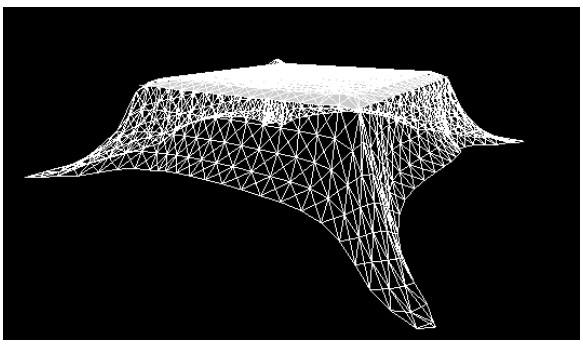

Figure 7: Cloth falling on a table.

9 Future Work

As future work I plan to implement all sugestions from [Bridson et al. 2002], which includes friction, limiting strain using the velocity method, and a correct computation of the final velocity (using conjugate gradient method).

As mentioned in section 3.4 my choice of cloth model is the resposive model, so I plan to fully implement it and integrate it with the collision algorithm.

9.1 Future improvements.

9.1.1 AABB tree

I found that testing for self-collision was extremely slow. I believe that the problem is in unnecessary tests of adjacent triangles. I believe there is space for improvement in the construction of the AABB trees.

One way of improving this would be to further investigate the way of grouping triangles described by [Volino and Magnenat-Thalmann 2000], which I believe could result in a more efficient structure.

Another idea for improvement is illustrated by figure 8. During collision detection I check for intersecting bounding boxes. The bounding boxes of the red and blue triangle will always intersect, resulting in an enourmous number of unnecessary collision testing. If these triangles had the same bounding box these tests could be substituted for a simplier collision test (something like a normal-based test).

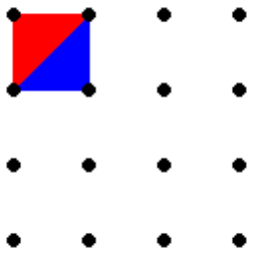

Figure 8: AABB tree construction. Red and blue triangles should have same bounding box.

9.1.2 Geometric collision test

I plan to further investigate a better way of checking for geometric collision. Professor Steve Rotemberg suggested testing point-triangle geometric collisions by making the triangle stationary, creating a triangle space, and projecting the point initial and final positions in this space. A collision would be detected if the the point initial and final positions were in oposite sides of the z-plane, for example. I think this test could be faster than what I have implemented, and I plan to further investigate it.

10 Comments

My implementation of the collision algorithm is not failsafe, but I believe that with some small corrections I can obtain significant improvements. Also, implementing all items listed as future work would result in a realistic cloth animation.

References

BARAFF, D., AND WITKIN, A. 1998. Large steps in cloth simulation. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 43–54.

BRIDSON, R., FEDKIW, R., AND ANDERSON, J. 2002. Robust treatment of collisions, contact and friction for cloth animation. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 594–603.

CHOI, K.-J., AND KO, H.-S. 2002. Stable but responsive cloth. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 604–611.

PROVOT, X. 1995. Deformation constraints in a mass-spring model to describe rigid cloth behavior. *Graphics Interface*, 147–155.

PROVOT, X. 1997. Collision and self-collision handling in cloth model dedicated to design garment. *Graphics Interface*.

SHEWCHUK, J. R. 1994. An introduction to the conjugate gradient method without the agonizing pain. Tech. rep., Pittsburgh, PA, USA.

SUNDAY, D. Distance between lines and segments with their closest point of approach. In *http://softsurfer.com/Archive/*.

VAN DEN BERGEN, G. 1997. Efficient collision detection of complex deformable models using aabb trees. *J. Graph. Tools 2*, 4, 1–13.

VOLINO, P., AND MAGNENAT-THALMANN, N. 2000. *Virtual Clothing*. Springer.

WEISSTEIN, E. W. Cubic formula. In *MathWorld – A Wolfram Web Resource*, http://mathworld.wolfram.com/CubicFormula.html.