Cloth Simulaiton with Discrete Mass-Spring and Partical System

Giancarlo Gonzalez
University of Illinois at Chicago
ggonza20@uic.edu

Muxuan Wang
University of Illinois at Chicago
mwang30@uic.edu

1. INTRODUCTION

The goal of the project is cloth simulation. This problem has been existing for more than 2 decades. Cloth simulation is modeled as a partial differential equation [1].

$$\ddot{x} = M^{-1}\left(-\frac{\delta E}{\delta x} + F\right) \quad (1)$$

In equation 1 above, x is a vector representing the geometric position of a cloth, and M is the mass distribution of it. E is a function of x and F includes forces like air, contact, etc which are acting upon the cloth.

Physical models and numerical solvers are the basis of current cloth simulation engines. There are several physical models that have been employed for cloth animation, ranging from discrete mass-spring and particle system to finite element solutions for continuous cloth models. Our Cloth simulation was using the disrete mass-spring and particle system was describled in detail in this report.

Our Cloth simulation was based on Dr.Mosengaard's [3] implementation. Our approach of modeling cloth is discretized by a polygonal mesh. In discrete models, the mesh topology defines, how the particles (vertices) interact and exert forces on one another. The forces on each particle of the cloth mesh are computed depending on its position and velocity, and the position and velocities of a set of particles within its topological neighborhood. In a mass-spring system, we uses a two triangler mesh to form a rectangular mesh in which the particles are connected by structural springs to counteract tension, diagonal spring for shearing, and interleaving springs for bending.

Cloth is permissive in allowing bending and shearing motions which determines that there is a "stiff" underlying the differential equation of motion [4]. According to Baraff et al., explicit integration method is not suited for solving stiff equations as it requires lots of small steps to move the simulation forward in time. He proposed using implicit integration method to improve the performance limit inherent in the explicit method. So in our report, we gives a detail introducton on inmplicit integration method.

2. SIMULATION OVERVIEW

2.1 Internal Forces and Particle Movement

The most important forces in the system are the internal cloth forces which determines much of the cloth's characteristic behavior. Stretching, shearing and bending respectively. Streching or compression is caused by displacement along warp or weft direction. Shear can be seen as displacement along diagnoal direction and bend is the curvature of cloth surface. As shown in Figure 1. Breen et al.[2] describes as the in-plane shearing and out-of-plane bending forces in cloth. We formulate the shear force on a per triangle basis, and the bend force on per edge basis- between pairs of adjacent triangles. The strongest internal force - stretch force is also formulated per triangle.

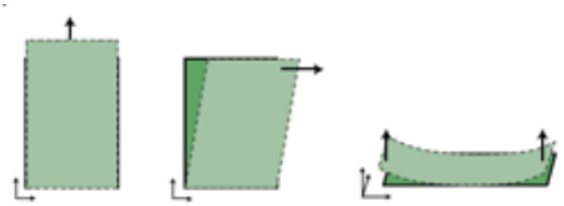

Figure 1: Stretching, shearing and bending respectively.

The cloth simulation, in our example, is the interconnections between particles by a spring force or constraint. These particles can move around by some external forces being acted on them. In order to affect particles with forces we must use Newton's second law and transform it into acceleration = force/mass as shown in Figure 2 and is part of our Particle class.

```
void addForce(vec3 f) {
    acceleration += f/mass;
}
```

Figure 2: Function to add a force to a particle

To get a position, acceleration can be integrated twice using a numerical integration. We have used the Verlet integaration to solve the at this point. The verlet integaration is a numerical method used to integrate Newton's equations of mothion. According to Verlet[5], it provides a good numerical stability, as well as other properties that are important in physical systems such as time-reversibility and preservation

of the symplectic form on phase space, at no significant additional computational cost over the simple Euler method. A simple explaination of how verlet integration works is given. Newton's equation of motion for conservative physical systems is (explaination from wikipedia):

$$M\ddot{\vec{x}} = F(x(t)) = -\nabla V(\vec{x}(t)) \tag{2}$$

where,
• t is time,
• $\vec{x}(t) = (\vec{x_1}(t), \ldots, \vec{x_N}(t))$ is the ensemble of the position vector of N objects,
• V is the scalar potential function,
• F is the negative gradient of the potential giving the ensemble of forces on the particles,
• M is the mass matrix

To discretize and numerically solve this initial value problem, a time step $\Delta t > 0$ is chosen and the sampling point sequence $t_n = n\Delta t$ considered. The task is to construct a sequence of points $\vec{x}_n$ that closely follow the points $\vec{x}(t_n)$ on the trajectory of the exact solution.

Euler's method uses the forward difference approximation to the first derivative in differential equations of order one, Verlet Integration can be seen as using the central difference approximation to the second derivative to the first derivative in differential equations of order one, comparing to Euler's method which uses the forward difference approximation.

$$\begin{aligned} \frac{\Delta^2 \vec{x}_n}{\Delta t^2} &= \frac{\frac{\vec{x}_{n+1}-\vec{x}_n}{\Delta t} - \frac{\vec{x}_n-\vec{x}_{n-1}}{\Delta t}}{\Delta t} \\ &= \frac{\vec{x}_{n+1} - 2\vec{x}_n + \vec{x}_{n-1}}{\Delta t^2} = \vec{a}_n \\ &= A(\vec{x}_n) \end{aligned} \tag{3}$$

To obtain the next position vector from the previous two, we have:

$$\vec{x}_{n+1} = 2\vec{x}_n - \vec{x}_{n-1} + \vec{a}_n\,\Delta t^2, \qquad \vec{a}_n = A(\vec{x}_n). \tag{4}$$

In our simulaiton, the verlet integration is done by making a call to the function timeStep which is shown in Figure 3. Verlet integration simply moves our particle from one position to the other, and the movement is scaled using a time step because given a small time step, the movement of the particle should also be small. Now the next time timeStep is called the particle should keep moving in that direction that the old one has been moved in. Damping is introduced so that there is some loss to velocity due to air resistance.

```
/* Given the equation force = mass * acceleration, the next position is
found through verlet integration */
void timeStep() {
    if(movable) {
        vec3 temp = pos;
        pos = pos + (pos-old_pos)*(float)(1.0f-DAMPING) + acceleration*(float)(
            TIME_STEPSIZE2);
        old_pos = temp;
        resetAcceleration(); // acceleration is reset since it HAS been
        translated into a change in position (and implicitely into velocity)
    }
}
```

Figure 3: Two integrations to the acceleration through verlet integration.

2.2 Constraints

In this section, we describe how constraints are imposed on individual cloth particles. The constraints we discuss in this section are contact constraints between a solid object and a particle. Particles thus be attached to a fixed or moving point in space, or constrained to a fixed or moving surface or curve. At any give time of the simulation, a cloth particle is either completely unconstrained, or the particle may be in contrained in one, two or there dimensions. If the particle is completely constrained, we may just explicity setting some value to the particle's velocity. If there are only one or two constraints, we are constraining the particles along either one or two mutually orthogonal axes.

In order to keep the particles connected in a grid and behaving realistically we use structural, shear, and bending constraints. These constraints are made in the constructor of the class as seen in Figure 4. Structural constraint keeps the particles connected in a grid. Shearing constraint helps the particles move in parallel of each other. Bending constraints is to help resist the bend of the material.
pecify distance constraint using flexible spring model:

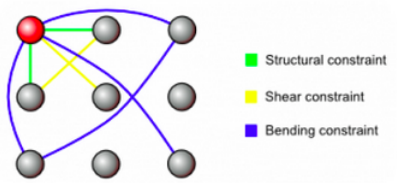

Figure 4: Particle (red) constraints in the constructor of the cloth class.

We speicfy distance constraint using flexible spring model. Therefore, our partical system with contraints will be like:

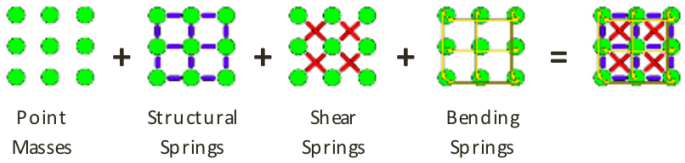

Figure 5: Mass-Spring system for cloth.

A lot of the information in the constraint class comes from Hook's spring law. Two particles are held together by having a spring exert a force between them, a distance between them, the resting length of the spring, the stiffness of the spring (a constant), a damping constant (for a force of friction), and the velocity of the particles. A Mass-Spring System is therefore applied. The following figure 6 shows a liner strain model of a spring motion based on Hook's Law.

This spring force allows for simulation of how far a material can be stretched and how it bounces back to its original shape by the resting length of the spring. This is called a dynamic particle simulator and it changes every time step of the simulation. As explained above in the particles section, we add external force and a resting distance to each particle

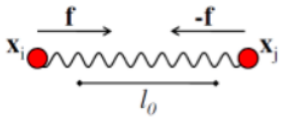

Figure 6: Spring motion based on Hook's law.

to add constraint.

2.3 Dynamic behavior of the cloth

In order to have some interesting behavior for the cloth, we have added external forces, such as a wind force, gravity, and collision with an object. Also, controls are available, as demonstrated in the video, to add or remove gravity, the object colliding, and the wind force to see how they react without or just on their own.

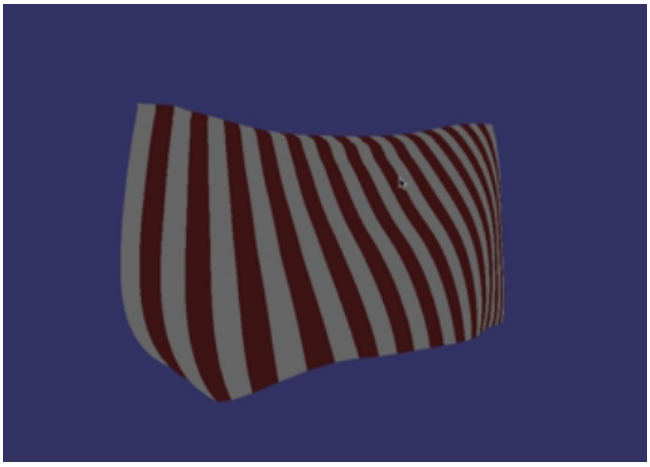

Figure 7: Cloth interacting with wind.

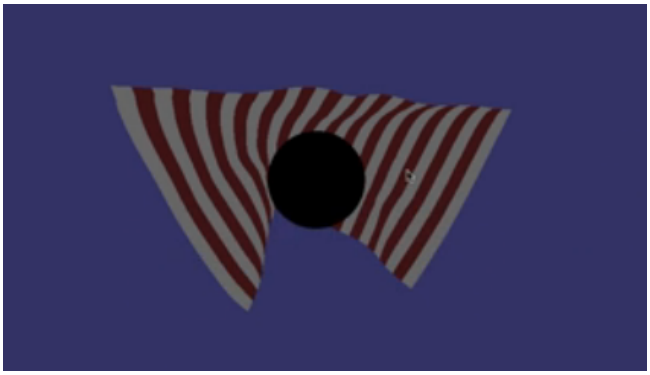

Figure 8: Cloth interacting with ball.

Gravity is an acceleration vector pointing in the downward direction added to all particles and is added by calling the addForce function with a vector. In our case the vec3(0.0, 0.2, 0.0) times the time step size adds it.

The object that we are showing colliding with the cloth is a sphere. Collision is detected by checking if a particle is within the object. By giving the position of the sphere and its radius, this is easily checked, if a particle is found to be inside the radius we move the particle the radius minus the distance where it is found. We take into account the fact that in order to avoid the ball protruding the cloth, the radius of the ball given to the function ballCollison is a bit bigger than the ball's actual radius. Adding wind was a bit more difficult. Knowing that we draw the particles by connecting them and making up triangles with them to draw cloth, wind is just a force acting on the normal of these triangles and should be proportional to the angle its coming at. We found a simple function in the tutorial by Moosegaards that adds forces to the triangles as shown in Figure 9.

```
/* A private method used by windForce() to calcualte the wind force for a single
triangle
defined by p1,p2,p3*/
void addWindForcesForTriangle(Particle *p1,Particle *p2,Particle *p3, const vec3
    direction)
{
    vec3 normal = calcTriangleNormal(p1,p2,p3);
    vec3 d = glm::normalize(normal);
    vec3 force = normal * (glm::dot(d, direction));
    p1->addForce(force);
    p2->addForce(force);
    p3->addForce(force);
}
```

Figure 9: Force added to a triangle defined by the three particles given.

3. LARGE TIME STEP AND IMPLICIT INTEGRATION

The bottle-neck in most cloth simulation systems is that time steps must be small to avoid numerical instability. In our cloth simulation system, we also uses explicite integration method to slove the equations which requires lots of small steps to move the simulation forward in time. However, using small time step really slow down our simulation. To make the simulation run smoother, we have to sacrifice the simulaiton realism by reducing the total number of particales. In our current simulation, we only use 30 * 30 particles.

Cloth is permissive in allowing bending and shearing motions which determines that there is a "stiff" underlying the differential equation of motion [4]. According to Baraff et al., explicit integration method is not suited for solving stiff equations as it requires lots of small steps to move the simulation forward in time. He proposed using implicit integration method to improve the performance limit inherent in the explicit method. For example, given the known position $x(t_0)$ and velocity $\dot{x}(t_0)$ of the system at time t_0, the goal is to determine a new position $x(t_0+h)$ and velocity $\dot{x}(t_0+h)$ at time t_0+h. To compute the new state and velocity using an implicit technique, we must first transform equation 1 into a first-order differential equation. The new first-order differential equation is:

$$\frac{d}{dt}\begin{pmatrix} x \\ \dot{x} \end{pmatrix} = \frac{d}{dt}\begin{pmatrix} x \\ v \end{pmatrix} = \begin{pmatrix} v \\ M^{-1}f(x,v) \end{pmatrix}. \quad (5)$$

To simplify notation, we will define $x_0 = x(t_0)$ and $v_0 =$

$v(t_0)$. And also define $\Delta x = x(t_0 + h) - x(t_0)$ and $\Delta v = v(t_0 + h) - v(t_0)$. The linear system is:

$$\begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = h \begin{pmatrix} v_0 + \Delta v \\ M^{-1}(f_0 + \frac{\partial f}{\partial x}\Delta x + \frac{\partial f}{\partial v}\Delta v) \end{pmatrix}. \quad (6)$$

Taking the bottom row of equation 6 and substituting $\Delta x = h(v_0 + \Delta v)$ yields

$$\Delta v = hM^{-1}(f_0 + \frac{\partial f}{\partial x}h(v_0 + \Delta v) + \frac{\partial f}{\partial v}\Delta v). \quad (7)$$

which we can solve for Δv. So given Δv, we trivially compute $\Delta x = h(v_0 + \Delta v)$.
According to Baraff, by enforcing constarints on individual cloth particles with the above impicit integration method, the cloth simulaiton system can stably take large time steps. He also introduced a simple treatment of damping forces. The key issue here is, by using implicit integration method, the solution of $O(n) \times O(n)$ sparse linear system. Their simulator enforces constraints without adding penalty terms in energy function E or Lagrange-multiplier forces to F. They used a modified version of Conjugate Gradient (CG) method to solve the linear system introduced by implicit integration. One of the properties of their solution is the constrains are maintained without being affected by the number of iterations taken by linear solver. They also showed a method to dynamically adapt the size of time steps over the process of simulation.

4. CONCLUSION

Physical models and numerical solvers are the basis of current cloth simulation engines. There are several physical models that have been employed for cloth animation, ranging from discrete mass-spring and particle system to finite element solutions for continuous cloth models. Our Cloth simulation was using the disrete mass-spring and particle system was describled in detail in this report.

Our Cloth simulation was based on Dr.Mosengaard's [3] implementation. Our worst fears being realized, it did not work with modern OpenGL and for obvious reason with Aluminum. First off, a class for vector multiplication, division, normalization, etc. was defined, but the vec3 class that glm/math has defined is much better, especially when sending it to the shaders. We had to replace all code that used that vector class and add the new commands that modern OpenGL has used. Also, shaders were not at all in use by the program in the tutorial so we wrote simple ones that added lighting to the scene. It is nothing special, just simply adds an ambient and diffuse lighting. In order to use these shaders in Aluminum, we had to go throughout the code adding the model, projection, and view for the cloth creation. The ball was simple since the addSphere was provided by the library already. I would recommend adding color to the ball but for time sake did not. I believe it is as simple as adding color to each triangle by using the color array in the MeshBuffer. The most difficult piece was drawing the cloth. I wrote the cloth code in OpenGL instead of using the provided Aluminum methods and it has worked so far. By simply adding a struct called Vertex that contained the positions, normals, and colors of the triangles we were able to get the triangles drawn using OpenGL.

Although we found out that using explict integration methods requires small steps to move the simulation forward in time and implict integration methods could solve this probem, we weren't able to implement at this time due to time limitation. However, we gave a detailed explaination about how the implict integration methods works in cloth simuation.

5. REFERENCES

[1] Baraff, D., and Witkin, A. Large steps in cloth simulation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1998), SIGGRAPH '98, ACM, pp. 43–54.

[2] D.E. Breen, D. H., and Wozny., M. Predicting the drape of woven cloth using interacting particlbnbbnbbes. In *Computer Graphics (Proc. SIGGRAPH)* (1994).

[3] Mosegaard, J. Mosegaards cloth simulation coding tutorial.

[4] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, 3 ed. Cambridge University Press, New York, NY, USA, 2007.

[5] Verlet, L. Computer "experiments" on classical fluids. i. thermodynamical properties of lennardâĹŠjones molecules. In *Physical Review 159: 98âĂŞ103. doi:10.1103/PhysRev.159.98* (1967).