Cloth Simulation

Naeem Quddus (nhq58)
Keivaun Waugh (ksw734)

May 8, 2017

1 Math Overview

1.1 Internal Dynamics

We referenced two papers for deciding the forces to include in our cloth simulation. The first was *Large Steps in Cloth Simulation* by Baraff and Witkin. In this paper, they argue that cloth can be accurately modeled with large time steps if an implicit time integrated such as implicit euler is used. They chose to model the cloth with three distinct internal forces: stretch, shear, and bend. We used the second paper, *Implementing Baraff & Witkin's Cloth Simulation* by Pritchard to go through the force derivations without using higher order tensors by differentiating each energy condition on a component by component basis. The energy function associated with a condition is defined as

$$E_C(x) = \frac{k}{2} C(x)^T C(x)$$

where k is a stiffness constant. The force of a condition can then by obtained by taking the negative transpose of the differential of the energy.

1.1.1 Stretch Force

To model the stretch and shear forces, we kept track of a u,v parameterization of each point. w_u and w_v describe the stretch or compression at a point in the u or v direction by differentiating a mapping of plane coordinates to world space coordinates. They are defined as follows:

$$(w_u, w_v) = (\Delta x_1, \Delta x_2) \begin{bmatrix} \Delta u_1 & \Delta u_2 \\ \Delta v_1 & \Delta v_2 \end{bmatrix}^{-1}$$

The stretch condition:

$$C(x) = a \begin{bmatrix} ||w_u(x)|| - 1 \\ ||w_v(x)|| - 1 \end{bmatrix}$$

increases the force on a particle as it is deformed from its local u, v parameterization.

1.1.2 Shear Force

The condition for the shear force is defined as

$$C(x) = a w_u(x)^T w_v(x)$$

where a is the area of the triangle in the uv plane.

1.1.3 Bend Force

The bend force condition is simply the angle between adjacent pairs of triangles. This can be calculated using the `atan2` function. Differentiating this condition is extremely painful, but luckily the second paper walks through these calculations.

1.2 Collisions

We modeled our collisions based on the paper *Robust Treatmentof Collisions, Contact and Friction for Cloth Animation* by Bridson et al. Whenever we detect a collision, we apply an inelastic impulse to stop the penetration and a small spring force to repulse the cloth. The repulsion force helps to eliminate some collisions as the cloth comes in more contact with itself. This provides a speed boost in situations that would normally grind the simulation to a halt. The forces are applied directly to the velocities of the points in the mesh and are weighted by the barycentric coordinates of the vertices in the triangle.

1.3 Collision Detection

We chose to omit detecting edge-edge collisions only tracked point-face collisions. We initlially wrote code that detected the edge-edge collisions, but our collision handling code was buggy and improperly handled these. We removed the code since point-face collisions are enough to produce reasonable results and is a common shortcut used in practice.

1.3.1 Inelastic Impulses

The inelastic impulse is defined as

$$I_c = \frac{mv_n}{2}$$

where m is the mass of the colliding point, and v_n is the relative velocity in the face-normal direction. The impulse is only applied when the relative velocity indicates that the objects are moving towards each other. Bridson assumes in his paper that all points have the same mass. We do not make this assumption and instead calculate point masses based on the total area of the triangles that it composes.

1.3.2 Repulsion Spring Force

The repulsion spring force is Δtkd where k is a repulsion constant set to be equal to the stretch coefficient. d is the distance that the point has penetrated into the thickness of the cloth. It is visually pleasing to avoid repelling away within a single time step, so the repulsion force is capped at a max value of $m(\frac{.1d}{\Delta t} - v_n)$.

1.3.3 Cloth-Other Object

Detecting a collision between a cloth point and a sphere is simply seeing if the distance between the point and the sphere center is smaller than the sphere radius plus some thickness. Detecting a collision between a cloth point and an axis aligned cube is doing the same on a per component basis. We fix all non-cloth objects in place, so the magnitude of the impulses and repulsion forces have to be adjusted to account for the object not moving. We do not do repulsion forces for this type of collision since the detection is not time expensive.

2 Implementation

2.1 Code Structure

The entrypoint of our code is in the file `main.cpp`. This initializes the simulation and the viewer code on separate threads, because we found that the rendering was bottlenecking the simulation code for simulations with small numbers of triangles. The file `viewer_wrapper.cpp` handles all the rendering by generating the geometry from the simulation code and making the appropriate calls to the libigl viewer.

The simulation code begins by generating a cloth (whose code is located in `cloth.cpp`) and some number of other static objects that can be spheres or cubes (whose code is located in `object.cpp`). The flow of the simulation class matches that of the other projects we have completed. On every time step, we build the configuration of the cloth into local variables, perform implicit euler time integration, handle collisions, and unpack the configuration back into the cloth geometry. If newton's method did not converge, we half the time step and retry.

2.2 libigl

We use the libigl library to handle all the rendering and camera controls. libigl allows you to pass it a set of vertices and faces, and will construct the geometry with those values. It also allows you to pass in a set of colors for either vertices or faces. The GUI allows us to update parameters on the fly without recompiling our code like we did in the first milestone.

2.3 Mesh Creation

We used the "Triangle" library for generating 2D triangle meshes of different sizes for our cloth. This generates a non-uniform mesh that avoids behaving rigidly along canonical axes. We used meshlab to generate the geometry for the sphere and box files that are located in `src/resources`. These have a `.node` file which specifies the vertex locations and a `.ele` file that specifies the faces. We kept all points within a -1 to 1 bounding box and adjusted the scale and origin as we found fit.

2.4 BVH

All internal dynamics of the cloth run in linear time, so collisions dominated the run time of our program. Performing a quadratic comparison between pairs of triangles for collision detection became intractable as the number of triangles increased. We implemented a BVH using axis-aligned bounding boxes to prune out calls to our collision detection code. We contruct the BVH on each simulation step since the cloth particles move. Building the BVH is an $O(n \log n)$ time investment and each subsequent call to the BVH for intersection takes $O(\log n)$ time. This drastically reduces the runtime of the collision detection.

3 Goals

We originally planned on implementing the internal dynamics of the cloth, writing OpenGL rendering, handling collisions, and tearing of the cloth. We implemented all of these except for tearing. Due to time constraints, we omitted some features of cloth collisions that were described in the collision paper. These include: edge-edge collisions, friction, midstep collision handling, rigid impact zones, and post-processing with subdivisions. Omitting the edge-edge collisions and midstep collision handling means that the cloth with sometimes penetrate itself or other objects, but this is not an issue aside from visual artifacts. We also planned to texture map frames from Rick Astley's "Never Gonna Give You Up" music video onto the cloth surface. This was ommitted due to time constraints.

4 Extra Credit

On our honor, we affirm that we both filled out the course evaluations.