Names: Jimmy Xin (jjx88), Gavin Wang (gw7775)

Final Project

For our final project, we implemented a *mass-spring system* that we use to simulate *cloth*.

Mass-Spring System

We use a mass-spring system to simulate cloth. First, some preliminaries on mass-spring systems. This is a form of *physical simulation*, in particular a *particle simulation*. We maintain some point masses and then describe the forces acting on these point masses. In particular, instead of attempting to mathematically model the path of every particle as a function of time, we'll instead provide some basic physical principles and then just allow the system to "play out."

The primary interesting physical principle we use is *Hooke's Law*, which describes the behavior of springs. Given two points, we can attach a spring between them and model the force using the equation `F = -kx`, where `F` is the force, `k` is the spring constant, and `x` is the displacement between the two points.

Then, for each point, we can maintain its acceleration, velocity, and position. To simulate the system, we'll simply step through time (using very small timesteps) and use Newton's second law (`F = ma`) to update the acceleration, and some kinematic equations to update the velocity and position.

Cloth

We model a cloth simply as a network of points and springs connecting those points. In particular, we add:

- *structural springs*: Connects adjacent (horizontally or vertically) points and pulls the cloth together into one object.
- *sheer springs*: Connects diagonally adjacent points, prevents the cloth from collapsing into a line.
- *flex springs*: Connects adjacent (horizontally or vertically) points two spaces apart. This prevents the cloth from folding on top of itself.

Collisions

We do some rudimentary collision tracking, such as colliding with the floor or with a sphere. For every point, we check if it will intersect any objects. If it does, we project it back to the surface of the object.

Self-Intersections

We also have an option for the cloth to prevent any self-intersection issues, which ruins the immersion of the simulation. We do this by checking the distance between every pair of points, and making sure that they are pushed apart if the distances are too close.

Rendering

For the floor, we use the same rendering process as menger sponge, and for the sphere, we use and adapt some reference code taken from Github (https://github.com/ndesmic/geogl/blob/v3/js/lib/shape-gen.js), as rendering geometry is not the focus of our project.

Implementation Details

Integration Estimators

There are a few different ways of stepping through time and updating physical properties. We first tried to use Euler's method to estimate the effects of acceleration and velocity on each particle: $x_{n+\Delta t} = x_n + hf(t_n, x_n)$. This turned out to be incredibly unstable because conservation of energy does not hold. We then implemented Verlet integration $x_{n+\Delta t} = 2x_n - x_{n-\Delta t} + a_n \Delta t^2$, which ended up being stable enough to use for all of our calculations.

Other forces

However, with only spring forces, our simulation will oscillate forever. In order to make the system more realistic, we modeling *energy loss* by adding a *drag force*. This force resists the particle's current direction of motion and its magnitude scales with the velocity of the particle. Drag force is also especially useful in order to simulate the fluttering of the cloth. In addition, we find that drag for is useful for mitigating error compounding which would've caused the system to spiral out of control (it appears to implode or explode). Next, we add a constant force for *gravity*, which lets our cloth fall to the ground instead of floating in the air. Finally, we add an optional *wind* force, which is modeled by a constant function that varies globally. Together, these forces model cloth physics well enough to make a pretty realistic simulation.

Rendering

We render the system by converting the particle system into a triangle mesh. Since the cloth is arranged in a grid-like structure, for each "cell" (a square defined by four points), we can simply split that into two triangles and render those two triangles. We use a similar method to the Menger sponge, where we'll duplicate points when they need to be used for multiple triangles.

We implement smooth shading following the instructions here (https://computergraphics.stackexchange.com/questions/4031/programmatically-generating-vertex-normals). The idea here is that we should assign vertex normals to the average of the normals of all the faces that the vertex borders. This ensures that the cloth is smooth and hides the internal triangle mesh structure.

To turn smooth shading on and off, use the given checkbox. Smooth shading is by default on. When smooth shading is off, the scene instead uses flat shading, e.g. each triangle is a flat plane. Additionally, we only render half the triangles so that you can see exactly where the vertices are and how the triangles are being generated.

Limitations

There are several limitations of this approach. Importantly, this is not a physically realistic representation of cloth, but in many places is simply "good enough." For example, we implemented linear drag, which is simply an approximation of real-life drag, which is nonlinear and additionally dependent on the size, shape, and material of the object. Additionally, many constants are manually-tuned in order to generate plausible-looking behavior. We find that it is quite time-consuming to tune these constants, and values that cause the velocity per tick to exceed the distance between cloth particles will cause the cloth to "explode" or "implode".

Additionally, self-collisions are hard to model accurately with a limited computational budget, and our model runs extremely slow when self-intersection modeling is on.

Submission Details

The contents of this README and `report.pdf` are identical. We provide: 1. an executable demo, accessible using the following instructions. 2. some pre-generated videos in the environment for your viewing convenience in `demos/`.

How to Run

This project can be run the same way as every other WebGL project from this semester. First, run, `make-cloth.py` from the project root directory, and then launch an HTTP server with `http-server dist -c-1`. This will launch an interactive demo. You can use the typical camera controls (`WASD`, arrow keys) to navigate around the environment.

We provide some interesting sliders and toggles. We point to `Wind Strength` in particular, which will generate some interesting interactions. `Toggle Smooth Shading` changes the way the cloth is rendered as described above. `Prevent Cloth Self-Intersections` is also toggle-able, but is an experimental feature due to it being computationally expensive. We also provide settings for tensile strength, drag force, and speed (of animation).

Scenes

We have some different scenes, which can be set using the number keys.

1. Cloth is fixed by the two corners at the top
2. Cloth is fixed by all four corners
3. Cloth is fixed by one corner
4. Cloth has no fixed points and falls to the floor
5. Cloth is horizontal, and has no fixed points. It collides onto a sphere below it. This scene is particularly interesting with wind.
6. Cloth is horizontal and fixed at a center point.

References

https://www.youtube.com/watch?v=aDzMda7cPxI
https://ocw.mit.edu/courses/6-837-computer-graphics-fall-2012/resources/mit6_837f12_assn3/
https://graphics.stanford.edu/~mdfisher/cloth.html
https://github.com/ndesmic/geogl/blob/v3/js/lib/shape-gen.js
https://computergraphics.stackexchange.com/questions/4031/programmatically-generating-vertex-normals

Extra Credit

(10 pts) We have both completed the online course instructor survey.