CS 184: Computer Graphics and Imaging, Spring 2019

Project 4: Cloth Simulator

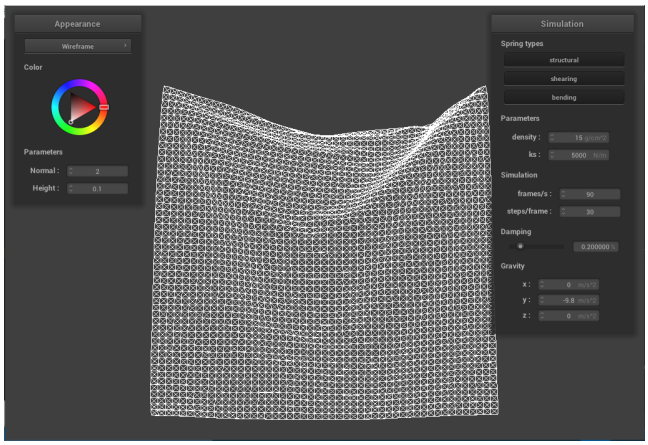
Khadijah Flowers, CS184

Overview

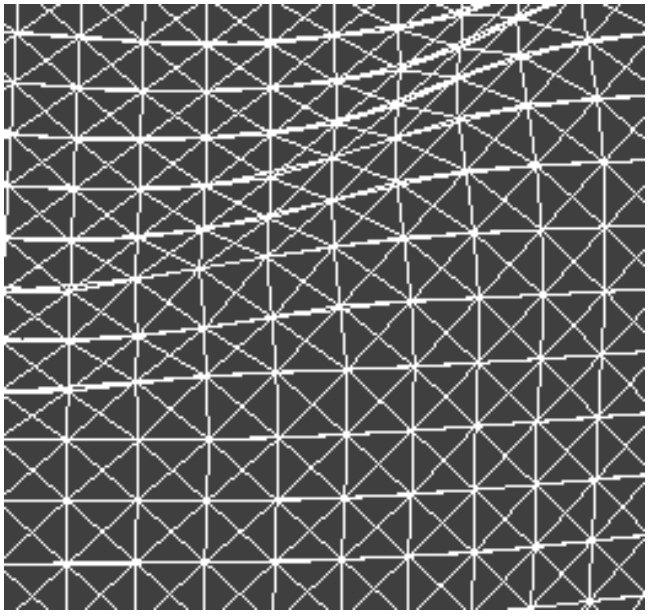
In this project, I implemented a massive point mass and spring system that is capable to interacting with outside objects and itself in many situations inclusing plane, sphere, and self-collisions. By defining structural, bending, and shearing springs that connect point masses in a very strategic way, I was able to create a realistic cloth that reacts to gravitational forces while still maintaining a comfortable resting distance between connected point masses. Overall, the project taught me how to model an object that behaves like a cloth in the real 3D space, and has given me the tools to model other real world, 3D objects and scenarios using point masses and springs to represent them. This has become my favorite part of the class and I will use what I learned here to build my final project. It was really exciting to get to see how we could use computer graphics to simulate 3D objects movements and interactions between the objects and a model. Given forces, point masses, and springs, anything can be modeled!

Part I: Masses and springs

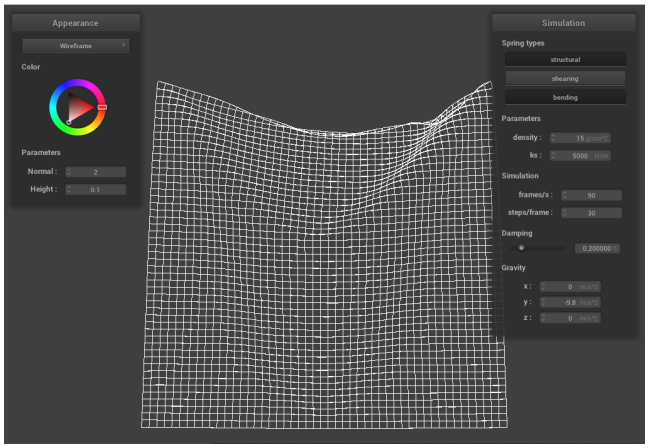
To complete this part of the project, I had to set point masses in starting positions in the scene and then connect them all using bending, shearing, and structural springs to model the realistic movements of a cloth. To keep the particles at a reasonable distance to each other and to have them move with a model in an appropriate way, we need the springs and update their position at the end of each timestep to make sure the point masses in the cloth don't stretch too far.



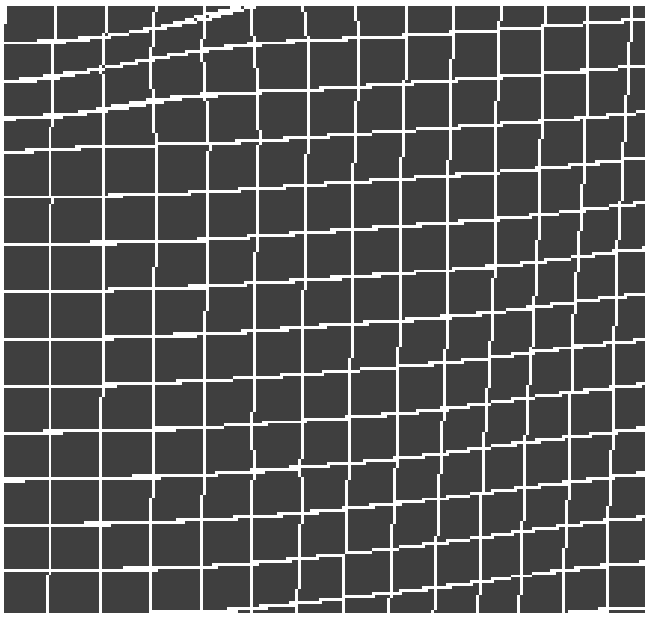
Wireframe w/All Constraints



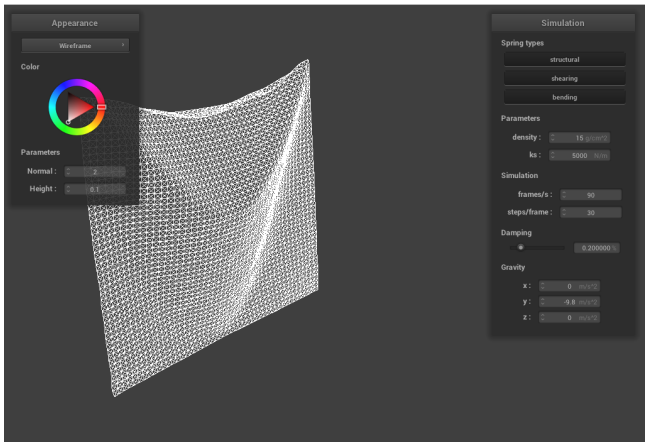
Closeup of Wireframe w/All Constraints



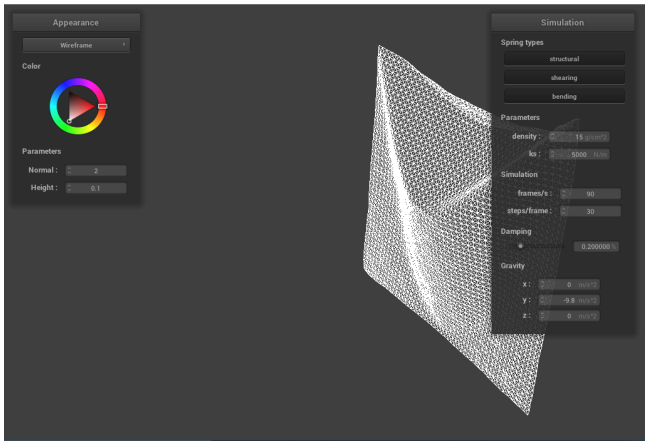
Wireframe w/o Shearing



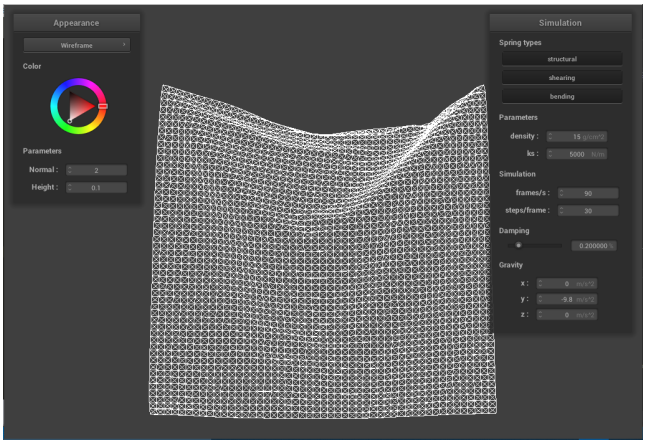
Closeup of Wireframe w/o Shearing



All Constraints Angle 1



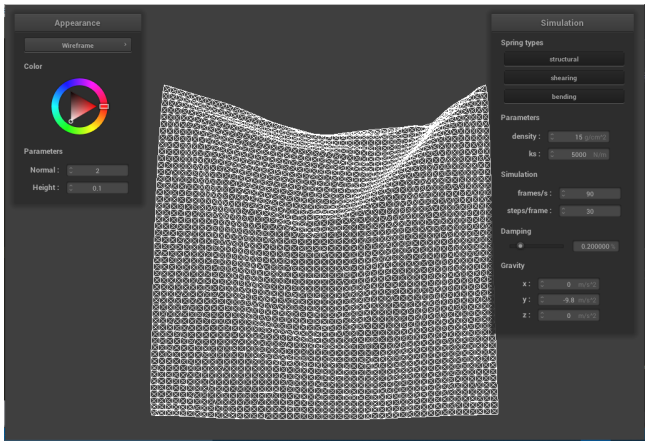
All Constraints Angle 2



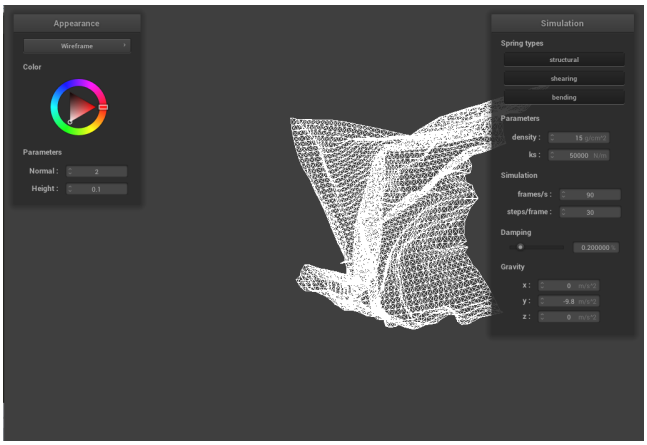
All Constraints Angle 3

Part II: Simulation via numerical integration

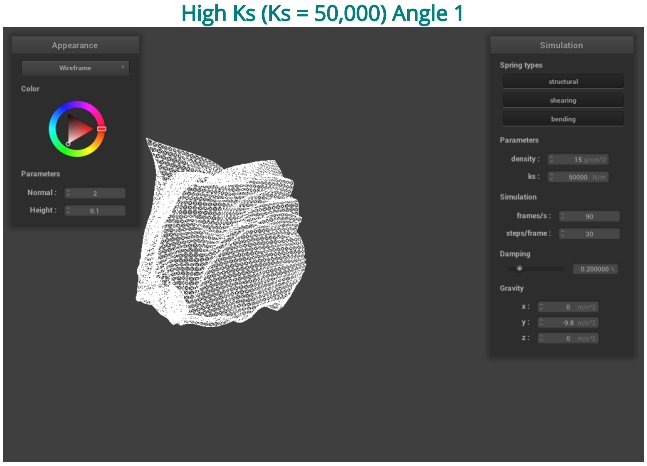
In order to simulate motion in real time, we use Verlet Integration, an algorithm that accurately simulates the motion of a point masses position given the current posiion, the acceleration, and the forces acting on it. This helps us model the cloth's change in position as time moves forward. We update the position, one timestep at a time, and update the distance between point masses as we do so.



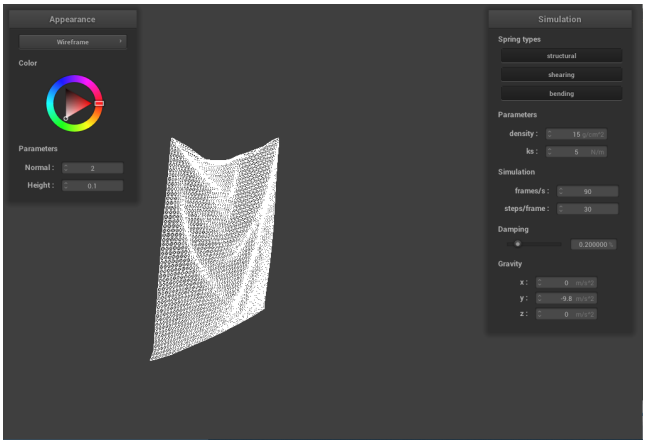
Default Ks (Ks = 5,000)



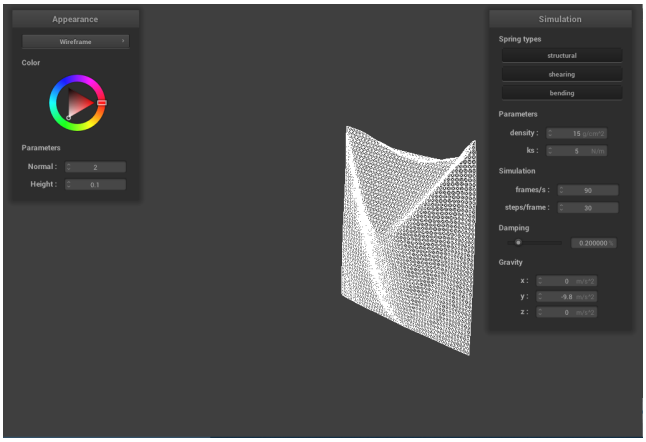
High Ks (Ks = 50,000) Angle 1



High Ks (Ks = 50,000) Angle 2

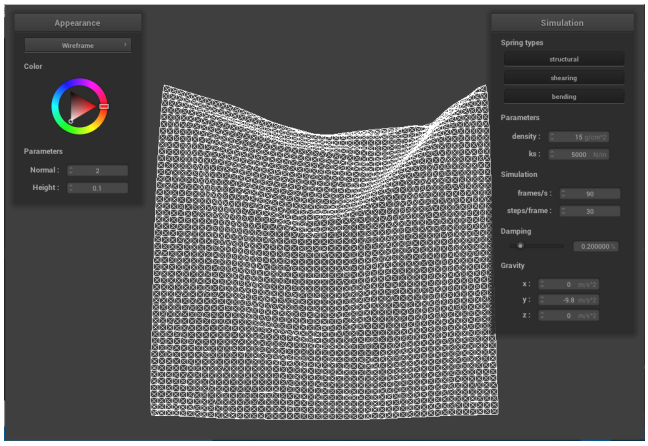


Low Ks (Ks = 5)

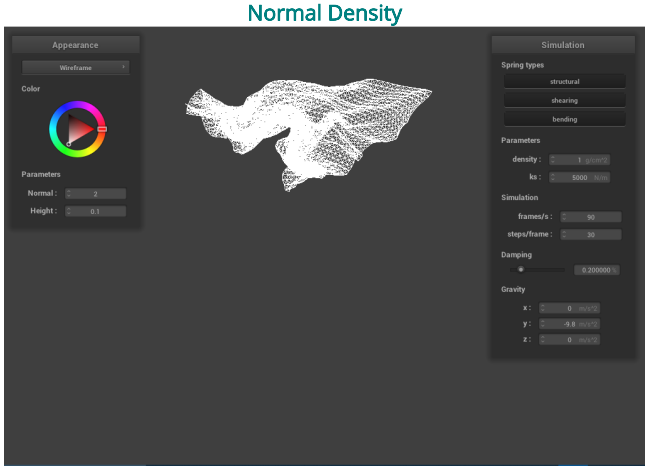


Low Ks (Ks = 5)

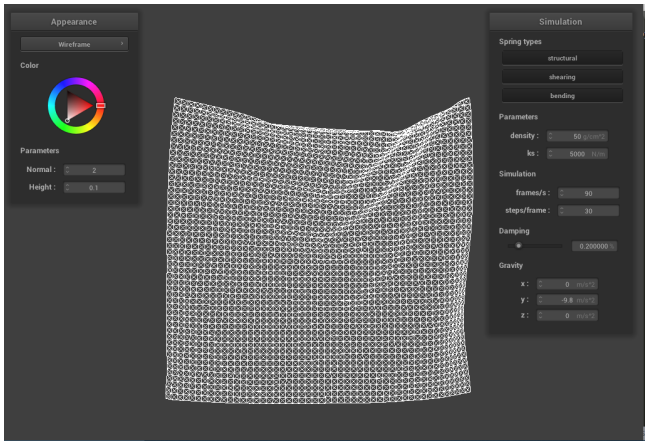
When the spring constant is low, the springs are easier to bend and stretch and seem to be affected more by gravity. When the spring contant is high, the springs and point masses cannot seem to fall and reach their resting position because the springs won't allow for too much deviation. A low spring constant seems as though it is much more susceptible to gravity and the cloth gets to its ending/rest position sooner as opposed to a high spring constant that seems to be at a standstill with forces acting on it. It is affected by the forces, but undoes a lot of the forces to keep the springs and point masses together.



Normal Density

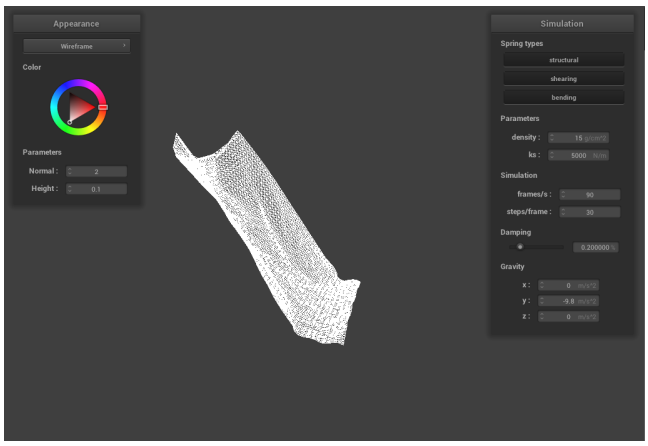


Low Density (density = 1)

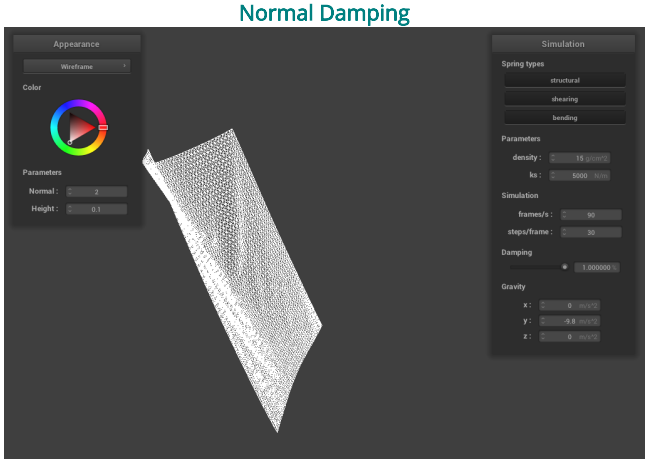


High Density (density = 50)

When the density is high, the cloth falls faster and reaches its resting place sooner. When the density is low, the cloth doesn't reach it's resting place at all. It just continues to hover in the air. The density of each point mass contributes to the overall force applied to it (F = M*a) and the lighter it is, the less obvious it is that a gravitational force is being applied on it. It's like the difference between a piece of paper and a solid object like a brick falling from high in the air.

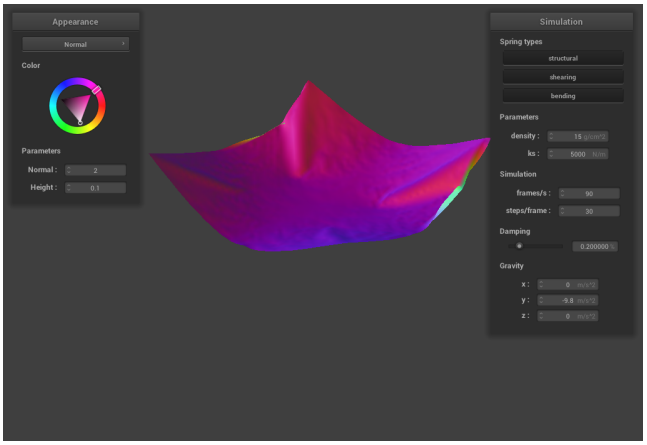


Normal Damping



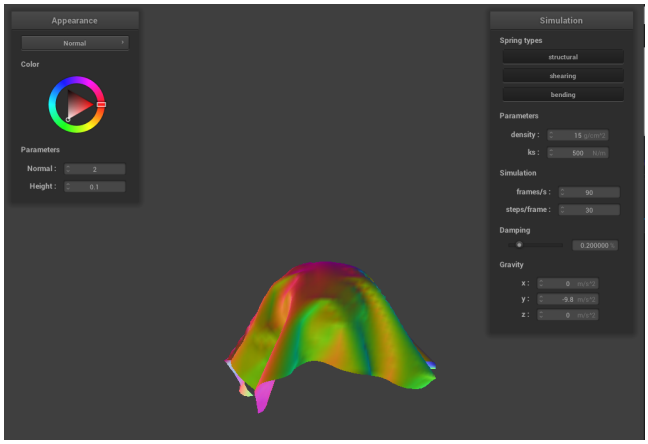
High Damping (1%)

With high damping, the point masses and springs on the cloth seem to be stiff as they fall. There are no curves in the cloth and no folds as it falls. It seems to only be affected by gravity and not the opposing forces underneath it that usually cause the cloth to flutter.

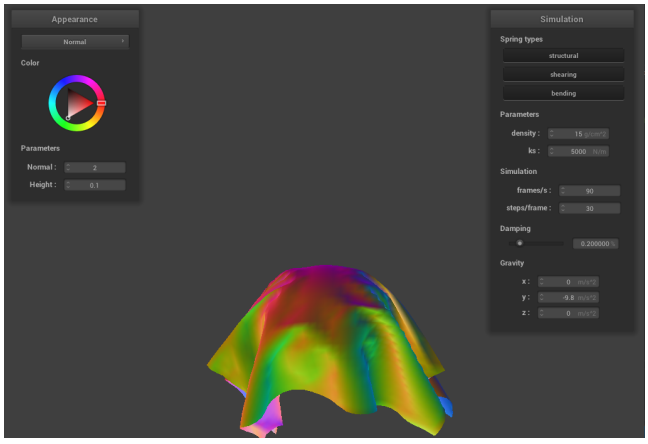


Shaded Cloth

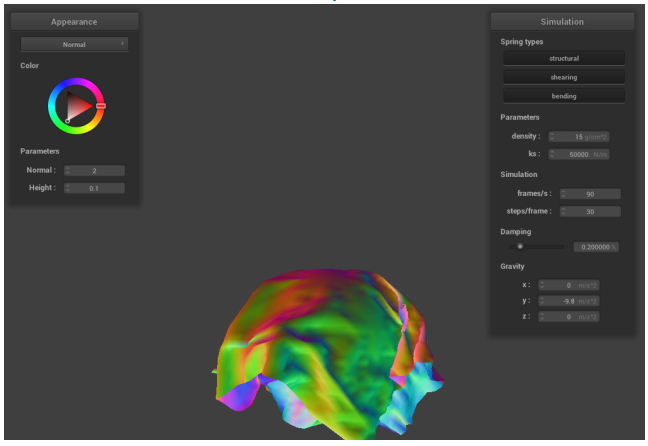
Part III: Handling Collisions With Other Objects



Ks = 500

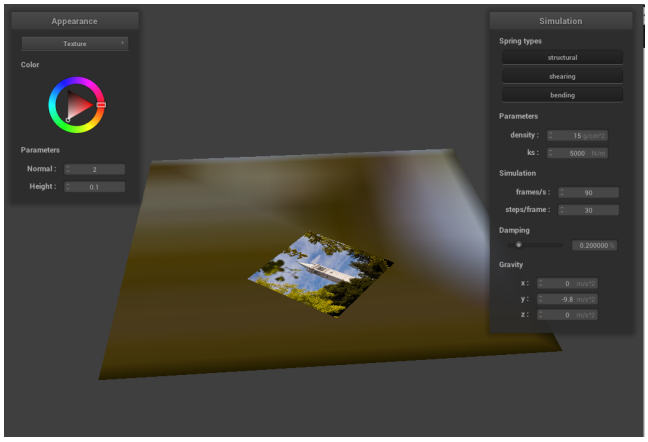


Ks = 5,000



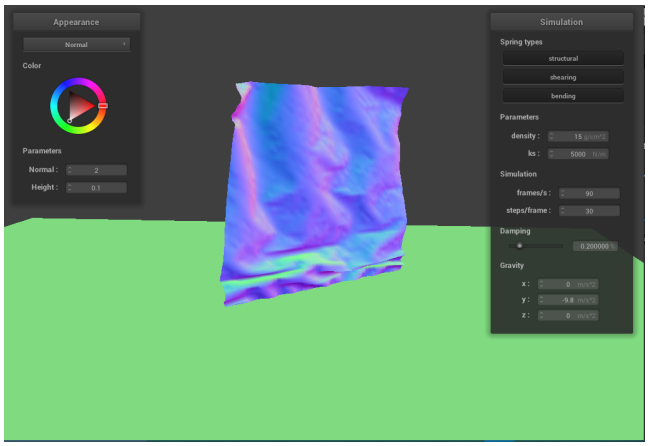
Ks = 50,000

The smallest Ks value causes the cloth to stick more to the geometry of the sphere. The largest Ks value falls onto the sphere, but it doesn't continue to fall and wrap itself around the geometry of the sphere like the smaller value. All of the point masses are being affected by gravity, but once any point gets pinned (i.e. the cloth gets stuck on the ball) we can see that the larger Ks value causes the point masses to seem as if they stop falling all together.

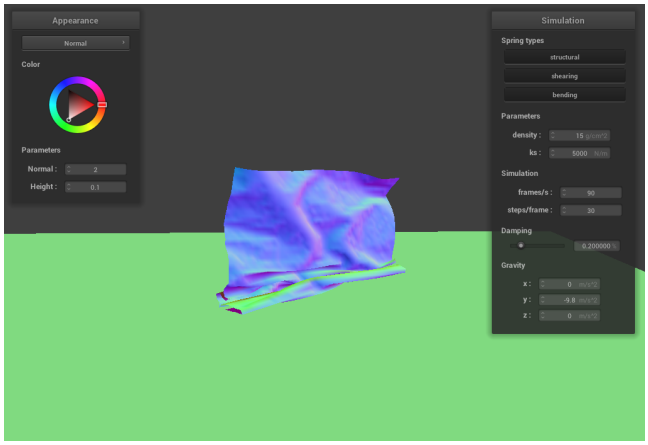


Ks = 500

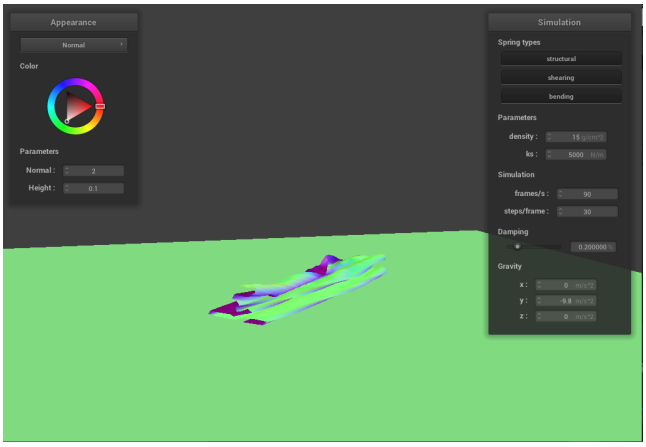
Part IV: Handling Self Collisions



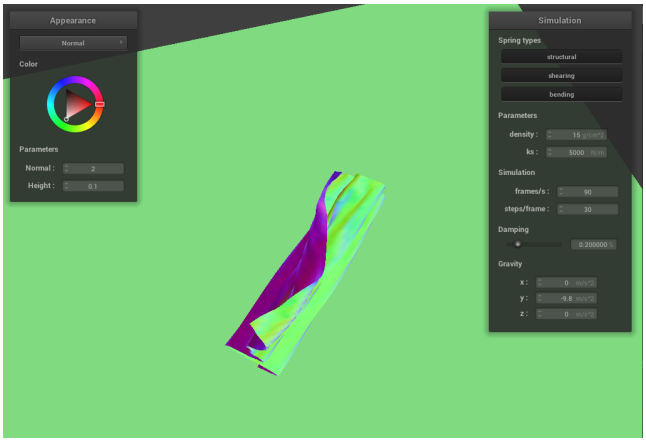
Initial State



Middle State

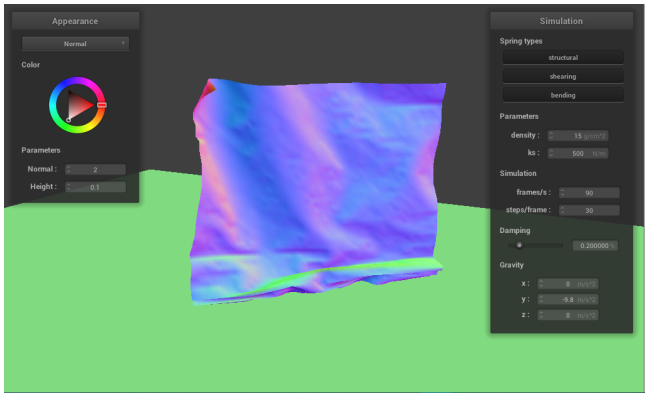


End State #1

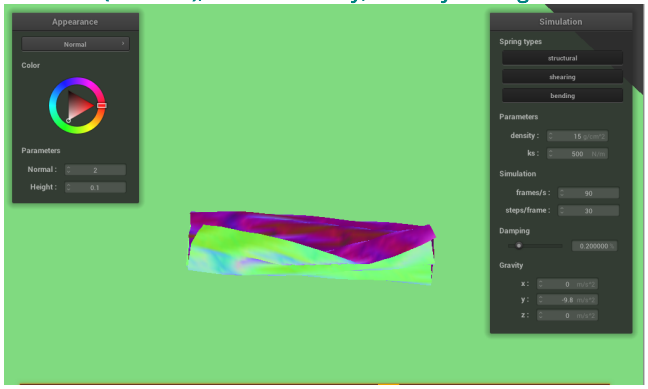


End State #2

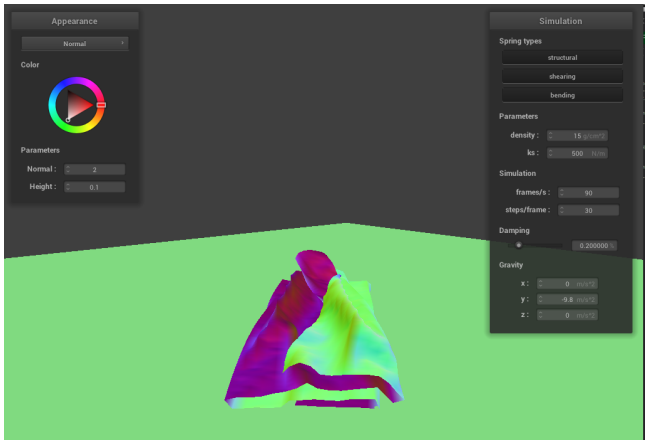
The above is the cloth with the default settings. Below are images of the cloth with varied Ks and Density Values



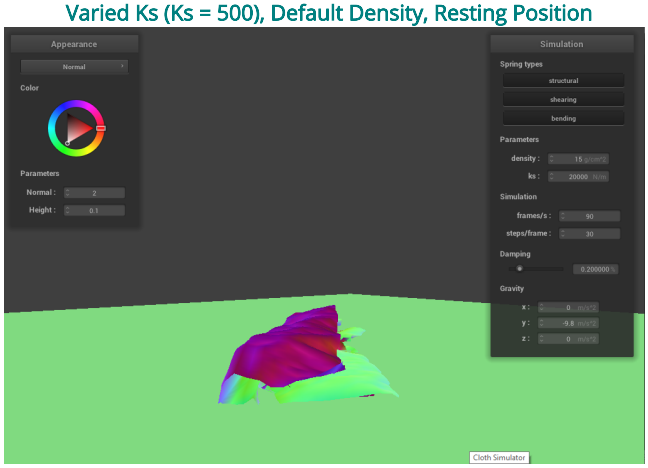
Varied Ks (Ks = 500), Default Density, Halfway Through Simulation



Varied Ks (Ks = 500), Default Density, Resting Position

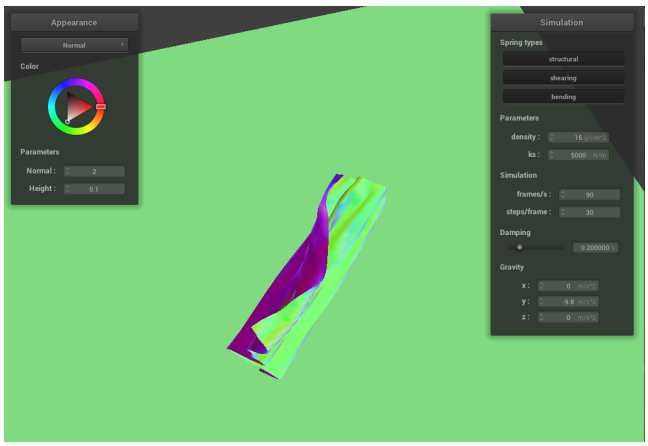


Varied Ks (Ks = 500), Default Density, Resting Position

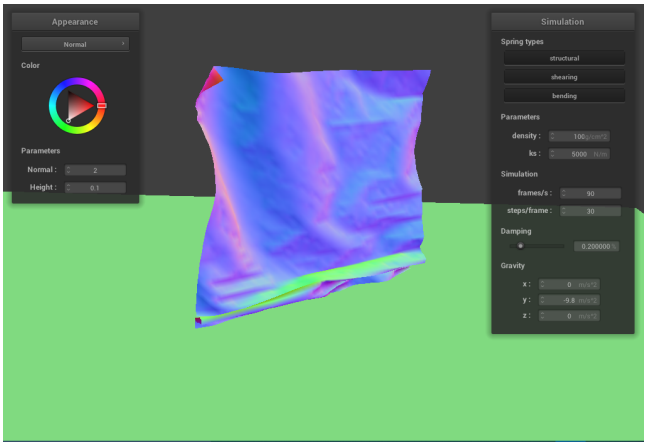


Varied Ks (Ks = 20,000), Default Density, Resting Position

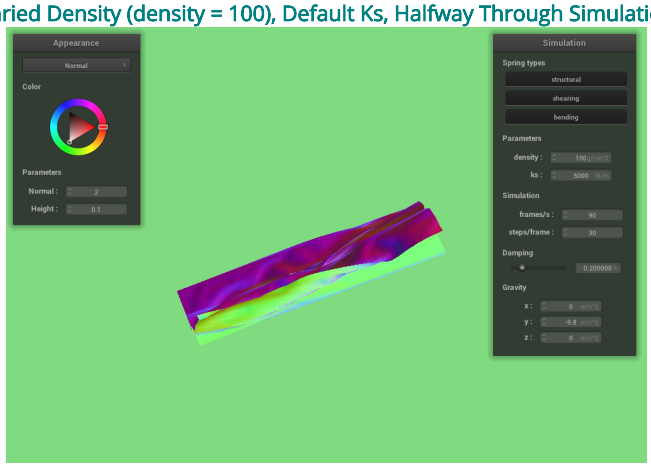
When the Ks value is smaller, the cloth sticks more to the geometry of the plane as opposed to a higher Ks value. I believe this is because the higher Ks doesn't allow as much free deviation of point masses connected by the springs, so there are parts of the cloth that are completely stiff and are not bended as easily or at all and the lack of bending is why the purple side is up and not the green part.



Default Density, Default Ks, Halfway Through Simulation

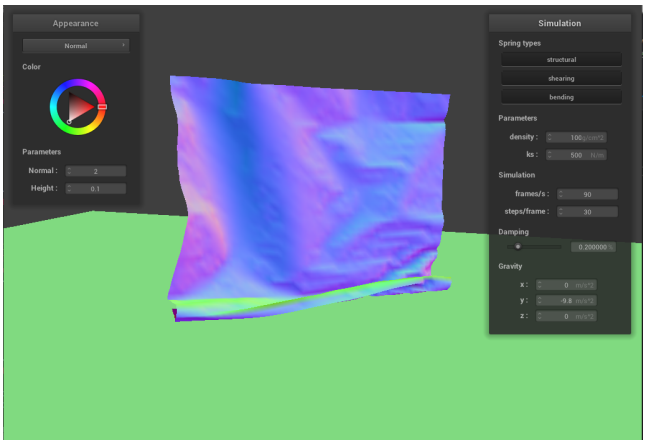


Varied Density (density = 100), Default Ks, Halfway Through Simulation

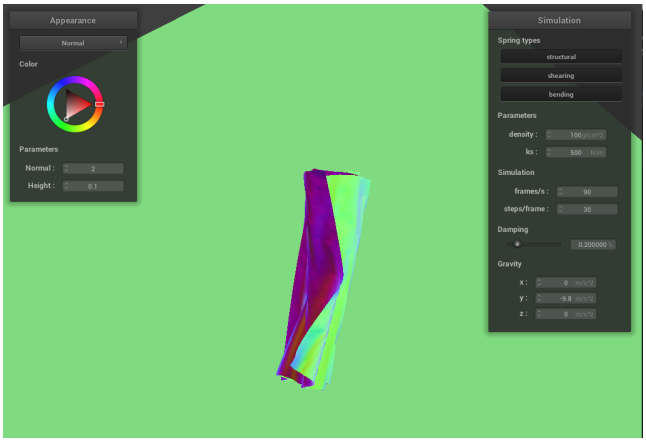


Varied Density (density = 100), Default Ks, Resting Position

As opposed to the default density, a higher density affects the forces applied on the point masses and cause them to fall faster, not allowing the cloth to bend much before it gets to its resting position, which is slightly similar to what happens with a high Ks value. The difference is that a high Ks doesn't allow bending at all while density seems to just not allow enough time to bend.



Varied Ks (Ks = 500), Varied Density (density = 100), Halfway



Varied Ks (Ks = 500), Varied Density (density = 100), Rest

With a low Ks and a high Density, we see almost the same cloth we would have simulated if we had let it stay in the default settings. We don't get that same cloth because the density is acting on the particles and springs and doesn't allow enough time for them to bend to the correct shape before they hit the ground.

Overall, the Ks value seems to be affecting how bendable the point masses and springs are. The higher the spring constant the less bendable it is. Density affects the speed at with the cloth falls because it affects the forces being applied, given the equation F = M*a. Density affects the mass directly and the more mass there is, the more force is being applied to that point mass.

Part V: Shaders

A shader program is a program used to shade an object or model in a scene. These shading affects include lighting, color, and the rendering of these affects. Fragment and vertex shaders work together to shade areas of an object. The vertex shader takes an object and, given the vertices of the object, transforms it from world space to screen space, modeling the geometry of the object. The fragment shader takes these modeled images, now represented as a series of pixels between the vertices that define them, and shades the individual pixels with the lighting and color affects.

The vertex shader sets the scene with the objects, separating their geometry. Then, we make the geometry into pixels that can be displated on the screen. After that, the fragment shader assigns color and lighting affects to produce the image.

Blinn-Phong

Blinn-Phong is a shading technique that approximates the amount of light shining on a particular spot on an object by taking into account the viewing position and the source of light and repeatedly computing what fraction of light the viewer should see based on the angles between the viewer's eyes and the light source.

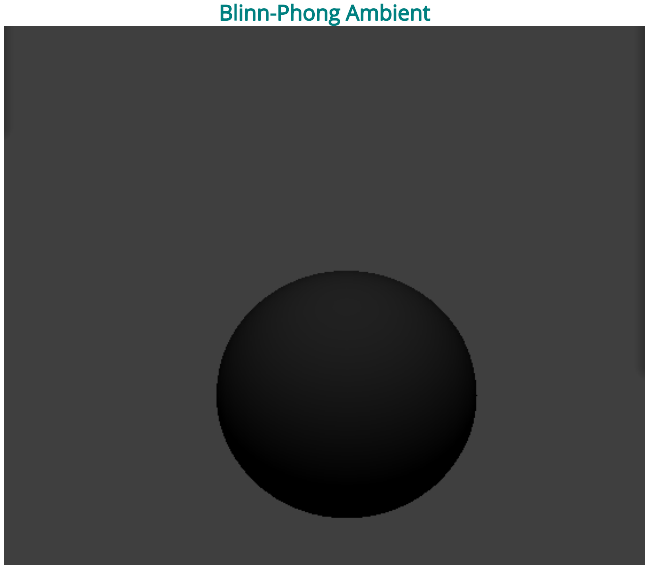
Blinn-Phong uses ambient, diffuse, and specular lighting to assign color and lghting affects to certain areas of an object. In all cases, it takes into account the distanc between the light source and the area of the object.

The diffuse lighting is calculated by trying to determine the angle between the light source and the normal vector relative to a position on the object. If **N** is the normal vector and **L** defines the light position, then we calculate the angle between the two using the dot product. If the angle between them is negative then we set the diffuse term to be **0** because there is no way the light can bounce off the object.

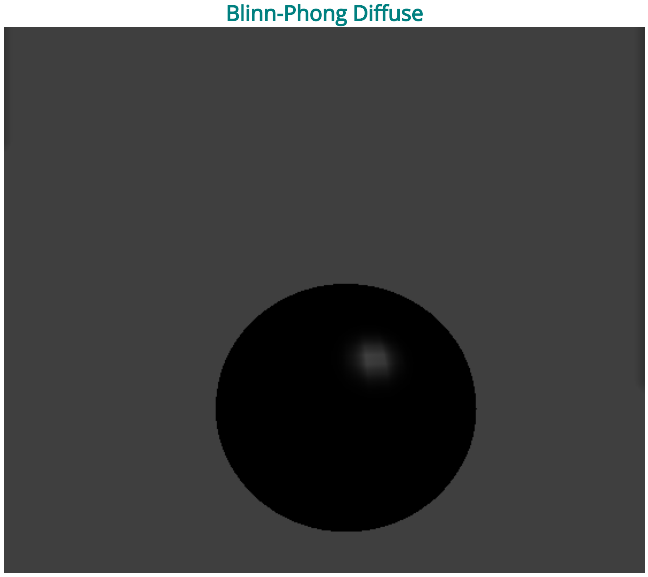
The specular lighing, as opposed to diffuse, computes a half vector that is near the normal by taking the average of the light direction vector and the viewing direction. After that, we calculate the angle between them and take the result to a power **p** to increase the specular reflection of the object relative to the light source.

Finally, the ambient light is not dependent on any lighting.

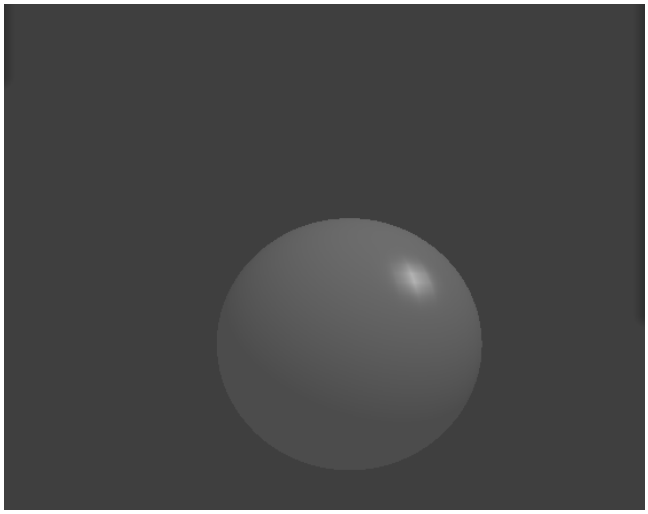
Blinn-Phong Ambient



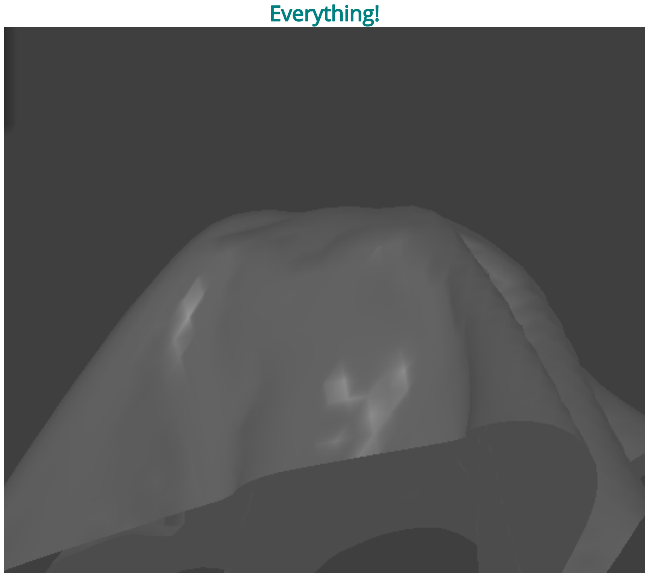
Blinn-Phong Diffuse



Specular

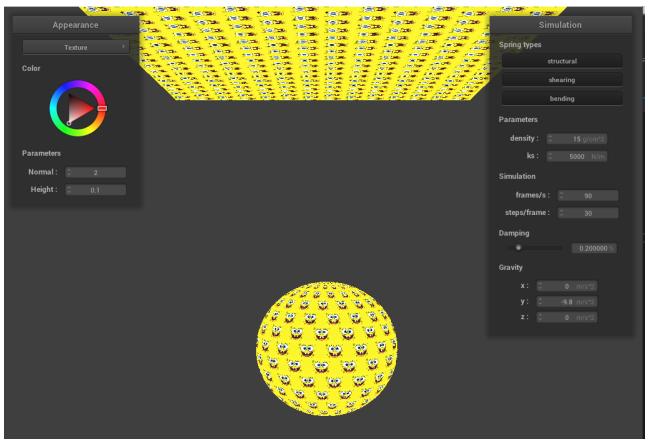


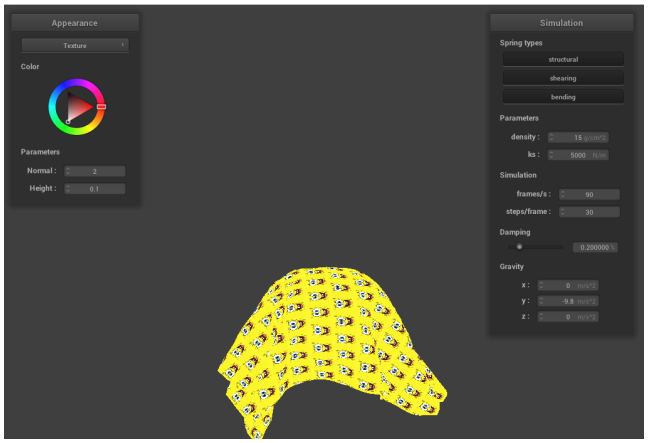
Everything!



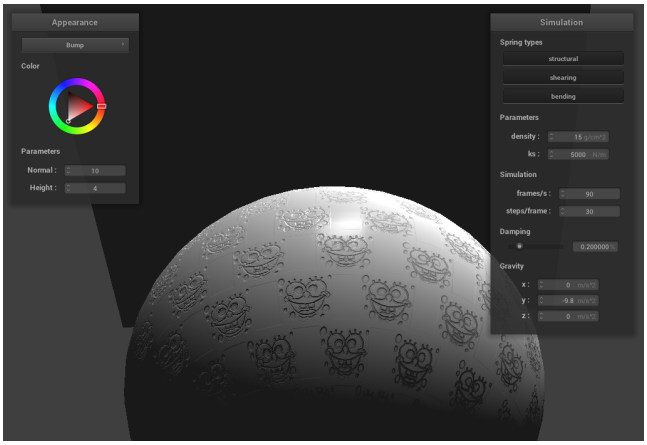
Everything!

Texture Mapping

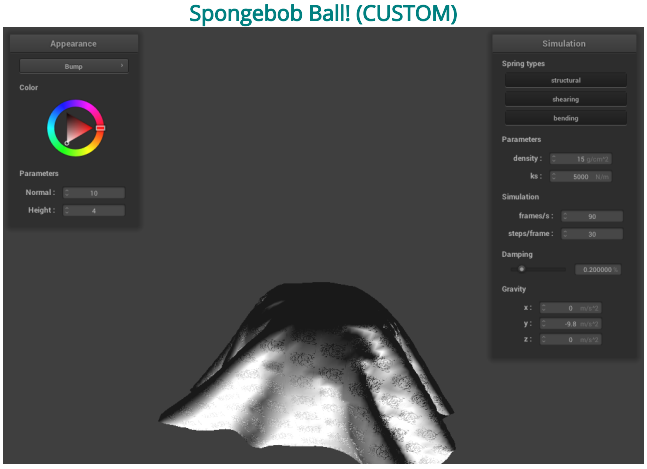




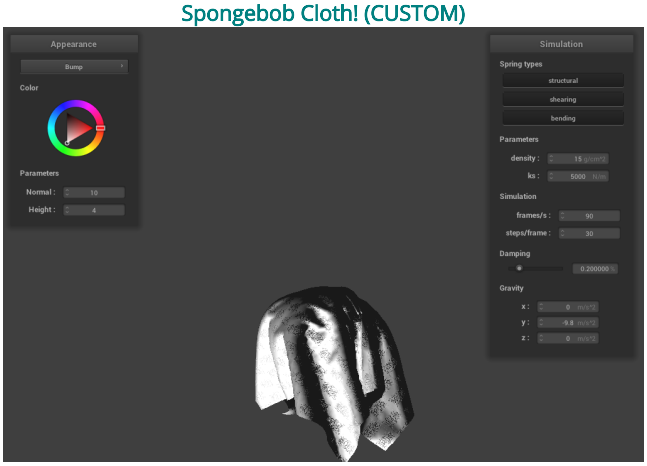
Bump Mapping and Displacement Mapping

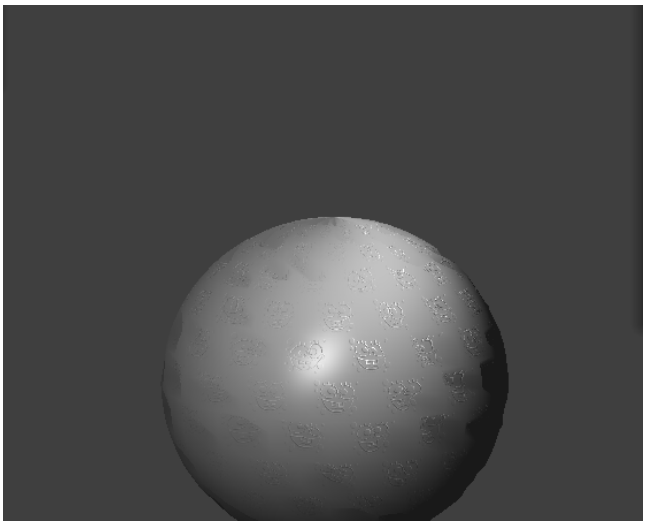


Spongebob Ball! (CUSTOM)



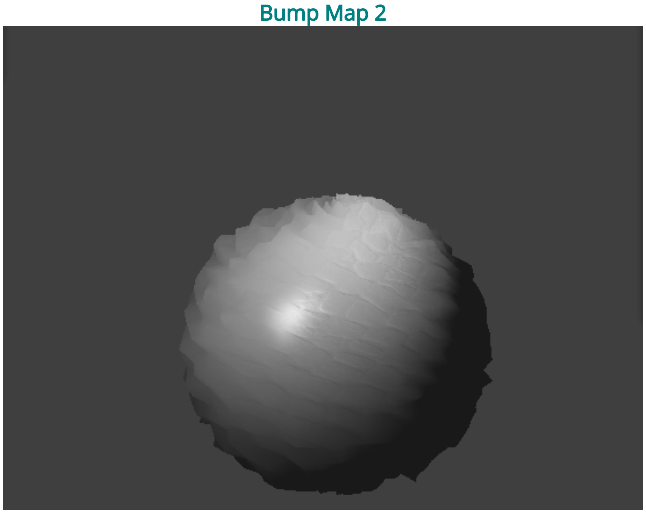
Spongebob Cloth! (CUSTOM)





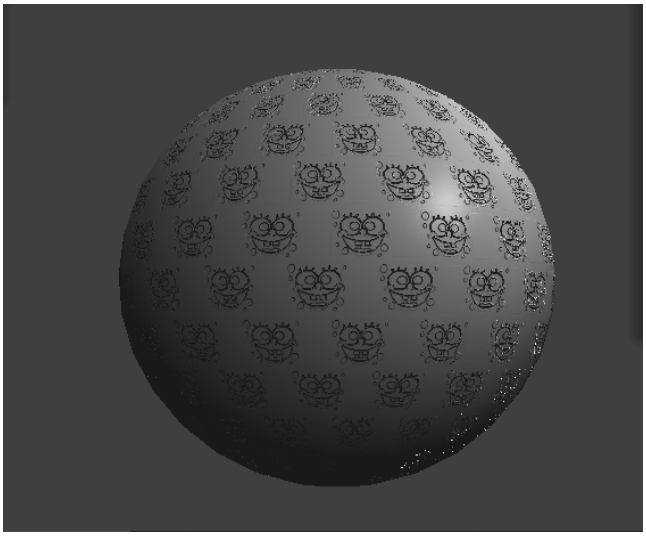
Bump Map 1

Bump Map 2

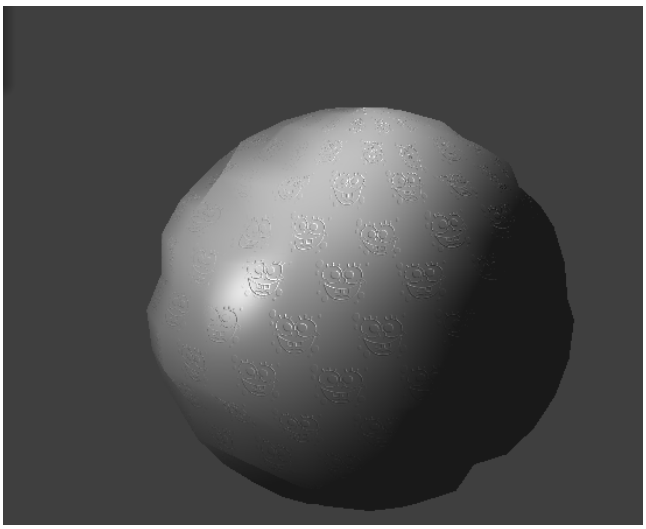


Displacement 1

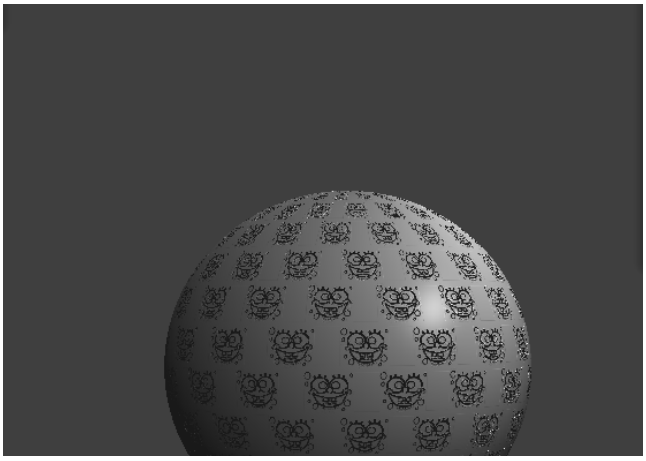
The bump mapping engraves the texture into the ball, but maintains the overall curviness of it. The displacement mapping, however, uses the texture to determine where the curviness of the sphere should deviate and creates bumps on the surface of the sphere.



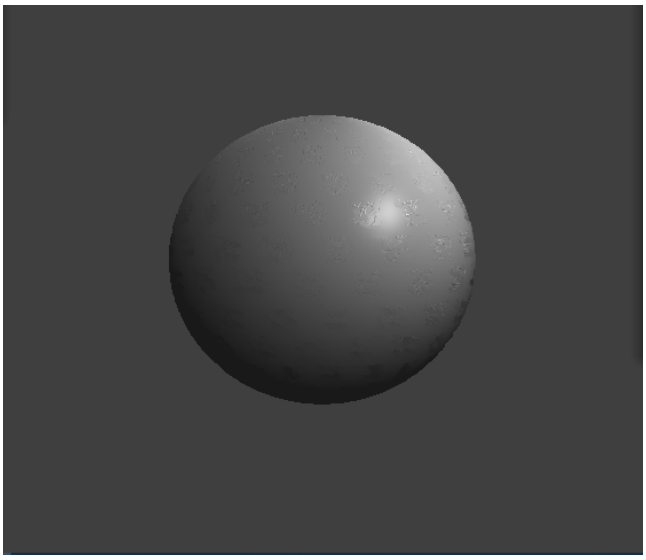
-o 16 -a 16



-o 16 -a 16



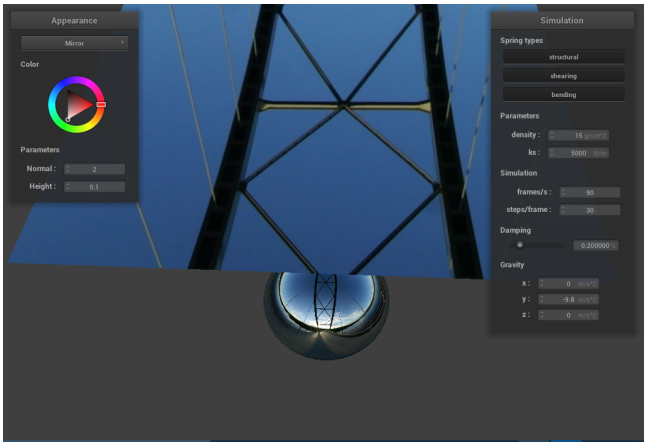
-o 128 -a 128

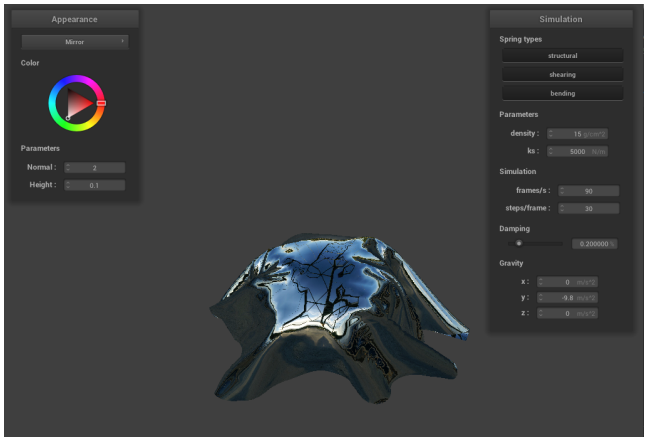


-o 128 -a 128

When we change the coarseness of mesh, we see that the sphere seems smoother. When we look at the difference between the 16 and 128 displacement mesh we see a smoother ball and the dents in the ball are less obvious. There are more dents around the 128 ball, but they are all significantly smaller than the ones in the 16 ball. In the bump ball, the dents in the 128 ball are slightly deeper and the engraved images are more defined.

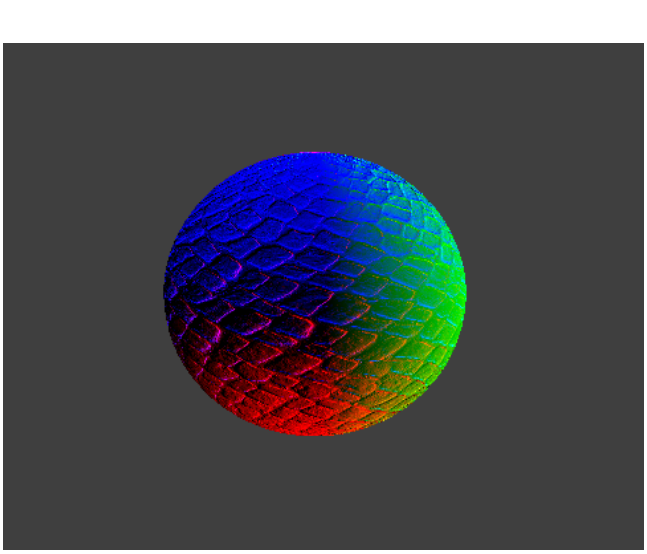
Mirror

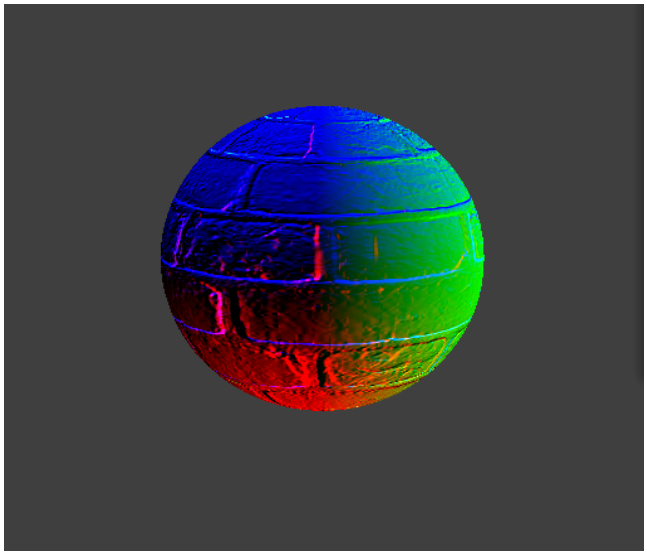


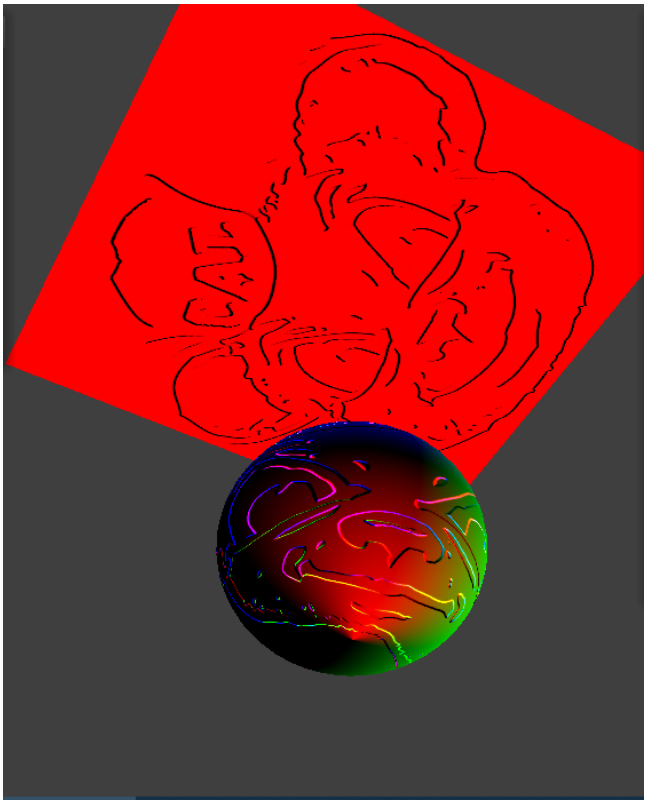


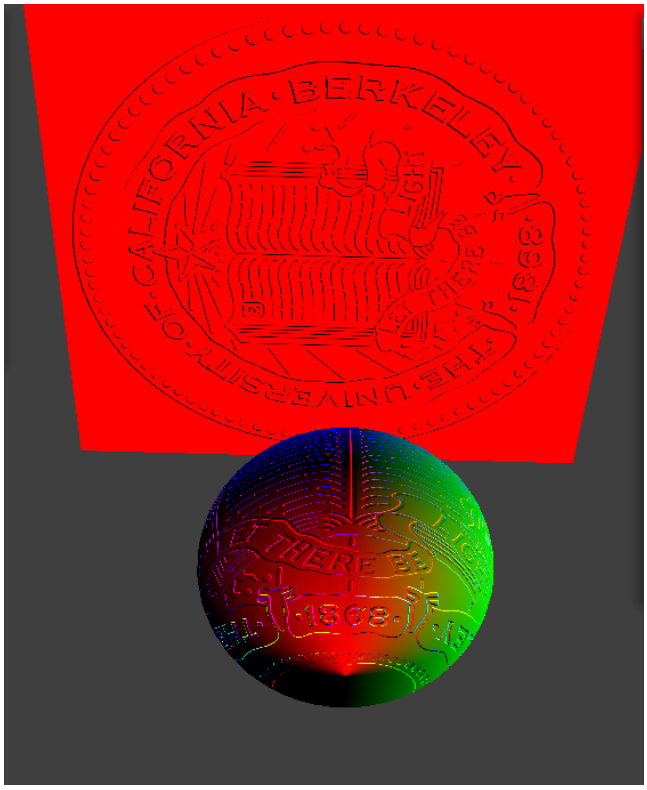
Custom Rainbow Bump Shader

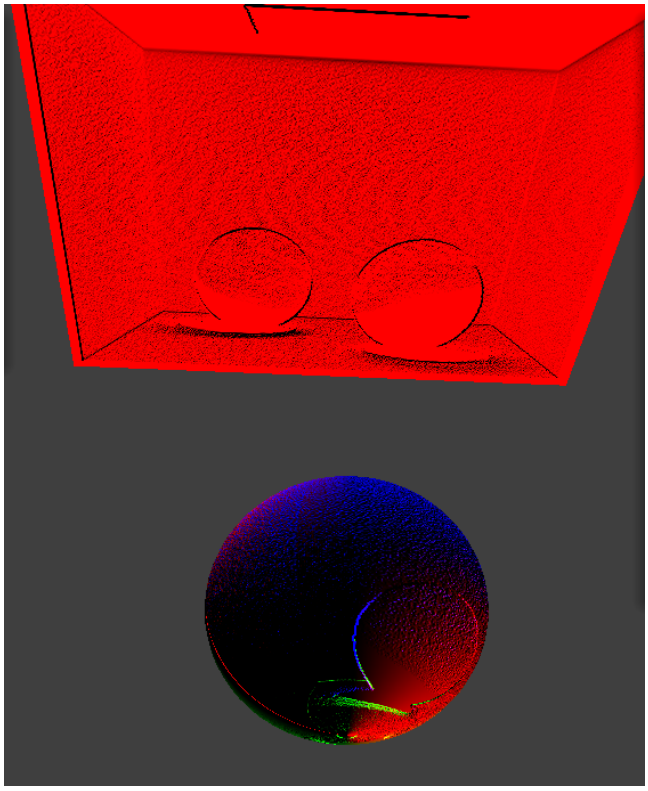
Do I have to say anything! Everything is better when you put a rainbow on it!











I used the bump mapping shader as a template and I colored each area using the color normals. I originally made this mistake when implementing bump mapping when I assigned the out_color to the calcuated displace vector and, as a result, it created this rainbow-like affect. The displace vector is meant to compute a halfvector to compute the amount of light hitting the object in any location.

The shades of the normals change based on the location and the curves and dents in the sphere itself. For example, in the Oski image, the curves in Oski's face are a magenta color, which deviates from the color surrounding it because of the bumps on the surface. We can see this happening in every image where the intensity of the color communicates how much light is hitting the surface at that moment.

With this, I created a Rainbow Bump Shader... On accident!