CS 184: Computer Graphics and Imaging, Spring 2024

Project 4: Cloth Simulator

Satvik Muddana

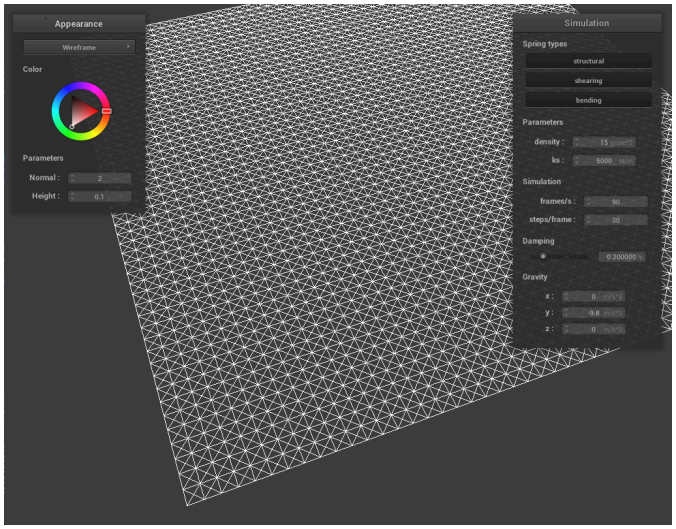
Link to my writeup webpage.

Overview

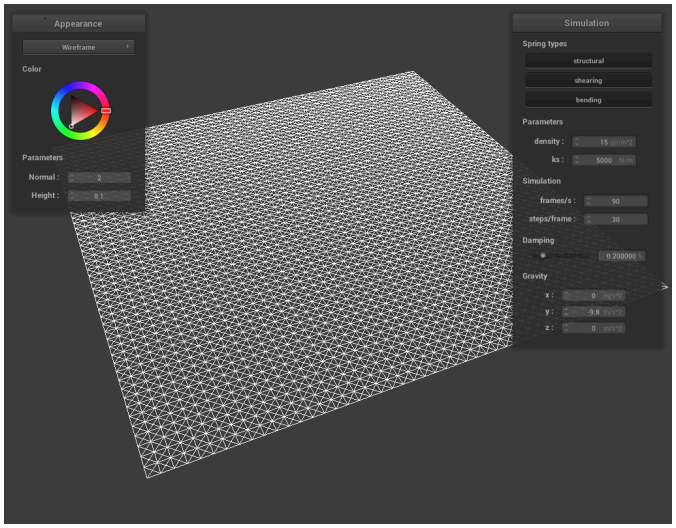
The overview of this project is cloth simulator that I added many features to such as collision handling and shaders to make it as realistic as possible. To try and emulate real life as best as possible, the cloth simulator uses mechanical equations of motion to define how the cloth moves and reacts. This part was challenging because there were many times where I would fail to consider a case or misunderstand a mechanical principle. I created the cloth with a 2D grid of points connected with springs. Thus, motion was driven by spring forces and Verlet integration. Overall, it was great to see a program like this run on my local computer and look so realistic.

Part 1: Masses and Springs

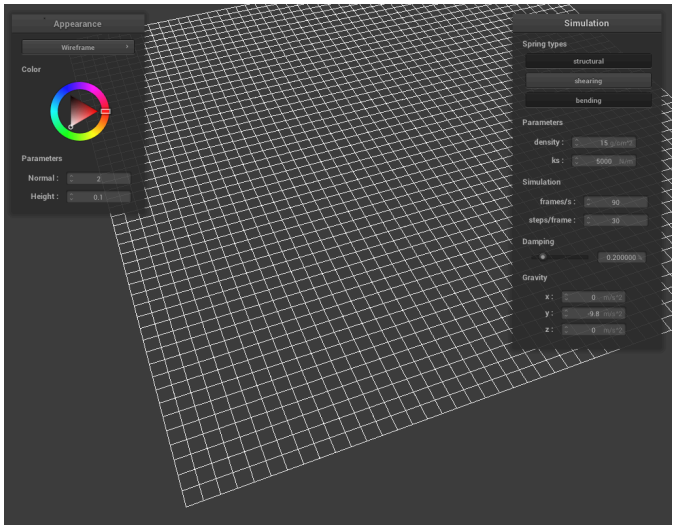
I created the cloth as a 2D grid of point masses where I can set each point mass to either be pinned or not pinned, depending on whether I want it to be stationary or not. All the point masses are connected through a spring lattice structure so that it can follow spring forces (Hooke's Law). The types of springs can be seen below where shear and structural have different rest lengths based on its geometric orientation.



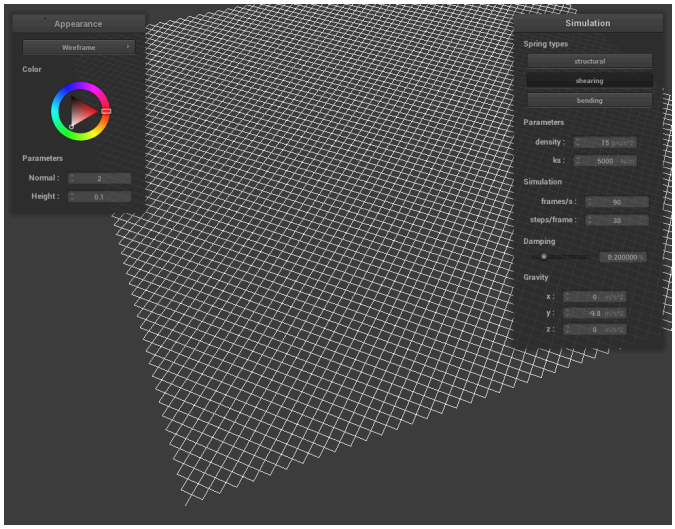
pinned2.json Angle 1



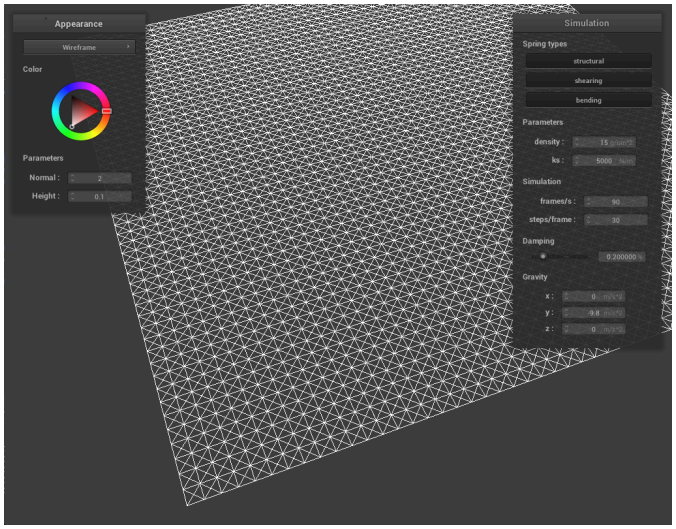
pinned2.json Angle 2



No Shearing Constraints



Only Shearing Constraints



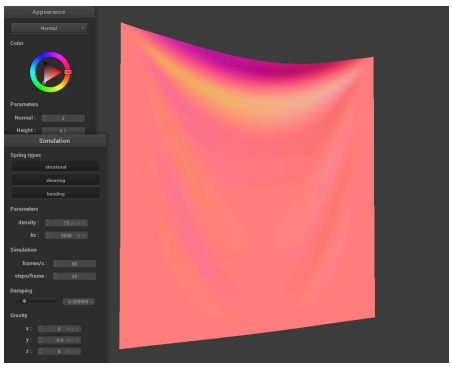
All Constraints

Part 2: Simulation via Numerical Integration

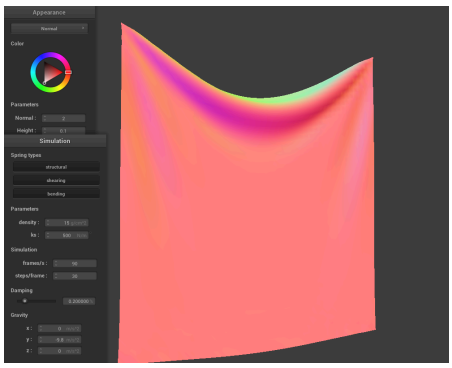
I user Verlet integration to propagate updates to the positions of the point masses in the 2D grid of the cloth. The three parameters that one can tune in the equations of motion are the spring constant, density (mass), and damping. I display the effects of changing these values below. Overall, the process of verlet integration is to run Forward Euler and ensure that springs do not extend past 10% of their rest length.

Changing Spring Constant (ks)

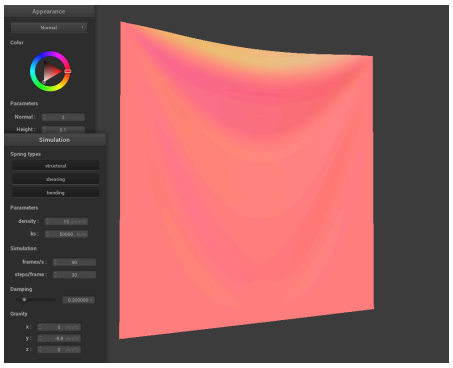
Changing the spring constant seems to alter how stretchy the cloth is. Increasing the constant by a power of 10 made it stiffer as shown in the 50,000 N/m image. Decreasing it made the cloth droop more. Another interesting observation (picture not shown) was that increasing the spring constant too much (70,000+) made it so that the cloth never reached a resting state.



Spring Constant: 5000 N/m (Default)



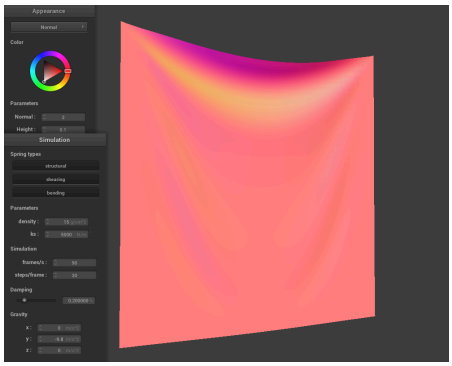
Spring Constant: 500 N/m

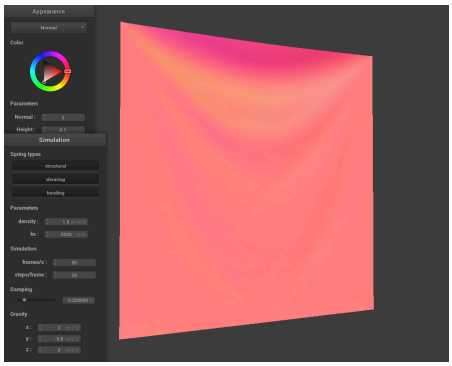


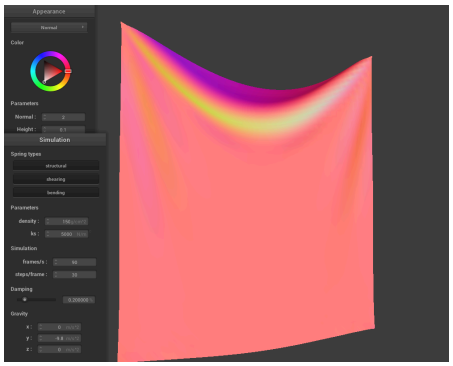
Spring Constant: 50000 N/m

Changing Density

Changing the density seems to make the cloth react more to external forces because an increase in density is also an increase in mass in this environment. The force of gravity can be seen more in the image with a higher density. Thus, the effect can almost be described as the inverse of changing the spring constant.

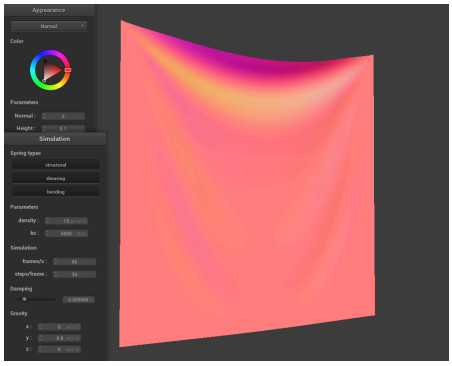

Density: 15 g/cm^3 (Default)

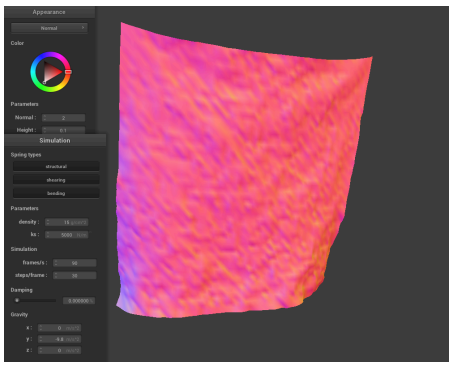

Density: 1.5 g/cm^3

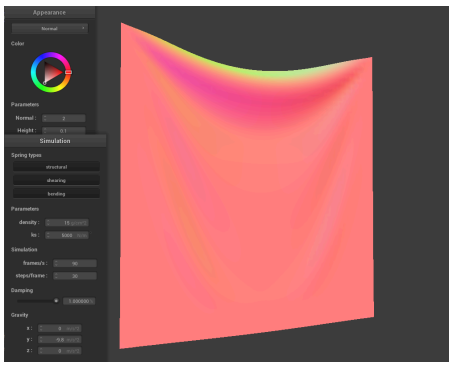

Density: 150 g/cm^3

Changing Damping

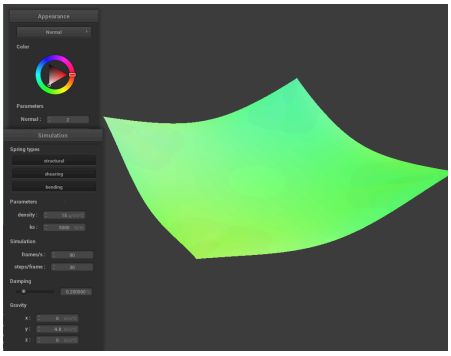
Changing the damping mostly changing how fast it took for the cloth to reach a resting state. At lower values like 0.1 %, the cloth fell very fast. At 0%, the cloth did not even reach a resting state after a decent amount of waiting time. However, at high values like 1%, the cloth fell very slowly and smoothly. It immediately reached the resting state after it fell.


Damping: 0.2% (Default)


Damping: 0%


Damping: 1%

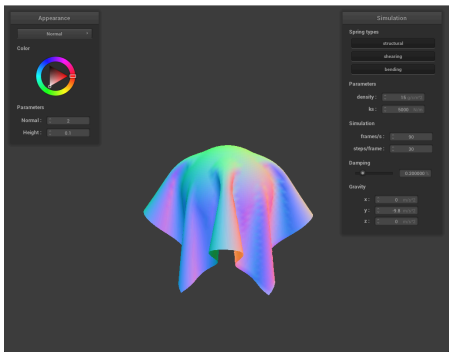
pinned4.json Final Resting State

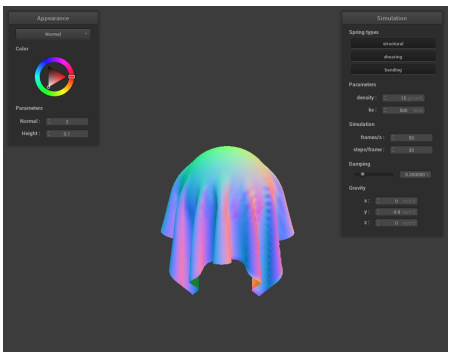

Default Parameters

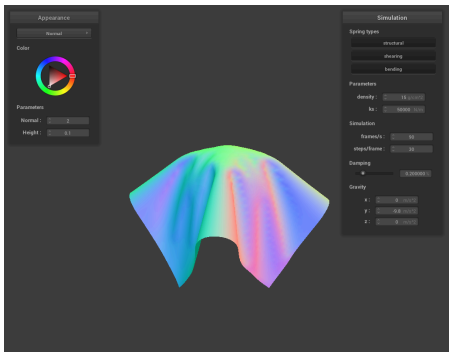
Part 3: Handling Collisions With Other Objects

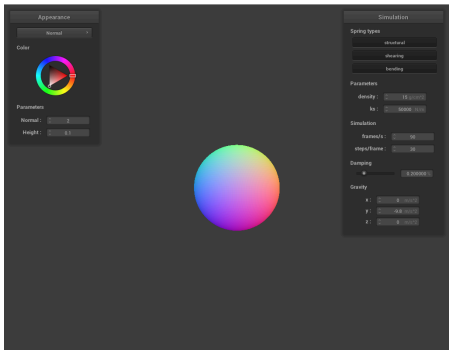
To handle collisions with other objects, I needed to establish where the cloth crossed the surface boundary of the primitives. I simply used the vector definitions to identify this and exert a normal force opposing gravity on the cloth depending on the nature of the contact (top of the sphere vs side of sphere).

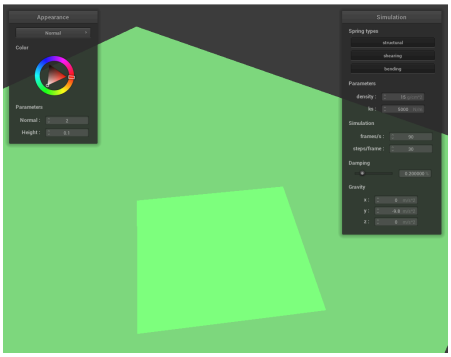
sphere.json Final Resting State


Spring Constant: 5000 N/m (Default)


Spring Constant: 500 N/m


Spring Constant: 50000 N/m


Spring Constant: 50000 N/m after waiting more time (cloth fell off)


Peaceful Rest (plane.json)

Part 4: Handling Self-Collisions

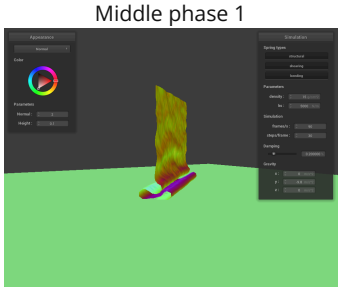
I handled self-collisions by separating the point masses in the 2D grid of the cloth into bins and then doing self-intersection checks only between points in the same bin. This works because a collision is not likely between points that are far away from each other. So binning points that are close to each in distance and only check for collisions between those points can boost performance.

Varying the spring constant created the opposite effect of varying the density. When I decreased spring constant or increased density, the cloth did not bounce as much and simply slipped into the ground, disappearing. The itself did not have any dramatic folds or self-collisions aside from its perturbations and wrinkles. On the other hand, increasing the spring constant or decreasing the density causes more curvature in the folds that are created from the self collisions. The cloth lands of the ground and slowly unfolds overtime.

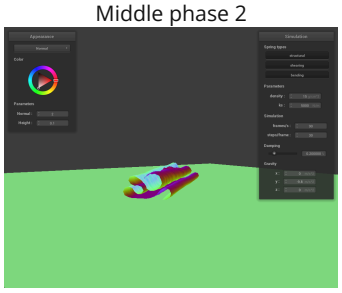
Early self-collision


Default

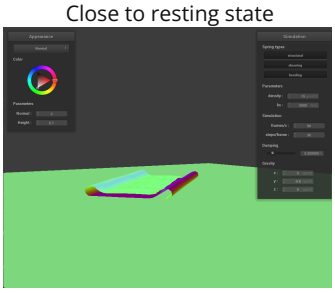
Middle phase 1

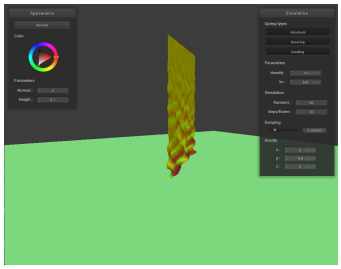

Default

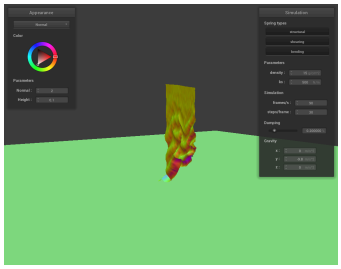
Middle phase 2

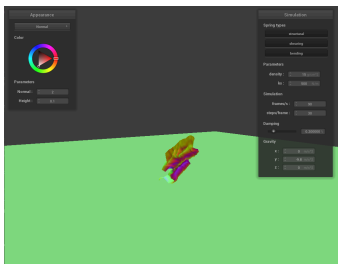

Default

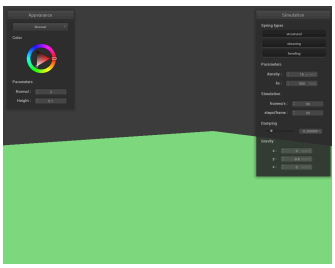
Close to resting state

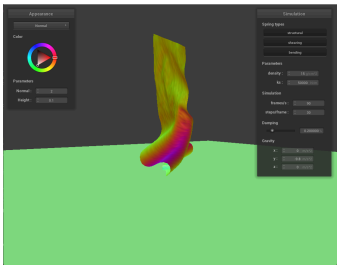

Default

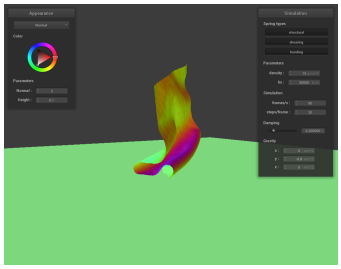

Spring Constant: 500 N/m

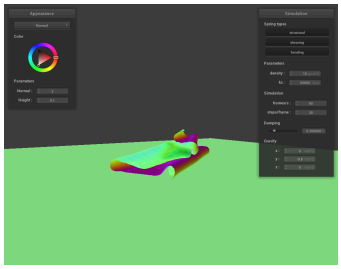

Spring Constant: 500 N/m

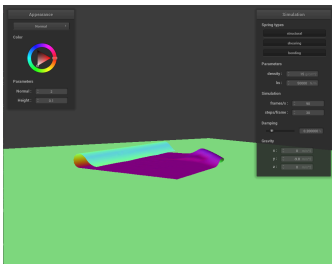

Spring Constant: 500 N/m

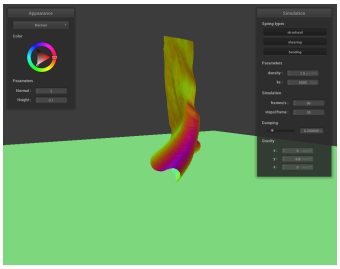

Spring Constant: 500 N/m

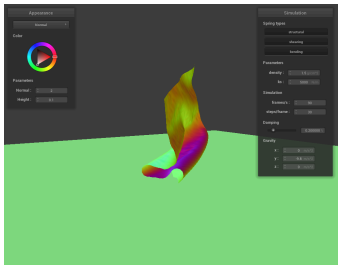

Spring Constant: 50000 N/m

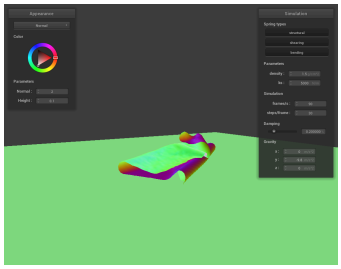

Spring Constant: 50000 N/m

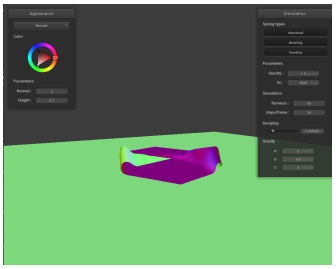

Spring Constant: 50000 N/m

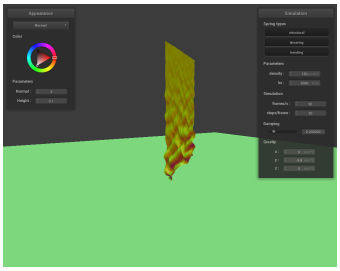

Spring Constant: 50000 N/m

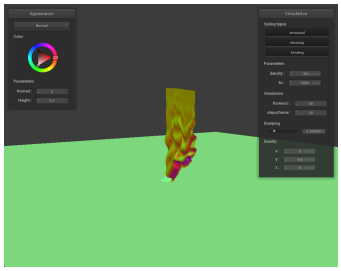

Density: 1.5 g/cm^3

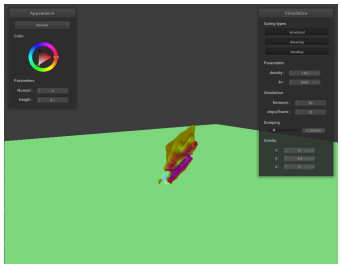

Density: 1.5 g/cm^3

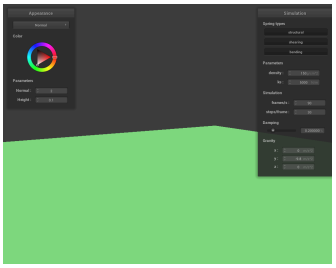

Density: 1.5 g/cm^3


Density: 1.5 g/cm^3


Density: 150 g/cm^3


Density: 150 g/cm^3


Density: 150 g/cm^3


Density: 150 g/cm^3

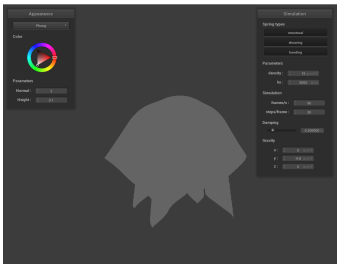
Part 5: Shaders

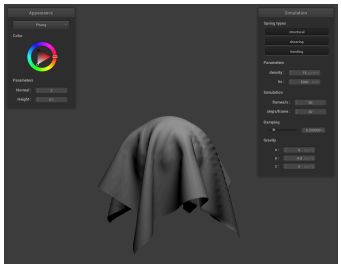
What is a Shader Program?

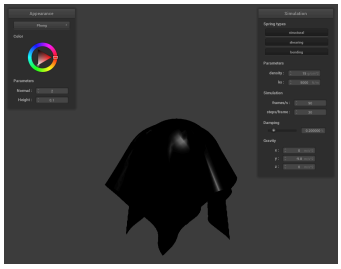
Shader programs are programs that first assign vertex attributes in the scene to convert it to normalized device coordinates and then assign an RGB-alpha vector for each pixel based on vertex data after interpolation. The first part is called a vertex shader, and the second is a fragment shader. The purpose of a shader program is to avoid exhaustive ray tracing when rendering realistic scenes, which can be very expensive. Shaders are an inexpensive alternate option.

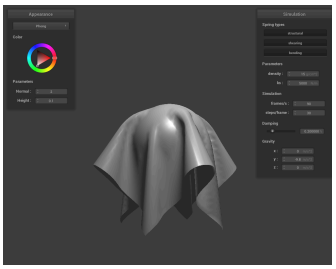
Blinn-Phong Shading Model

The Blinn-Phong Shading Model is a technique combining three different types of light/reflection. Ambient light is intensity reflecting uniformly off objects as we have seen in previous projects. Diffuser light uses Lambert's cosine law to assign higher intensities to the surface normals that point closer to the light source. Specular reflection creates a glare for materials that are glossy by relating the surface normal, viewing position, and light source positon.

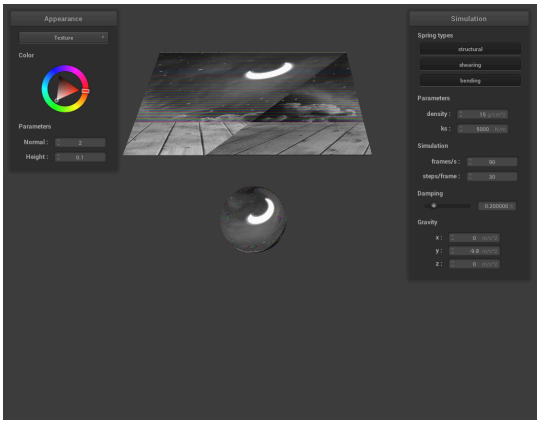

Ambient Component


Diffuse Component

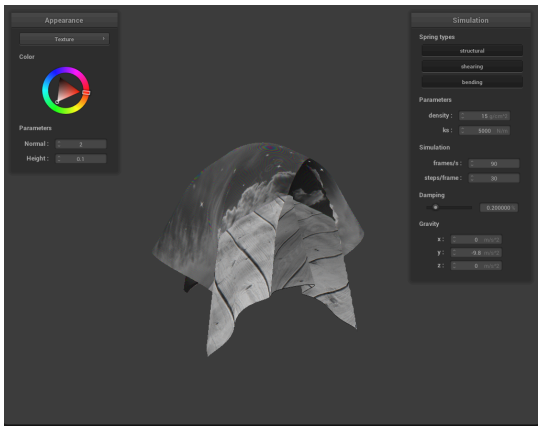

Specular Component


Blinn-Phong: ka=0.2, la=0.8, kd=0.9, ks=0.8, p=50

Custom Texture Mapping Shader



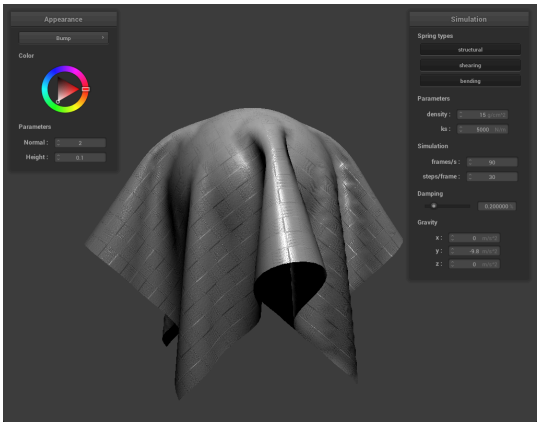
Custom Texture Cloth and Sphere



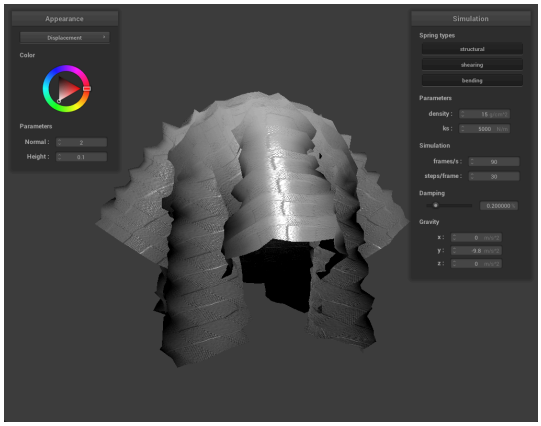
After Simulation

Bump vs. Displacement Mapping Shaders

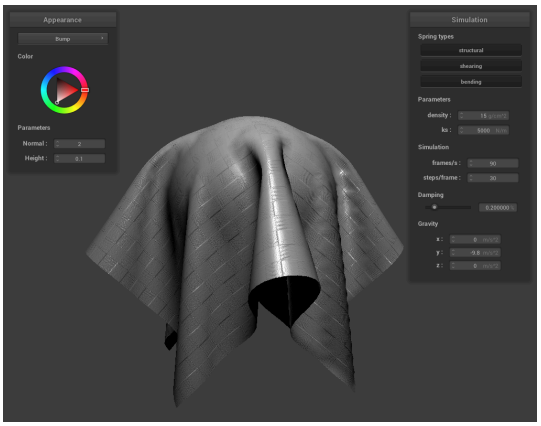
When comparing bump and displacement mapping at a sampling of 16 by 16 vertices (coarseness), it is evident that bump mapping is better. The displacement map is very deformed because surface displacements cannot match the higher frequency of the texture content because of the low sampling rate. Increase the sampling rate to 128 by 128 seems to improve the performance of displacement mapping a little bit. Visually, bump mapping seems to performs similarly regardless of the coarseness. However, it definitely performs better than displacement mapping at low sample rates/coarseness.



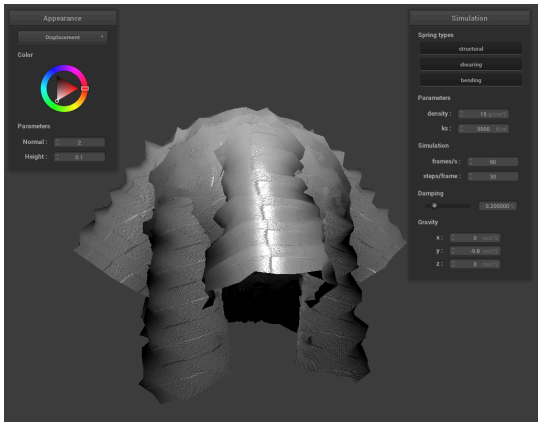
Bump Mapping Shader, Coarseness: 16x16



Displacement Mapping Shader, Coarseness: 16x16

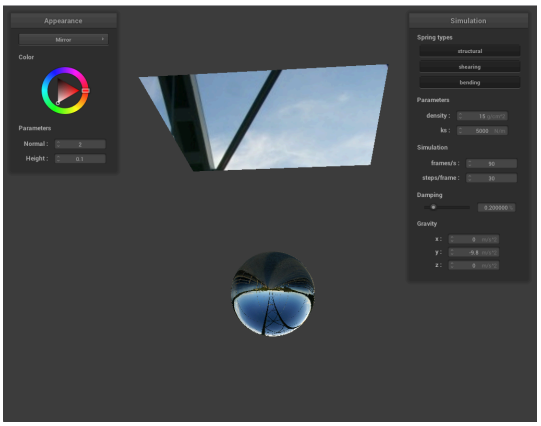


Bump Mapping Shader, Coarseness: 128x128

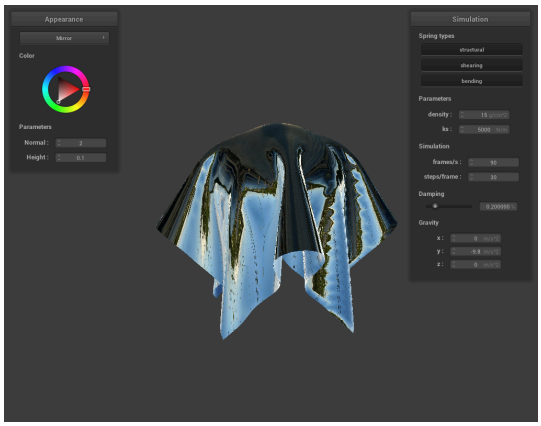


Displacement Mapping Shader, Coarseness: 128x128

Mirror Shader



Mirror Cloth and Sphere



After Simulation