CS 184: Computer Graphics, Spring 2024

Homework 4: Clothsim

Ernest Lu

website link

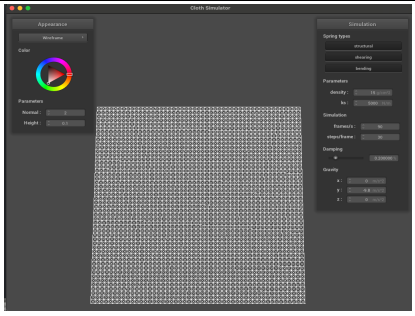
Overview

In this homework, we implement a cloth simulator that uses mass-spring systems to simulate the behavior of a cloth. Point masses spaced around the cloth are connected by springs and modeled by the forces acting on them. Spring forces and gravity control how the point mass moves, and the cloth is rendered as a collection of these mass-springs. Finally, we implement shading on the cloth, where the light source and shading method specified control how the cloth looks in light.

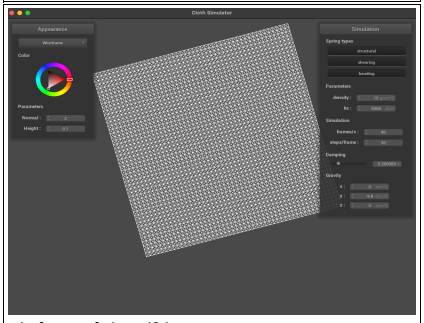
Part 1: Masses and Springs

We implement Cloth::buildGrid by taking the cloth's parameters and populating the cloth's `point_masses` and `springs` vectors. Specifically, we first set all the pinned point masses to be pinned depending on whether or not they were in the input. Then, we space point masses uniformly around the width and height of the cloth. This is supplemeted by an `is_pinned` array we maintain to see whether or not a point is pinned. Next, we populate the `springs` vector with structural, bending, and shearing constraints. For structural constraints, we connect point masses at (w, h) to $(w-1, h)$ and $(w, h-1)$ if possible. For shearing constraints, we connect point masses at (w, h) to $(w-1, h-1)$ and $(w+1, h-1)$ if possible. For bending constraints, we connect point masses at (w, h) to $(w-2, h)$ and $(w, h-2)$ if possible.

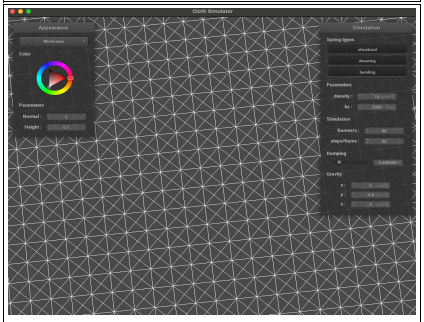
images of the wireframe



wireframe of pinned2.json

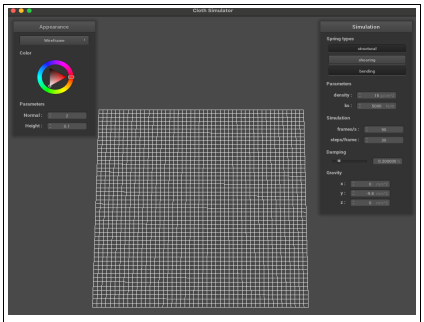


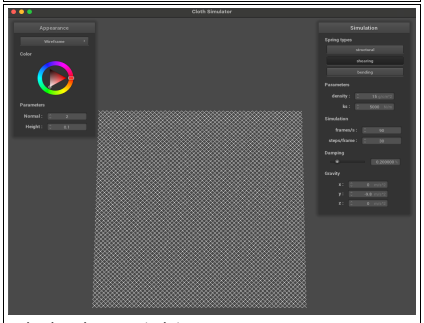
wireframe of pinned2.json

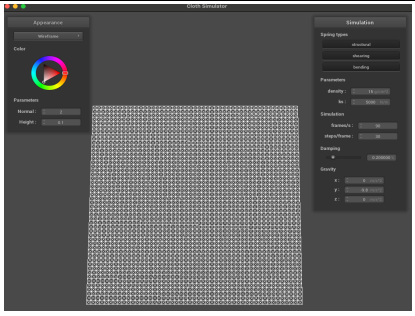


close up wireframe of pinned2.json

images of different sets of constraints

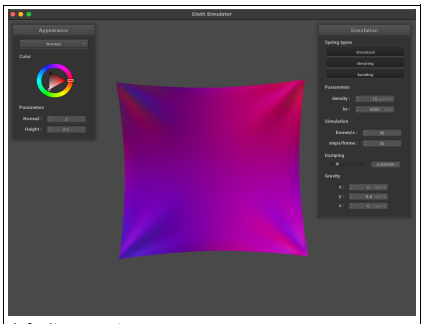

no shearing constraints


only shearing constraints


every constraint

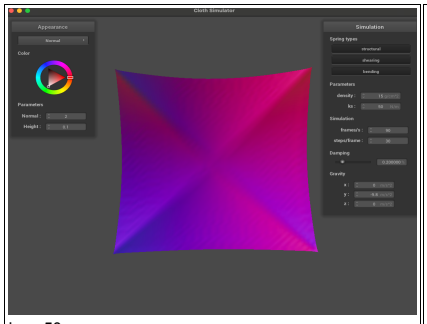
Part 2: Simulation via Numerical Integration

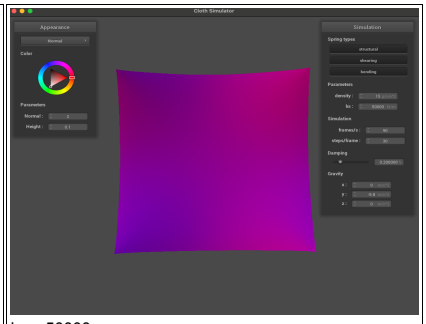
We implement simulation of forces acting on the cloth, below is an image of `pinned4.json` after a simulation with default parameters:


default parameters

Varying the k_s value:

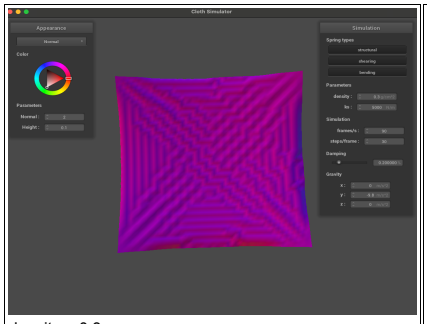
I noticed that as I decresed the k_s value, the cloth became more elastic and less stiff. As a result, the fabric is a bit looser and wrinkles seem to increase from this. Specifically, at the midpoint of the cloth where gravity has the most influence, we see more wrinkles and an "X" structure at the point with low spring k_s.

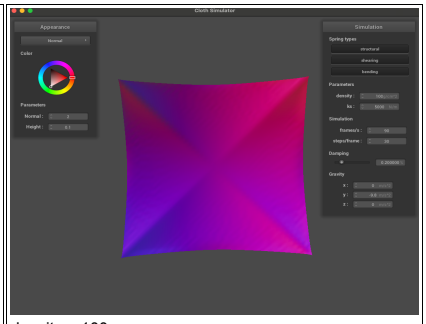

ks = 50


ks = 50000

Varying the density value:

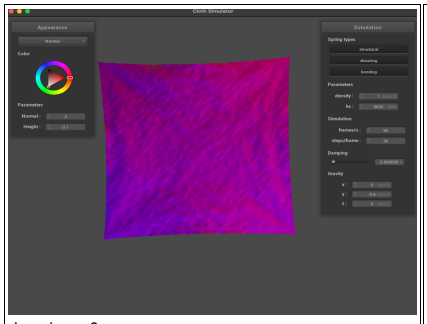
At lower densities, the cloth makes more folds because there is less force applied to the springs. In contrast, at higher densities, the cloth is more taut and less wrinkled. This is because the springs are more compressed and have more force applied to them. As a result, the cloth is more taut and less wrinkled. I also noticed that the cloth is more stable at higher densities, and less stable at lower densities. This is because the springs are more compressed and have more force applied to them. As a result, the cloth is more stable and less stable.

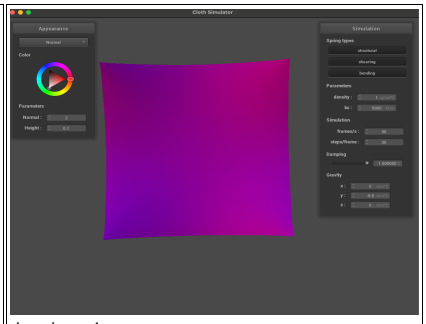

density = 0.3


density = 100

Varying the damping value:

At low damping values, the cloth is constantly moving. This is because the damping value is low, and the springs are not losing energy as quickly. As a result, the cloth is more elastic and moves more. At high damping values, the cloth is more stable and less elastic. This is because the damping value is high, and the springs are losing energy at a slower rate. As a result, the cloth is less elastic and moves less.


damping = 0

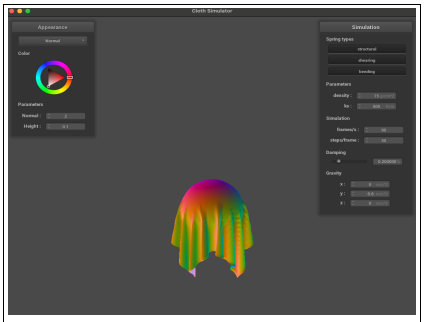

damping = 1

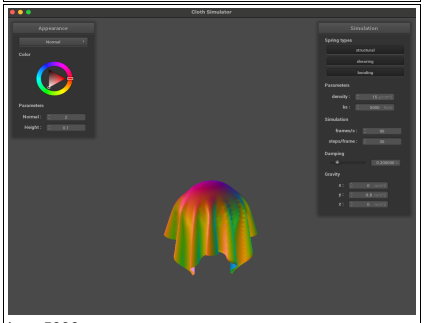
Part 3: Handling Collisions with Other Objects

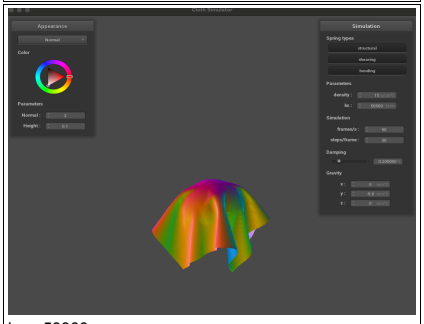
We implemented collisions with other objects by implement the `collide` functions in the `Sphere` and `Plane` classes. For the sphere case, we check if the point mass is inside the sphere, and if so, we "bump" it out by applying a correction force scaled down by friction. This force is added to the total forces acting on the point mass. For planes, since they line on a 2-dimensional plane, we instead check whether or not the point mass would have intersected with the point mass depending on whether or not its previous position and current position are on opposite sides of the plane. In this case, we apply a similar correction vector adjusted by friction to move the position back to the proper side.

At different k_s values, the spring becomes harder to extend/compress and so the fabric is tighter resting on top of the sphere for these k_s values. Specifically, at $k_s = 50000$, we notice a significantly stiffer cloth resting on top of the sphere.

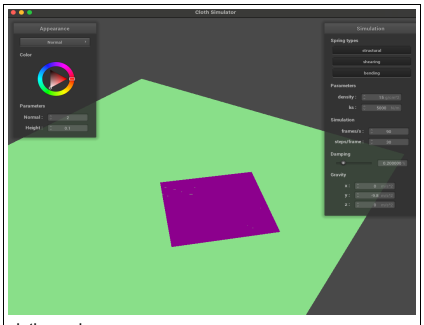
Sphere collisions

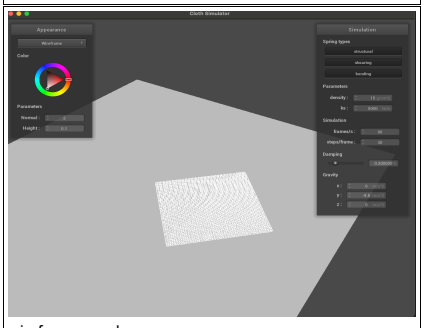

ks = 500


ks = 5000


ks = 50000

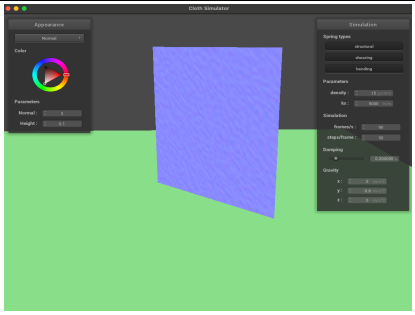
Plane collisions

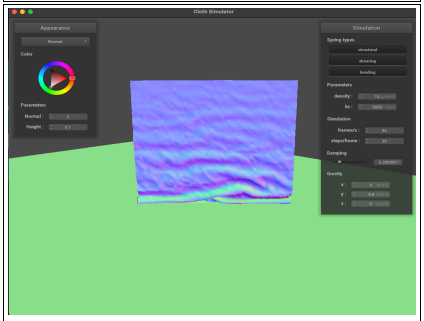

cloth on plane

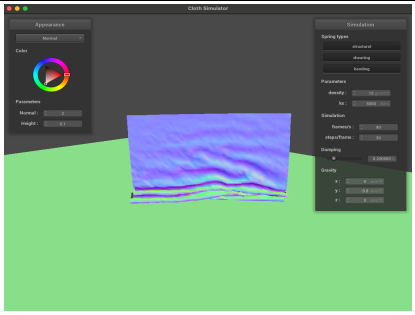

wireframe on plane

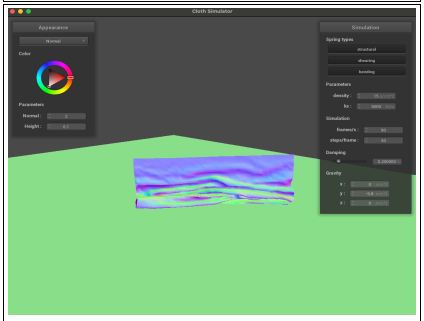
Part 4: Handling Self-Collisions

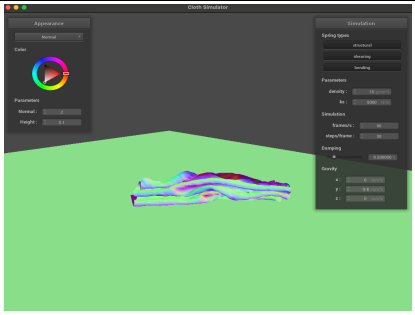
We implemented self collisions by first hashing each point mass into a bin. Then, we compare each point mass to its given bucket. Points are hashed according to a nearby box. If the point mass is within a certain distance from another one, then we add self-collision corrective forces. Below are images of self-collision at various timesteps.


self collision 0


self collision 1


self collision 2

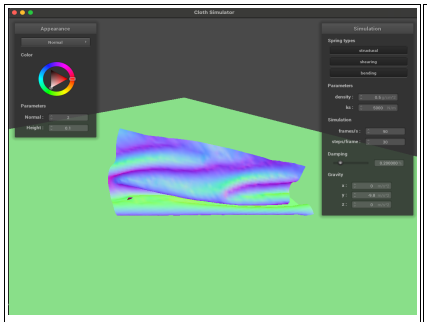

self collision 3

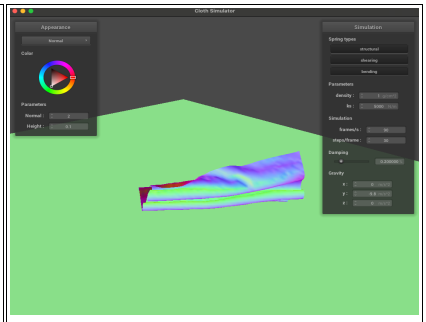

self collision 4

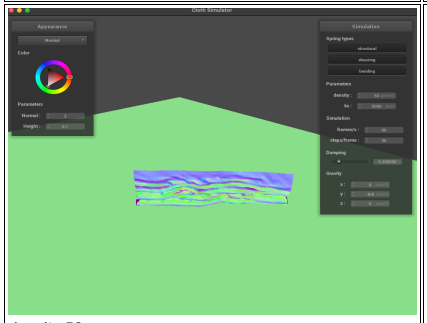
Then, we vary the density and k_s values to see how the cloth behaves.

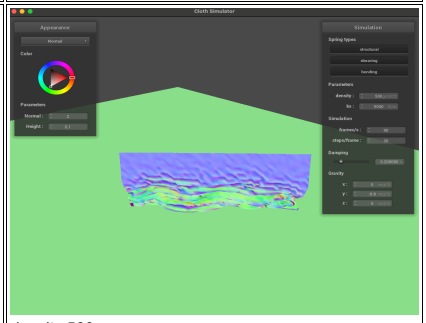
Varying density

We see that high densities create more wrinkles upon the cloth itself. The space between layers of these wrinkles is significantly smaller, corresponding to the higher density of the cloth. In contrast, lower densities have fewer wrinkles and the cloth is more taut. This is because the springs are more compressed and have more force applied to them. As a result, the cloth is less wrinkled.


density 0.5

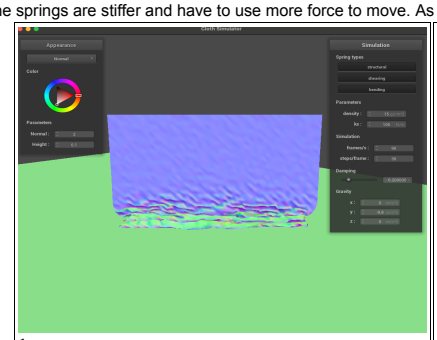

density 1

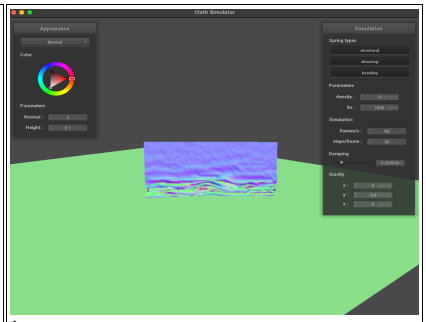

density 50

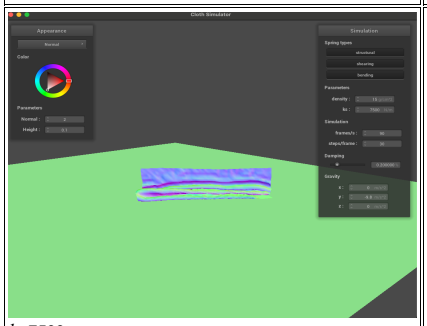

density 500

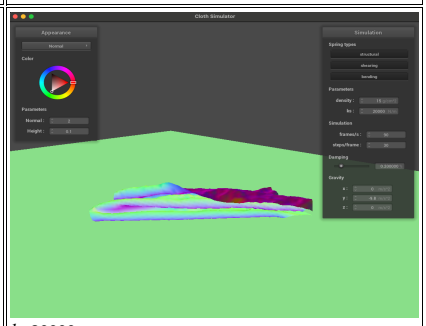
Varying k_s

We see that small ks values create more wrinkles upon the cloth itself. The space between layers of these wrinkles is small becuase the springs are easier to move (smaller spring constant). In contrast, higher ks values have fewer wrinkles and the cloth is more taut. This is because the springs are stiffer and have to use more force to move. As a result, the cloth is less wrinkled.


k_s 100


k_s 1000


k_s 7500


k_s 20000

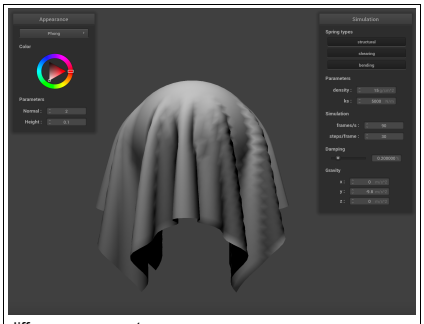
Part 5: Shaders

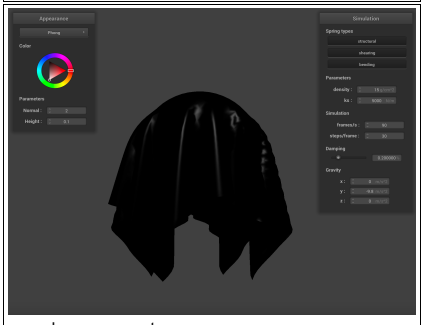
Shaders

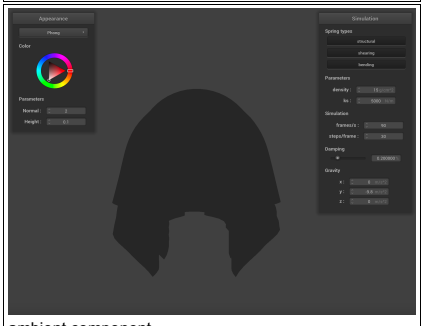
The shader programs run on the GPU and are written in GLSL. We use both vertex and fragment shaders in this project. A pipeline to take vertex data to screen output has the application send vertex data to the vertex shader which then sends data the the fragment shader that determines the colors to output to the device. Depending on the choice of shading, a vertex / fragment shader may be chosen. The vertex shader is responsible for transforming the vertex position, normal, and texture coordinates into screen space. It may also apply things such as bump mapping. The fragment shader is responsible for determining the color of the pixel.

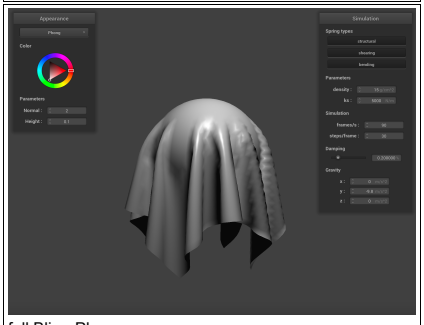
Blinn-Phong shading

The Blinn-Phong shading model consists of ambient, diffuse and specular components. The ambient component the light from the environment on the object. In Blinn-Phong shading, the ambient component is constant for each part of the object multiplied by the ambient coefficient. Next, the diffuse component represents the total irradiance arriving at the surface. The diffuse component is calculated by the following equation: $L_{diffuse} = k_{diffuse} \cdot (\frac{I}{r^2}) \cdot max(0, L \cdot N)$ with the $L \cdot N$ component representing that of Lambert's cosine law. Finally the specular component represents the specular reflection of an object, which can only be seen at certain angles. This shiny reflection is determined by a half-vector between the light vector and the eye vector.


diffuse component

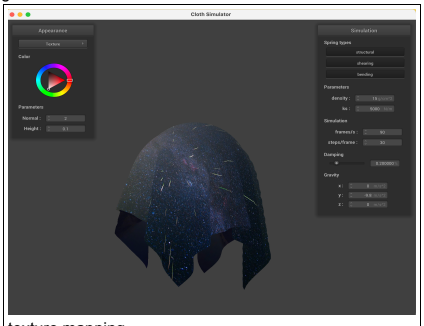

specular component


ambient component

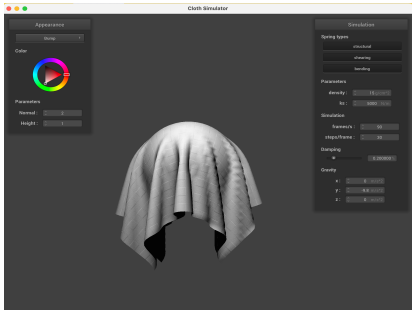

full Blinn-Phong

Custom Texture

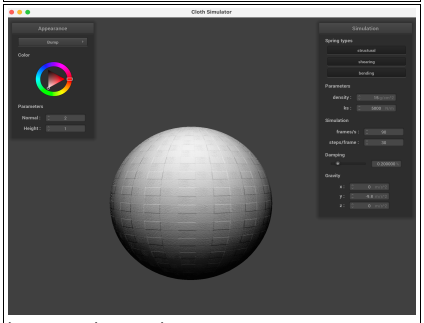
We also show our custom texture mapping below:


texture mapping

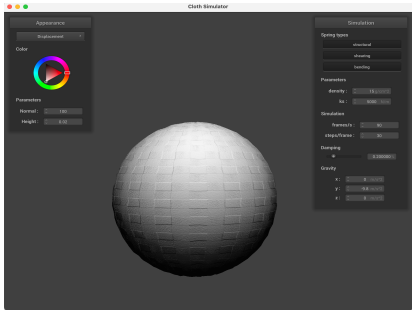
Bump mapping



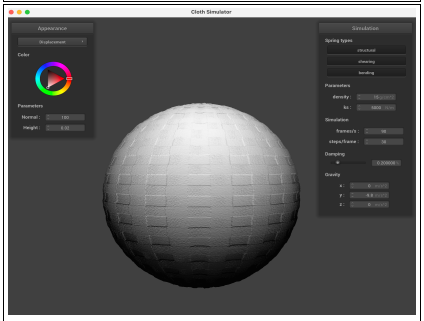
bump mapping on cloth



bump mapping on sphere



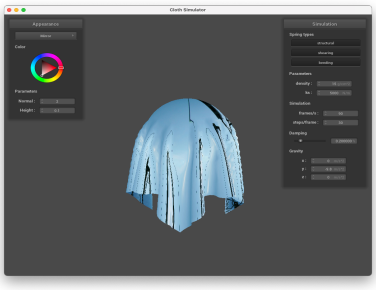
displacement mapping on sphere



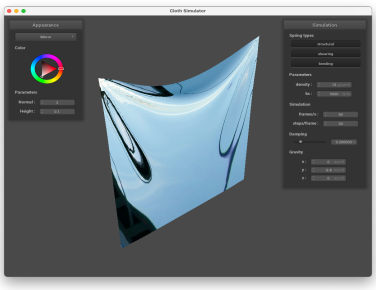
displacement mapping on sphere with a coarser mesh

Although it is a little difficult to see from the figures immediately, zooming in, we can se that bump mapping and displacement mapping have different effects on the object. In the case of bump mapping, we get a smoother shape with a rough surface. This is because the normal of the object is modified in the fragment shader, which changes the way light interacts with the object. In contrast, displacement mapping modifies the vertex position in the vertex shader, which changes the shape of the object. As a result, the object has a rough surface and a distorted shape. On the other hand, the sphere with displacement mapping has a rough surface and the shape is distorted. This is because the vertex position is modified in the vertex shader, which changes the shape of the object. Additionally, the courseness of the mesh can also affect the result as shown above with displacement mapping. A coarser mesh has a more distorted shape and rougher surface. This is because the vertex position is modified in the vertex shader, which changes the shape of the object. As a result, the object has a rough surface and a distorted shape.

Mirror



mirror shading on a sphere



mirror shading on a cloth