CS 184 Proj4 Write Up

Yao Fu Zuo and Bohan Yu

Overview:

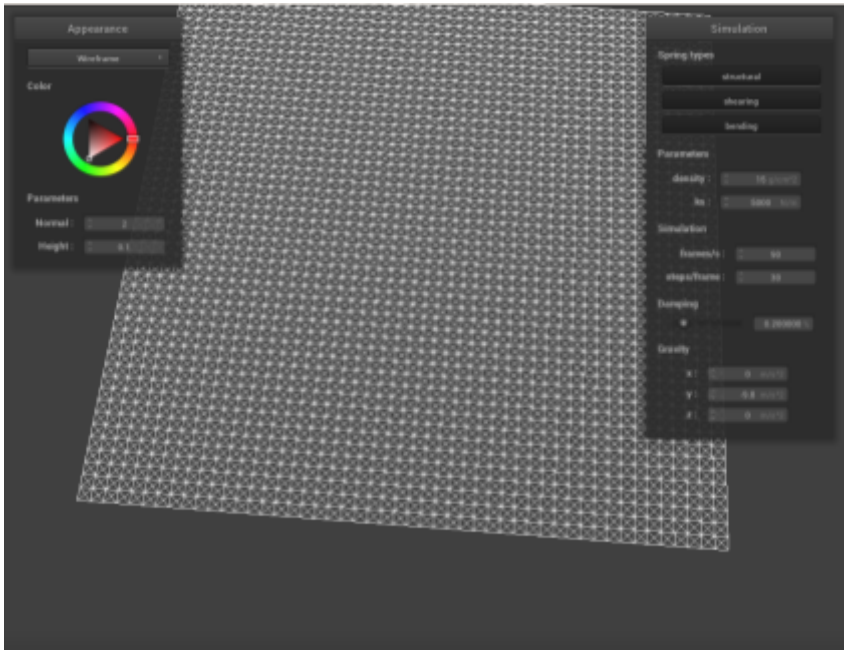
In this project, we simulated the physics of cloth using a mass and spring-based system. We also applied different shaders and mapping techniques to make the cloth look pretty.

Part 1:

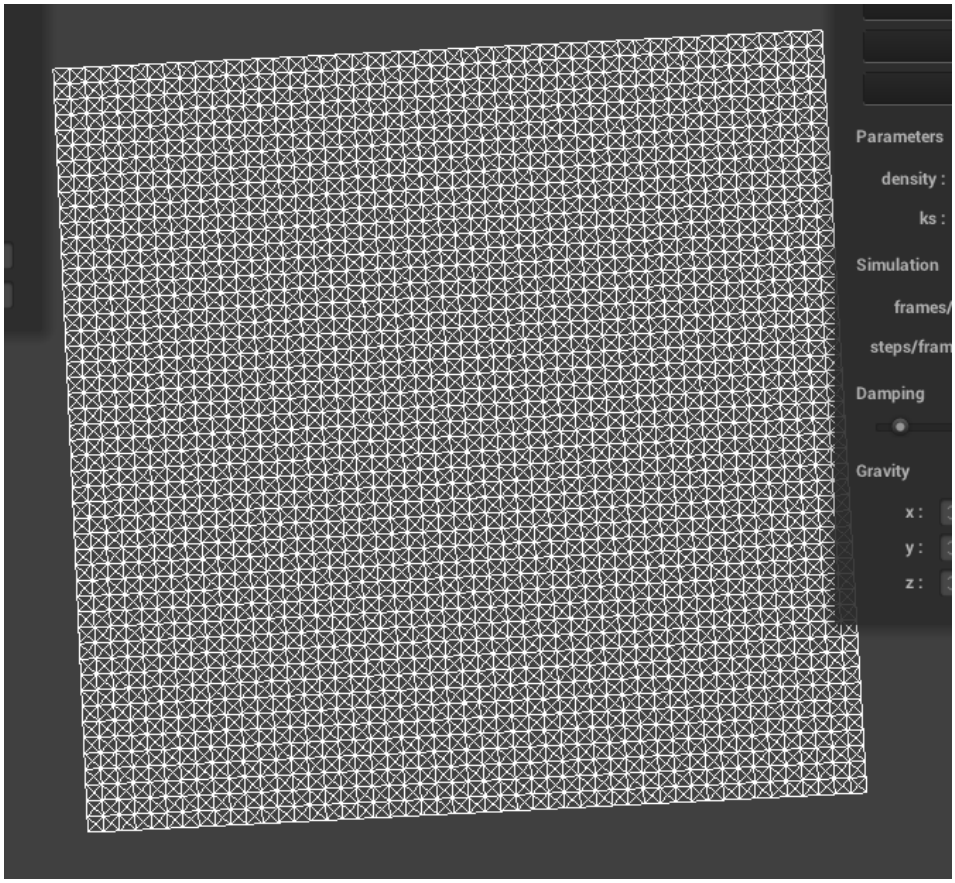
Implementation:

First, we created all the point masses by looping through all the width and height points in row-major order and added them to the point_masses vector. For each point mass, we added their 6 corresponding springs (2 structural springs, 2 shearing springs, 2 bending springs). Some point masses near the boundary may not have 6 springs, so we added a few bound checks to prevent any out-of-bounds error. We didn't encounter any major problems.

Screenshots of pinned2.json:

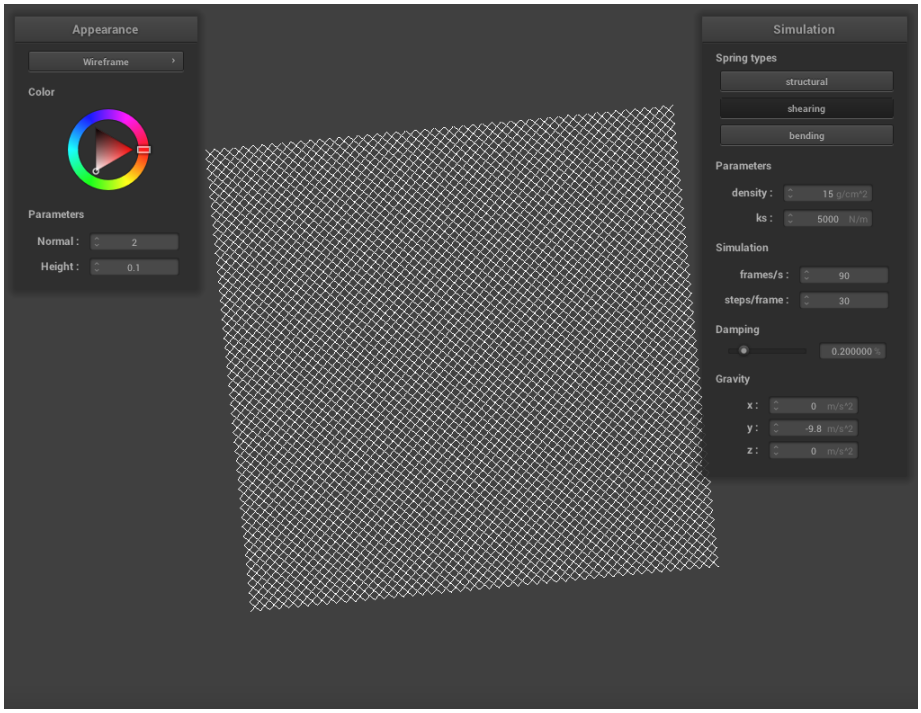


A bird-view of the complete wireframe:



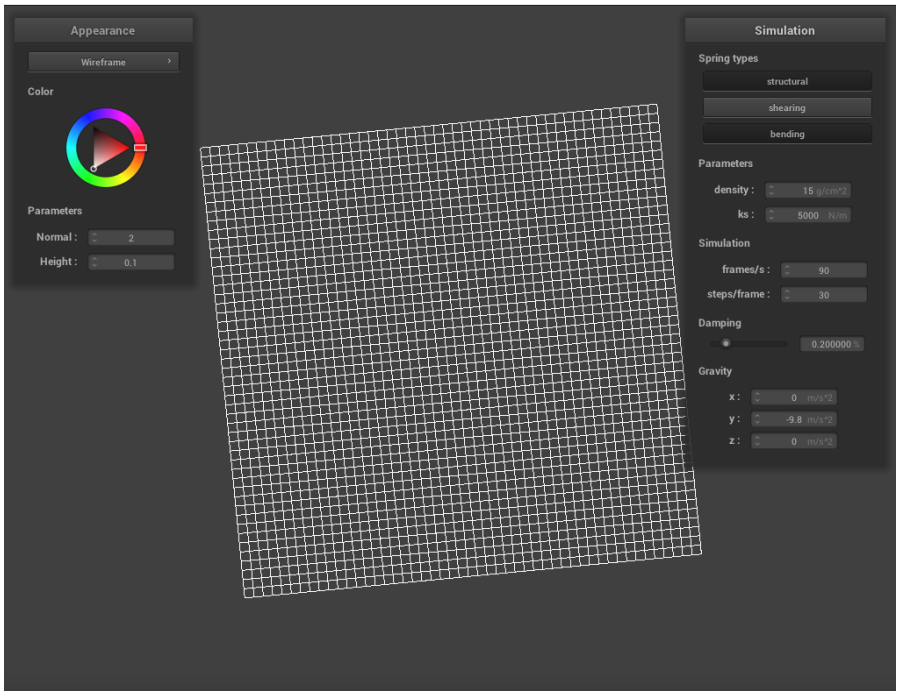
The basic structural springs could be seen clearly by the grid lines going horizontally and vertically. The diagonal lines at every grid square indicate the correctness of the shearing springs.

Wireframe with only shearing constraints:



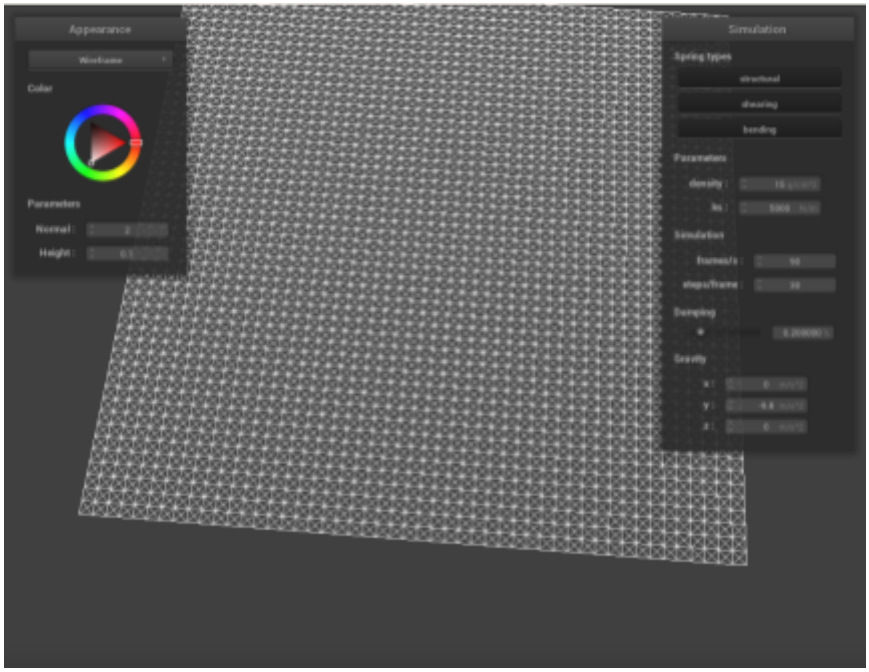
The diagonal lines indicates the shearing springs. There are no horizontal or vertical lines, indicating that there are no extra structural or bending springs labeled as shearing springs.

Wireframe without any shearing constraints:



The horizontal and vertical lines indicate the structural and bending springs. There are no diagonal lines or any shearing springs.

Wireframe with all constraints:



This image shows not only the structural and bending springs(as shown in the horizontal and vertical lines) but also the shearing springs(as shown in the diagonal lines).

Part 2:

Implementation:

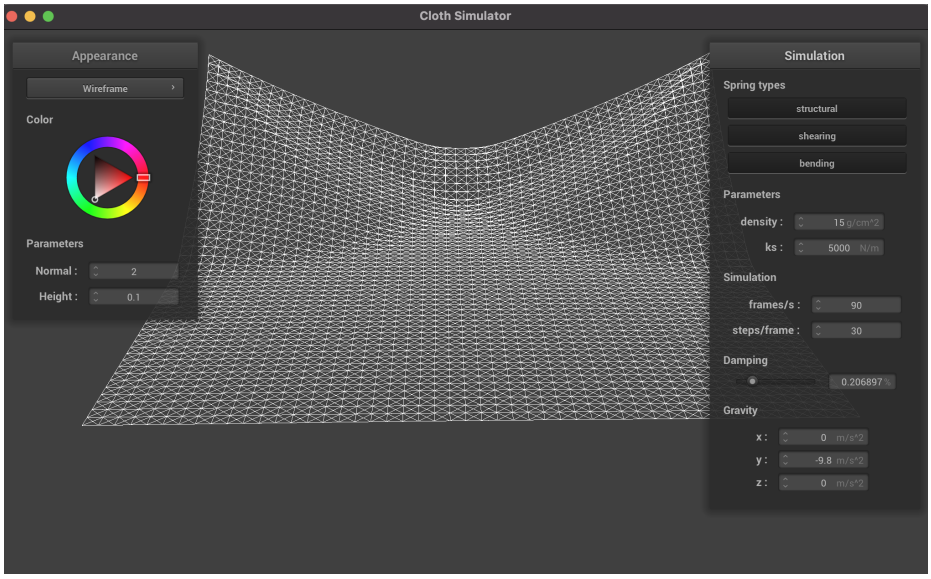
We first loop over the external_acceleration to compute the net force exerted on each point mass. We then loop over all springs to update the net force exerted on each point mass (i.e. add the spring force to existing net force for each point mass). Then, we update each point mass' position based on its net force. An important step is to check whether the spring, after updating the position of its two point mass, is too long. We check if the spring's length is greater than or equal to 1.1 * its rest length. If that's the case, we update each point mass' position again, subject to whether the point mass is pinned or not, so that the spring is not elongated. This gives a really good simulation! We didn't encounter any major problems.

Difference between a high and low damping:

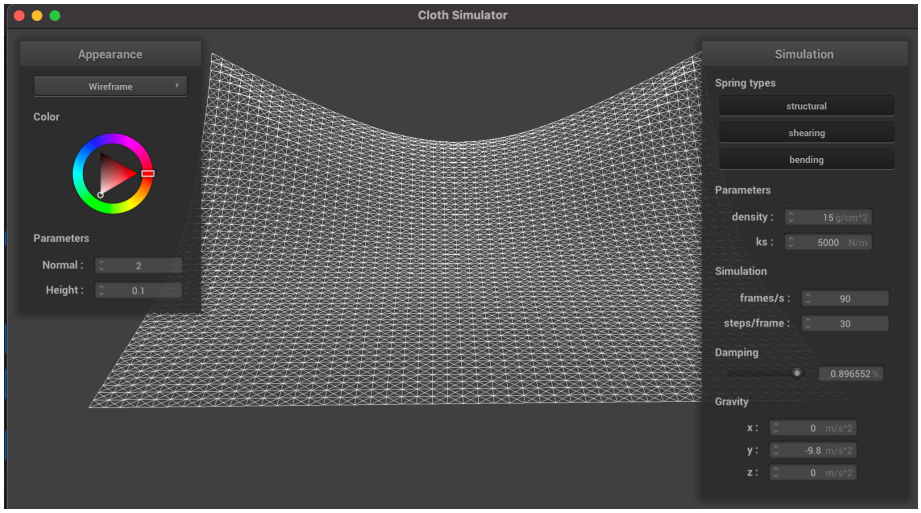
With very high damping, we can barely observe any "back-and-forth" swings as the cloth falls down. In contrast, the lower the damping, the more "back-and-forth" swings we can observe before the cloth finally comes to a rest. The speed of those "back-and-forth" swings is also slower with high damping, compared to that of low damping. Lastly, with a low ks, the cloth can easily get wrinkled as it falls down, whereas with a high ks, the cloth falls down without many wrinkles as it's so damped that the wrinkles couldn't form (or are less obvious). The screenshot below shows this:

Note: Please zoom in on the image since zooming out will generate a Moire pattern for some reason.

Damping = 0.20 (default):



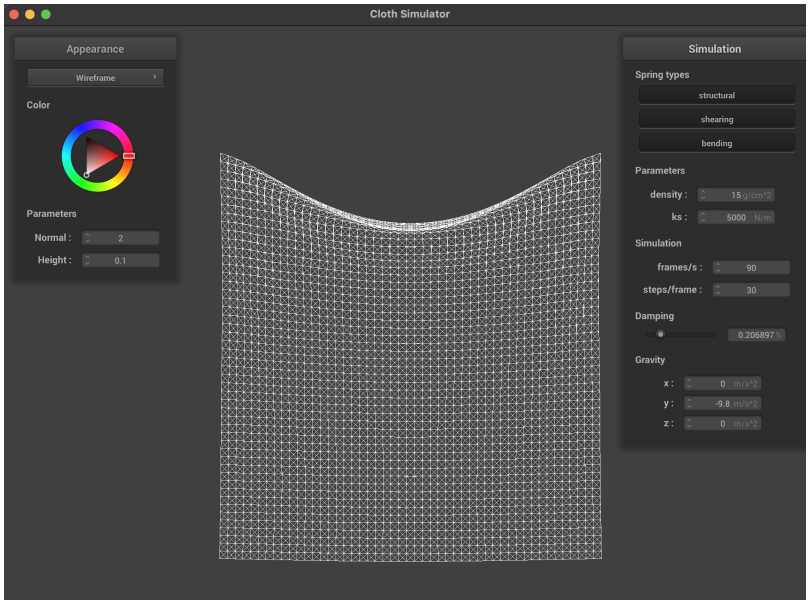
Damping = 0.89:



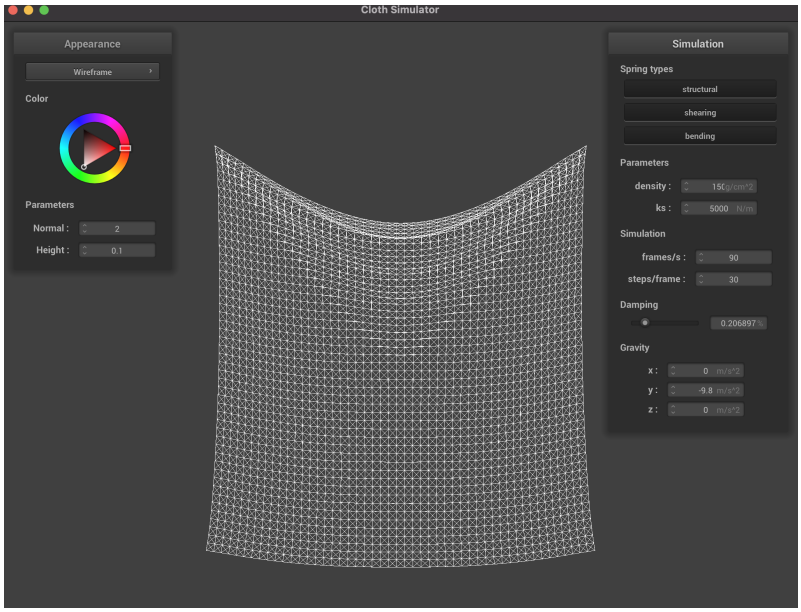
We can clearly see that with a small damping, the wrinkles at the top are “bigger” and more obvious than those with a high damping. This occurs because with a high damping, the springs are so damped that the oscillation could not “fold” the cloth that much, therefore the wrinkles are less obvious compared to those caused by a low damping, in which case the oscillations quickly folds the cloth and generate the wrinkles.

Difference between a high density and a low density:

Density = 15 (default):



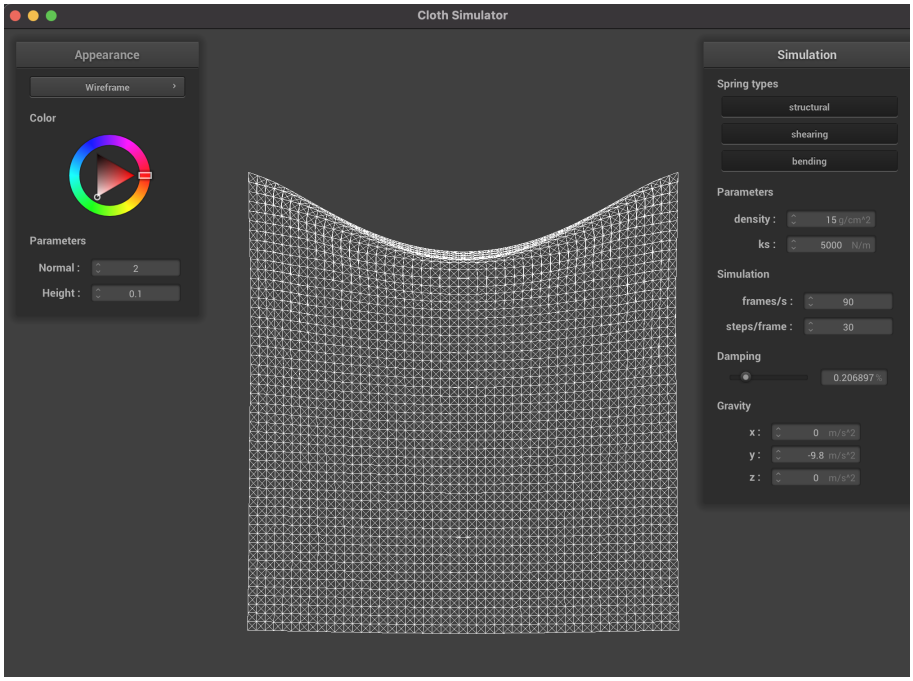
Density = 150:



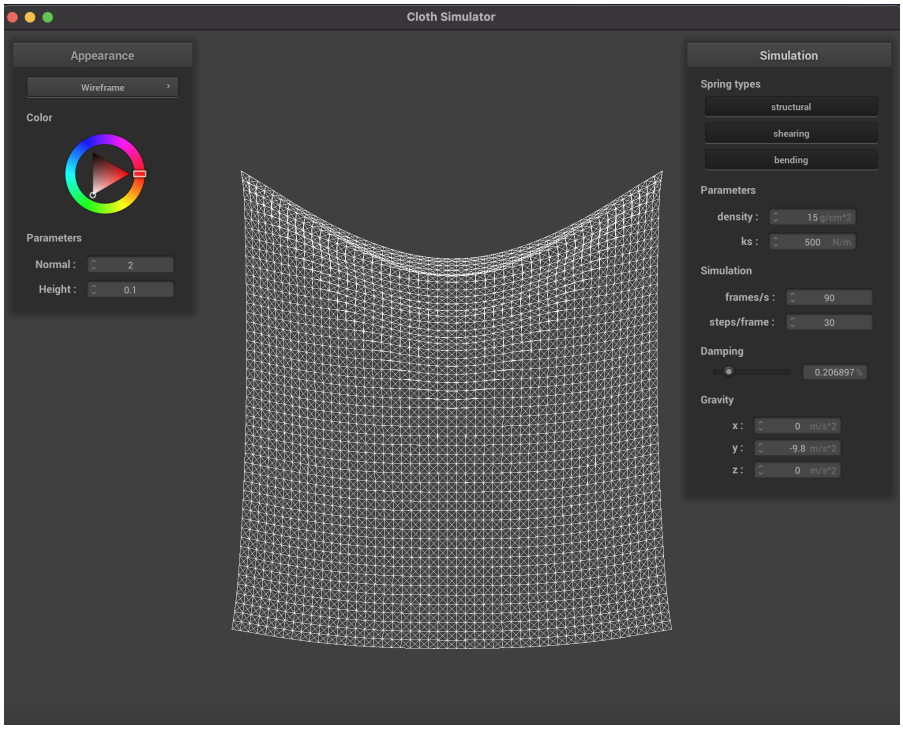
As the density increased from 15 to 150, we can see much clearer downward wrinkles on the cloth. The whole cloth appears to be quite heavy, dragging down the middle part that is not pinned. This occurs because the mass of the point masses are larger as density increases, so they have a larger gravitational force pulling on them. Thus, the springs associated with those point masses would be pulled further down, resulting in larger wrinkles pointing downwards(towards the gravitational force).

Difference between a high ks and a low ks:

Ks = 5000 (default):



Ks = 500:

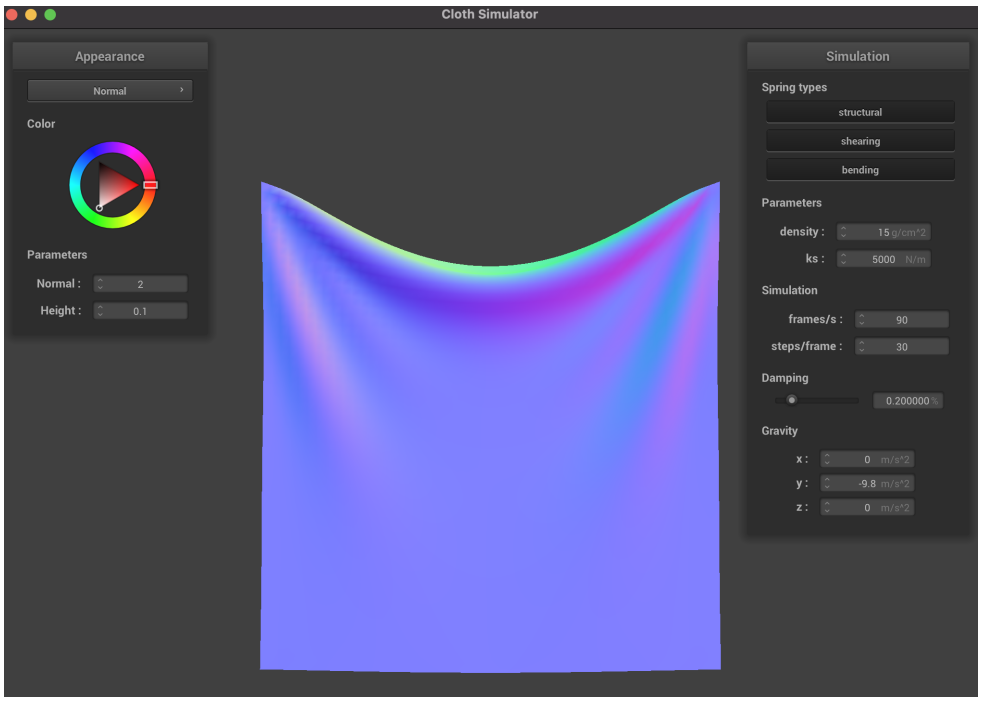


As K_s decreased, we can see much clearer downward wrinkles on the cloth. This occurs due to the Hooke's law:

$$F_s = k_s * (||p_a - p_b|| - l)$$

As shown in the formula, as ks decreases, the rebound-force of the springs also decreases. Thus, there would be less force countering the gravity force pulling the point masses downwards, so the downward net forces acting on those point masses would increase. As a result, the point masses would be pulled more downwards, resulting in larger downward-pointing wrinkles.

Shaded cloth from *scene/pinned4.json* in its final resting state (default):



Part 3:

Implementation:

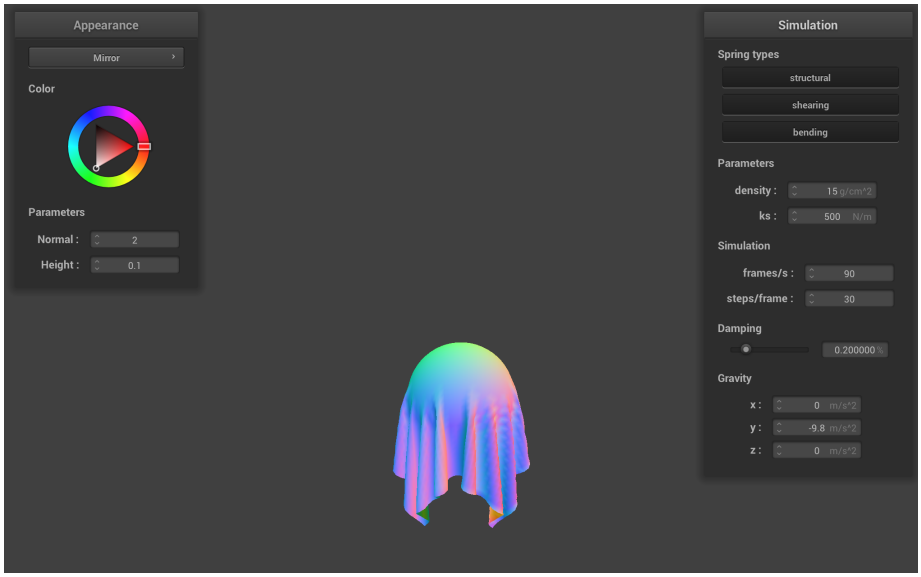
Sphere collision: we first calculated the distance from the point mass's position to the sphere. If the distance is less than the radius of the sphere, then that point mass is inside the sphere and needs correction! So we calculated the correction vector that pushes the point mass's last position back to the surface of the sphere. Finally, we set

the point mass's new position to be its last position, adjusted by correction vector * (1-friction).

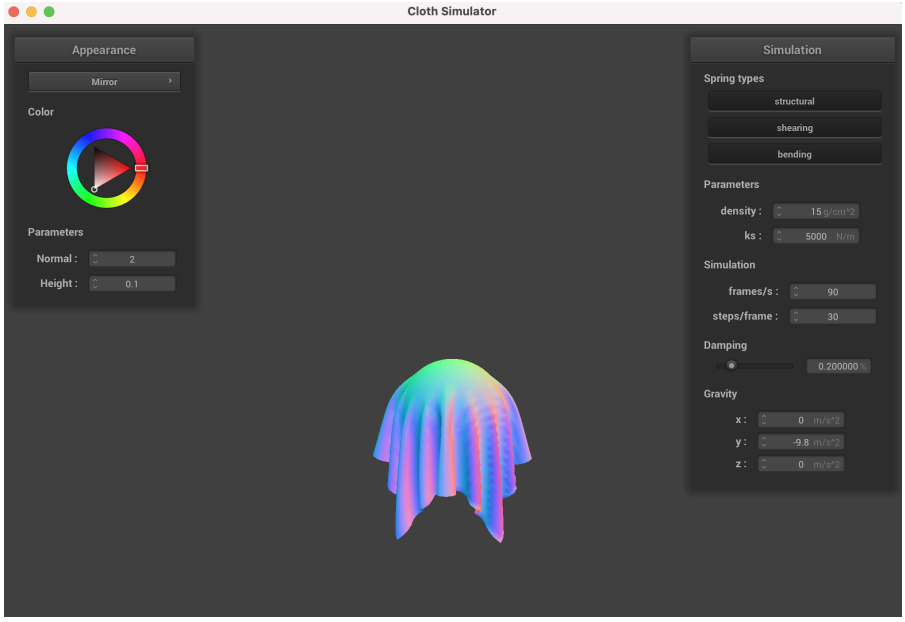
Plane collision: we first calculated the tangent point, which is where the point mass would intersect with the plane if it traveled in a straight line. If the point mass crossed over, we then calculated the correction vector that would push the point mass's last position back to slightly above the plane(on the same side that the point mass fell from). At last, we set the point mass's new position to be its last position adjusted by the correction vector multiplied by(1-friction).

Luckily, we didn't encounter any major problems!

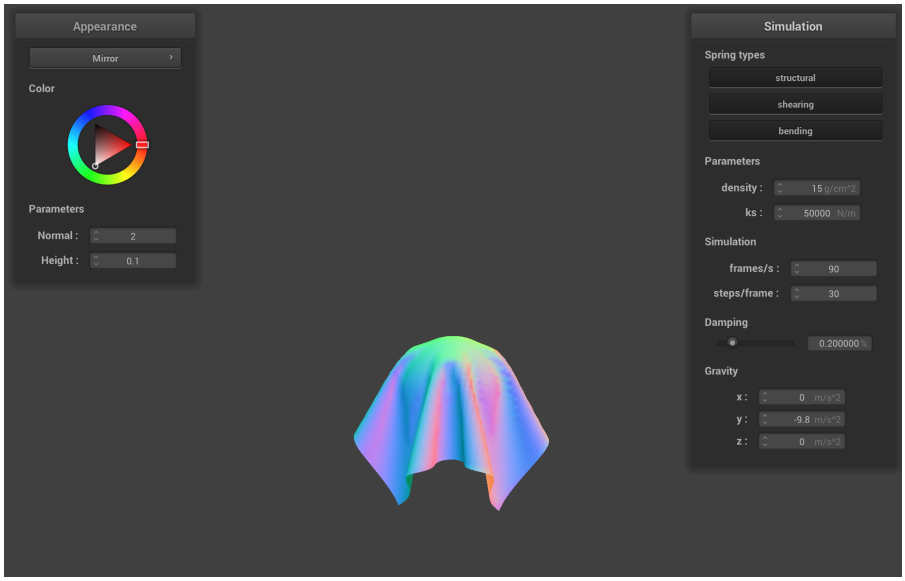
Cloth falling onto sphere with Ks = 500:



Cloth falling onto sphere with Ks = 5000:

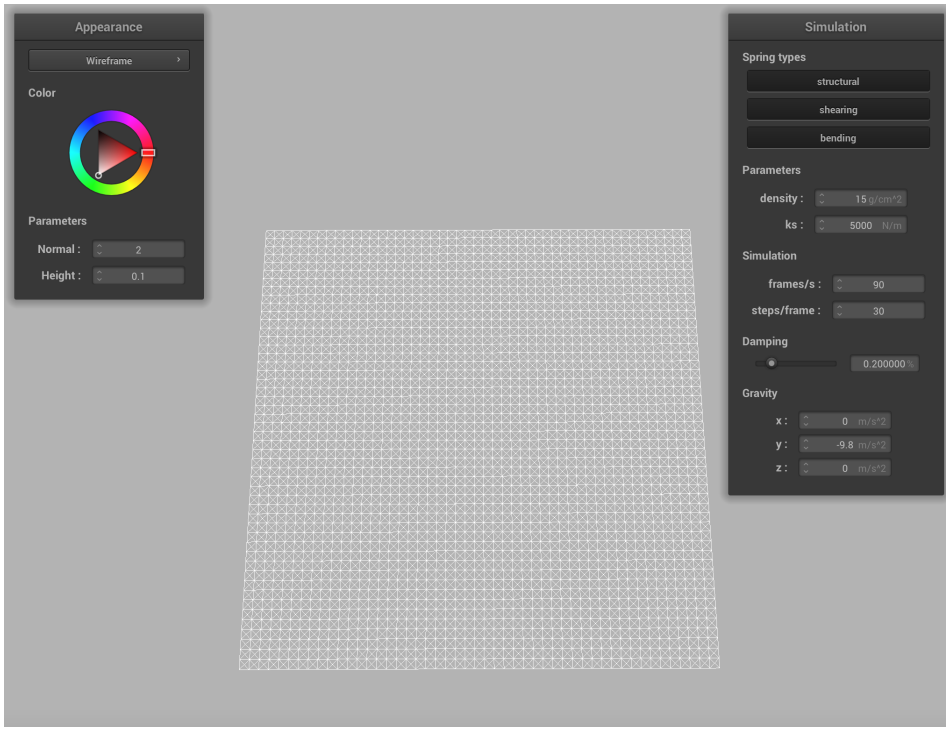


Cloth falling onto sphere with Ks = 50000:



The first noticeable difference is that as we increase the ks, the less downard the cloth drops, since the counter-force of the gravity exerted by the spring is larger. With ks = 500, the position of the edges of the cloth is lower than that of with a high ks. With a high ks, the resulting cloth appears to be “fatter”, and with a low ks, it becomes “thinner” as it’s more attracted by the downward gravity.
The second noticeable effect is that with a high ks, there are less wrinkles on the cloth. This is caused by a larger spring force that makes “folding” much more difficult.

The cloth lying peacefully on the plane:



Part 4:

Implementation:

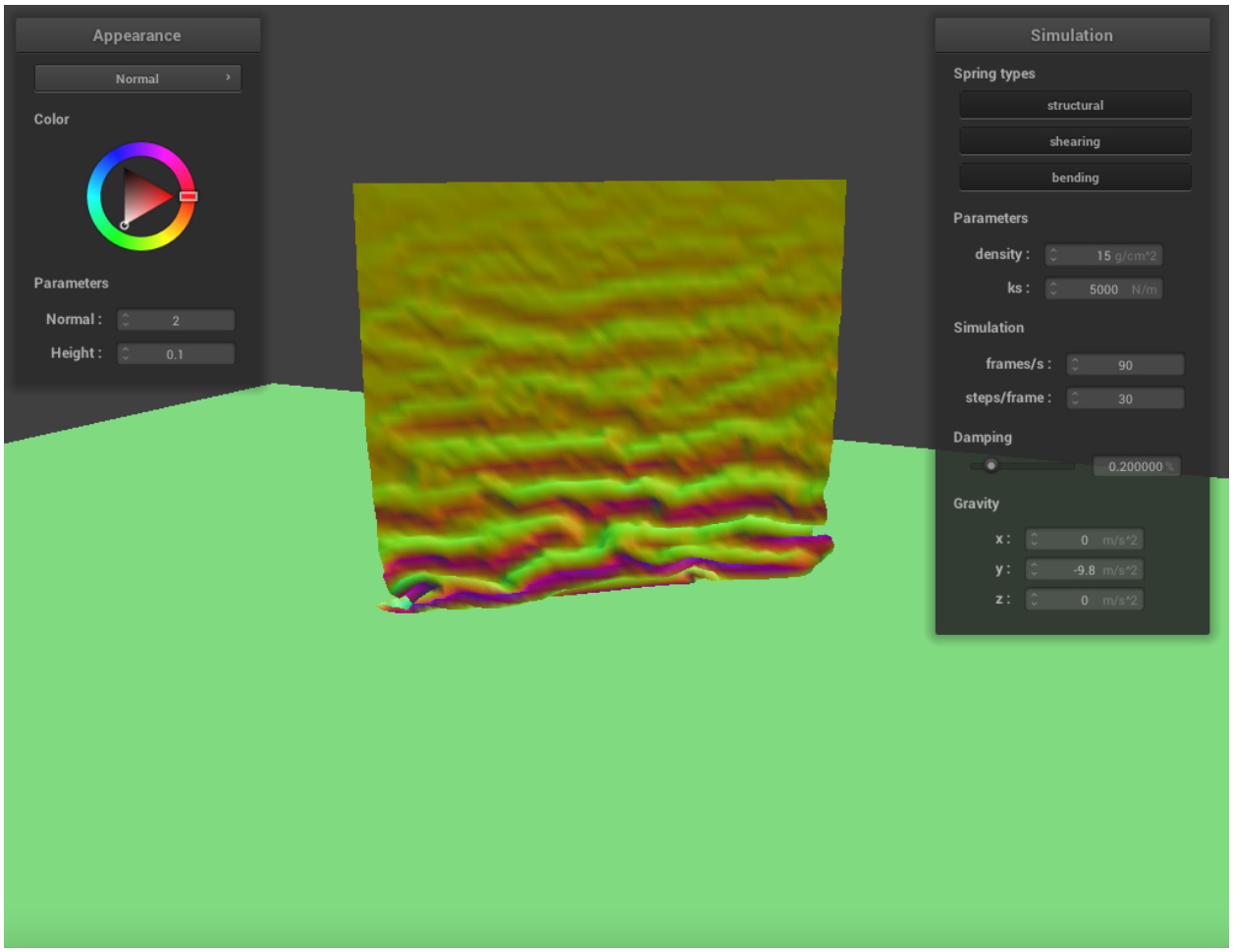
1. Hashing a position to a float, Cloth::hash_position()
    a. We first calculated the dimension of the 3D box by applying the formula on the spec.
    b. Then, we calculated the box coordinates (x,y,z) of the position vector that was passed in. This is done by dividing the original coordinates by the box size along its corresponding dimension.
    c. At last, we hashed the box coordinates to a float using this formula: (x * 29 + y) * 29 + z. We used the prime number 29 because it is larger than the maximum value that x, y, or z can take.
2. Building the spatial map, Cloth::build_spatial_map()
    a. First, we cleared the map.
    b. We looped over all the point masses. For each point mass, we calculated its position's hash value(using the function we created in step 1). Using

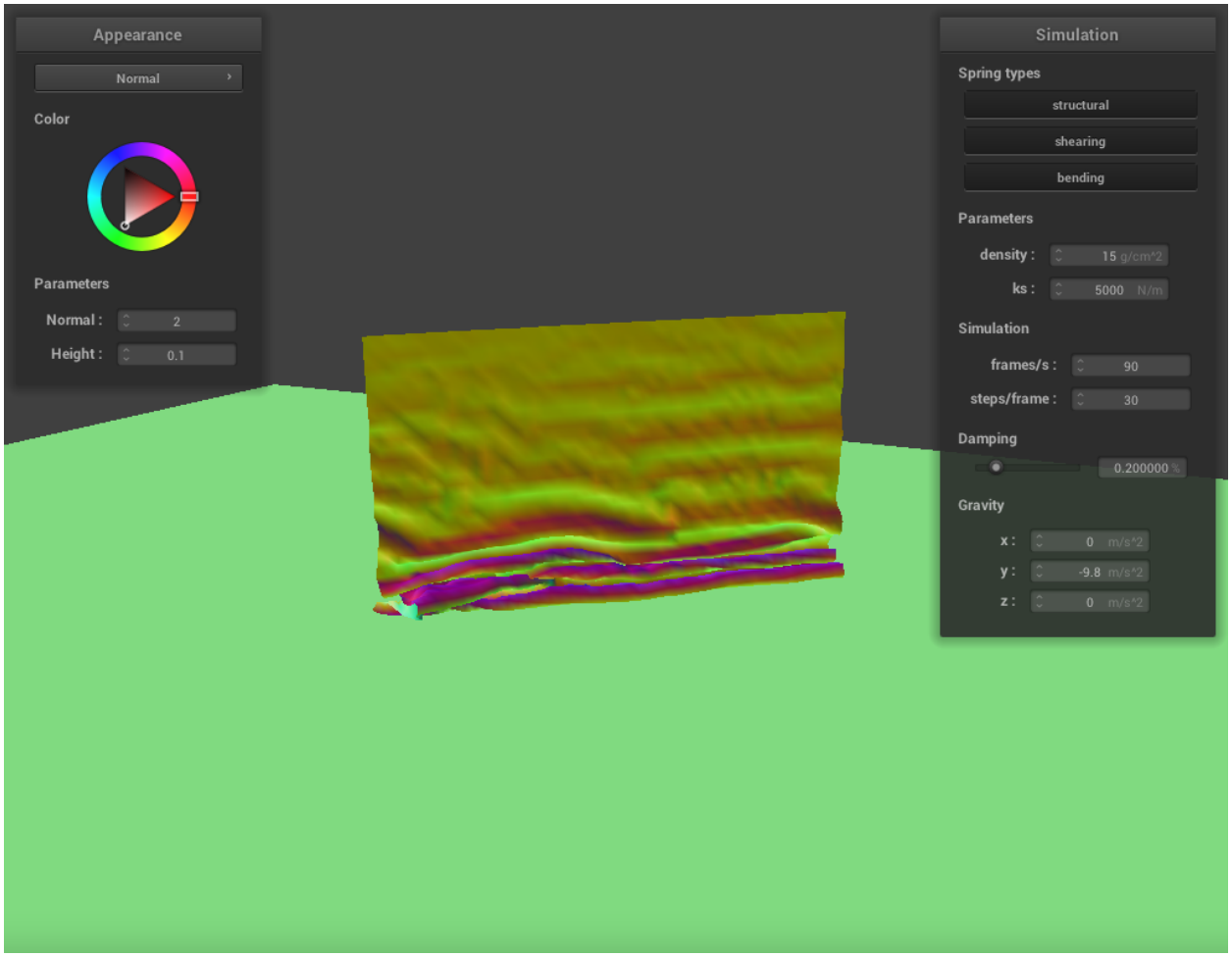
that hash value as the key to the map, we store that point mass's pointer into its corresponding vector of point masses.

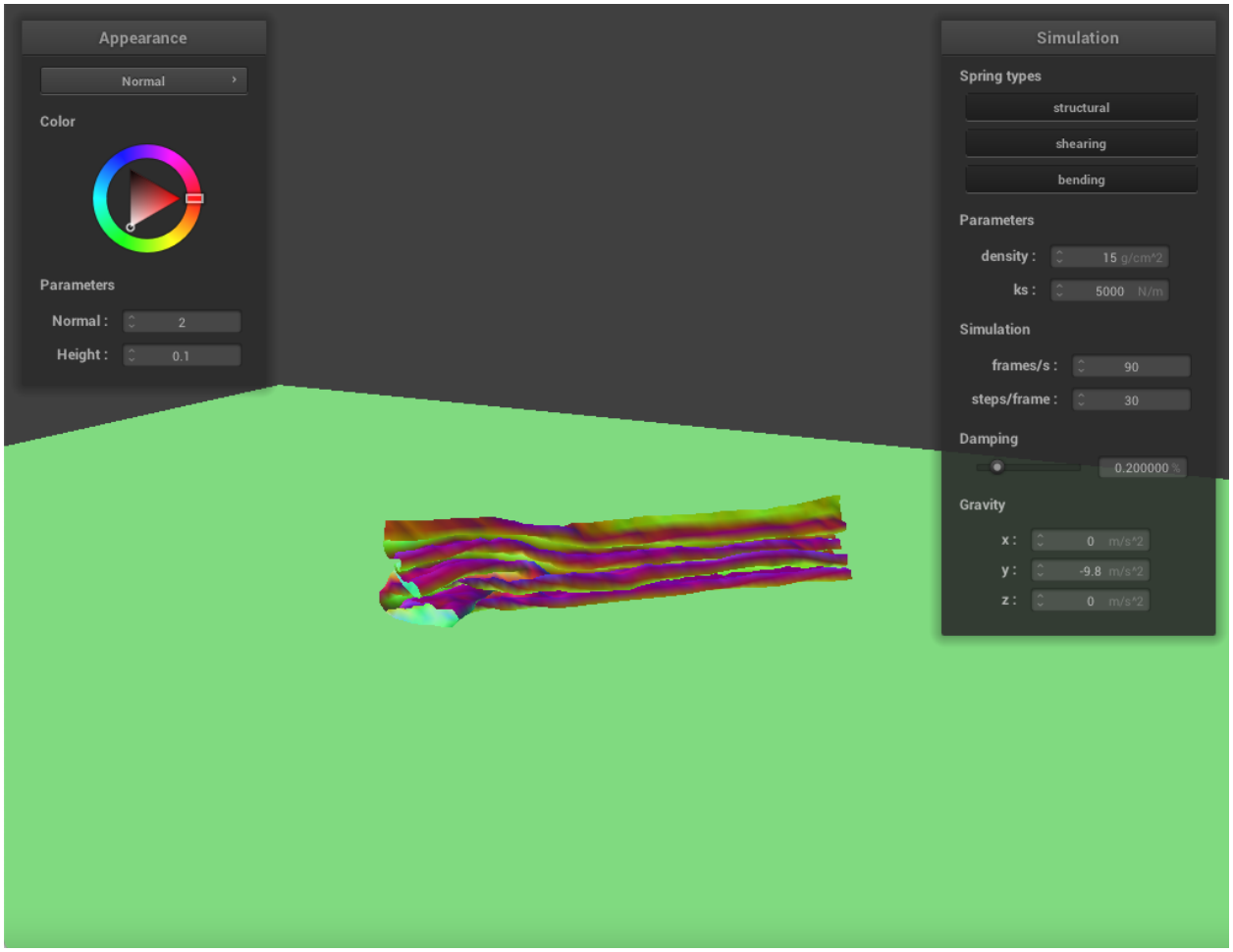
3. Cloth::self_collide()
    a. First, we calculated the point mass's position's hash value. Using its hash value, we got all of the candidate point masses that are close to it.
    b. For each candidate point mass that is not itself(which we check by comparing the memory address), we calculated the distance between the candidate point mass to the point mass. If that distance is less than 2 * thickness, we calculated the correction vector that would push the point mass's position to 2*thickness apart from the candidate.
    c. At last, we averaged all the correction vectors to get a total correction vector. We then scaled the total correction vector down by simulation_steps and adjusted the point mass's position by said vector.

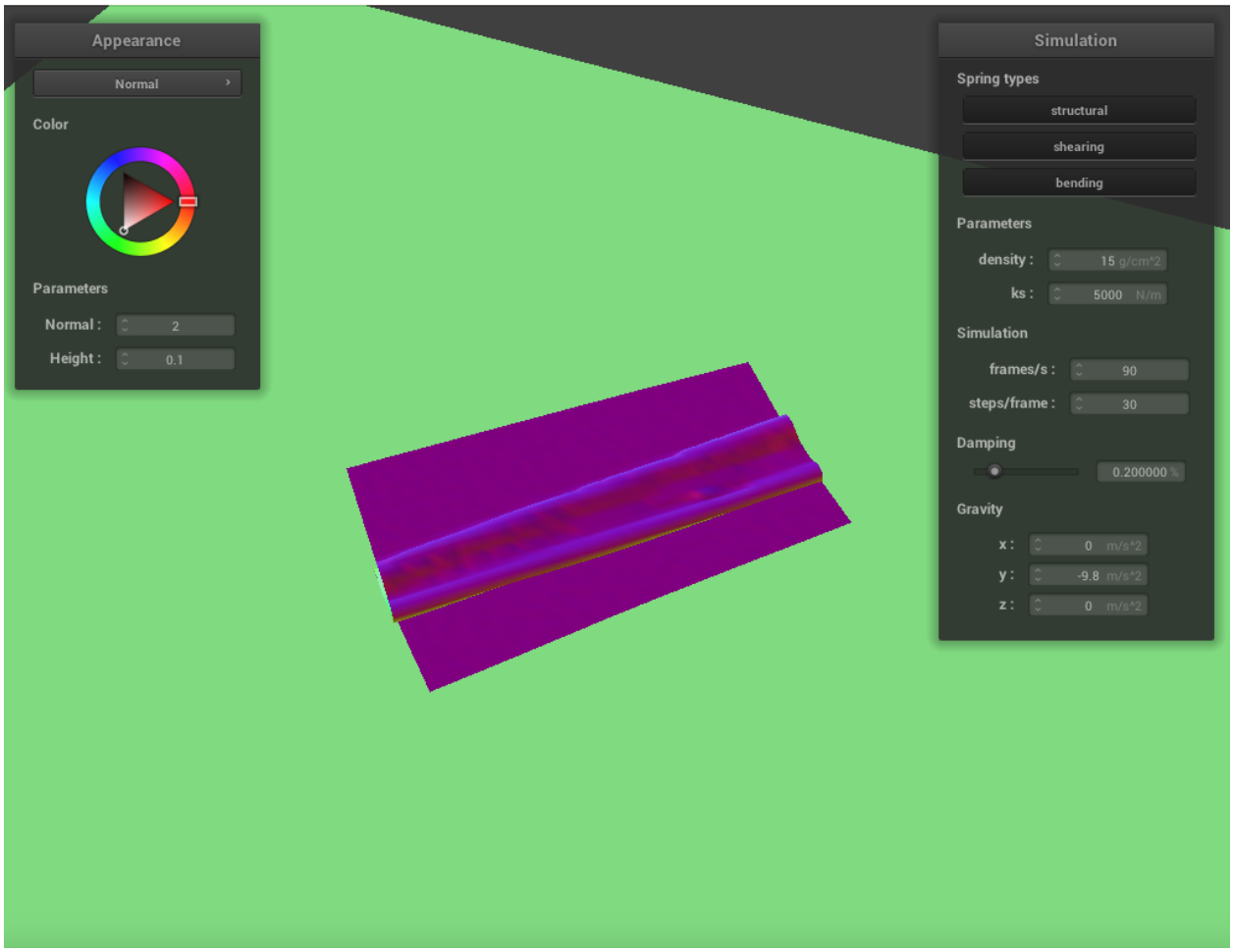
We didn't encounter any major problems.

4 Screenshots that document how the cloth falls and folds on itself, in chronological order:


Appearance
Normal
Color
Parameters
Normal : 2
Height : 0.1
Simulation
Spring types
structural
shearing
bending
Parameters
density : 15 g/cm^2
ks : 5000 N/m
Simulation
frames/s : 90
steps/frame : 30
Damping
0.200000 %
Gravity
x : 0 m/s^2
y : -9.8 m/s^2
z : 0 m/s^2


Appearance
Normal
Color
Parameters
Normal : 2
Height : 0.1
Simulation
Spring types
structural
shearing
bending
Parameters
density : 15 g/cm^2
ks : 5000 N/m
Simulation
frames/s : 90
steps/frame : 30
Damping
0.200000 %
Gravity
x : 0 m/s^2
y : -9.8 m/s^2
z : 0 m/s^2

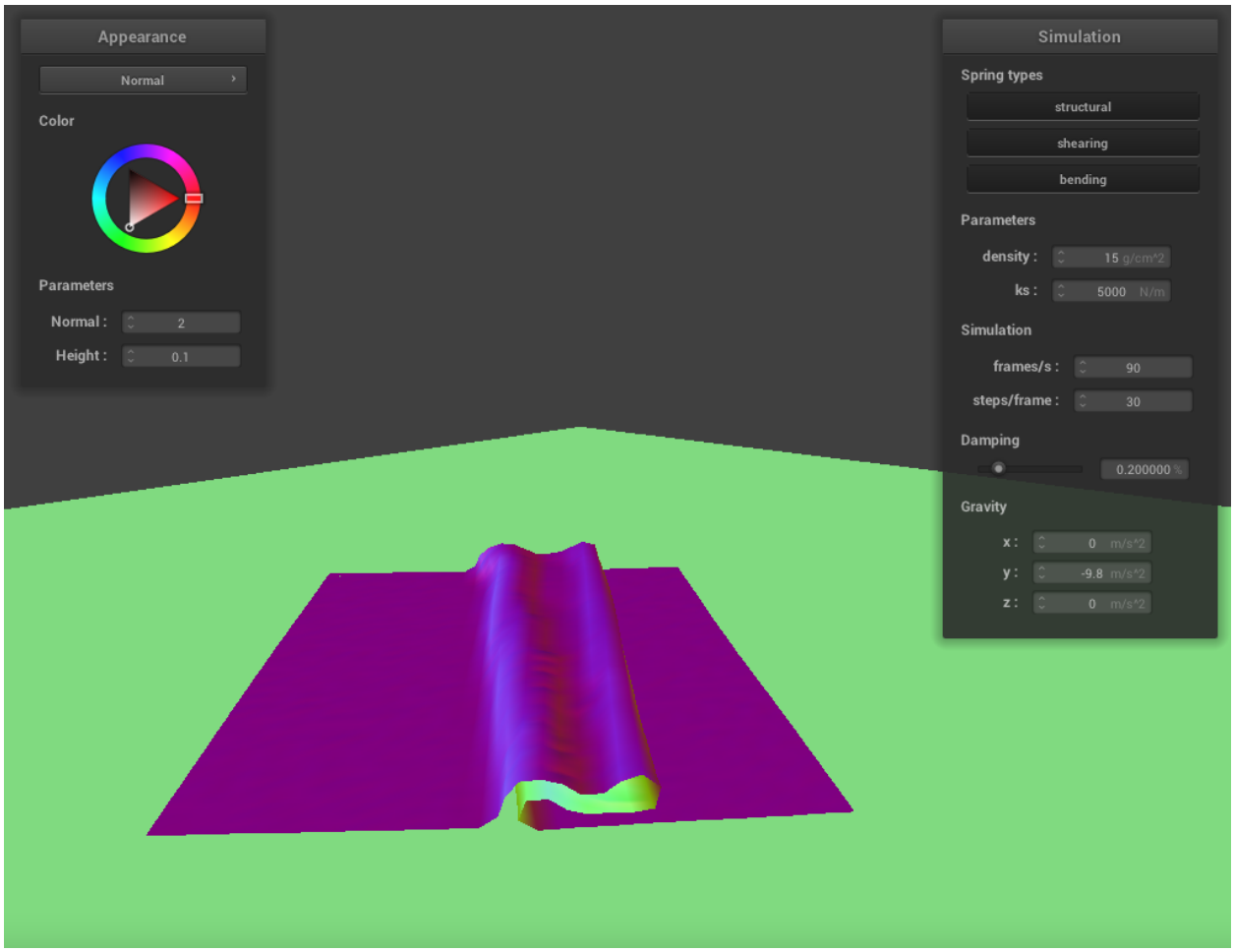

Appearance
Normal
Color
Parameters
Normal : 2
Height : 0.1
Simulation
Spring types
structural
shearing
bending
Parameters
density : 15 g/cm^2
ks : 5000 N/m
Simulation
frames/s : 90
steps/frame : 30
Damping
0.200000 %
Gravity
x : 0 m/s^2
y : -9.8 m/s^2
z : 0 m/s^2



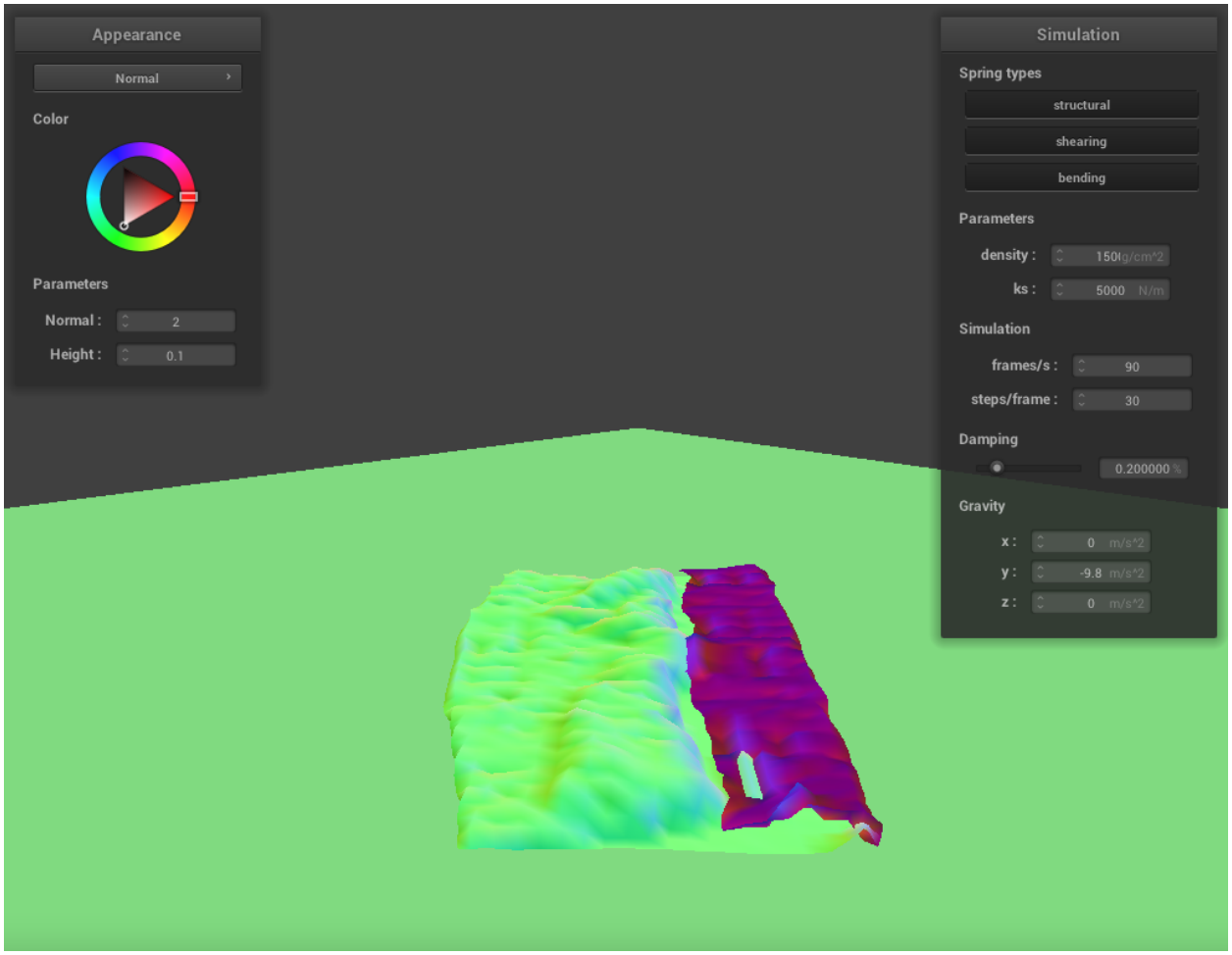
This series of images depicts how the cloth is falling and crumbling itself on a plane. The last image shows a bird-view of the cloth lying nearly flat, relatively restful on the plane.

Vary the density as well as ks and describe with words and screenshots how they affect the behavior of the cloth as it falls on itself.

density = 15, all other parameters are set to default setting:

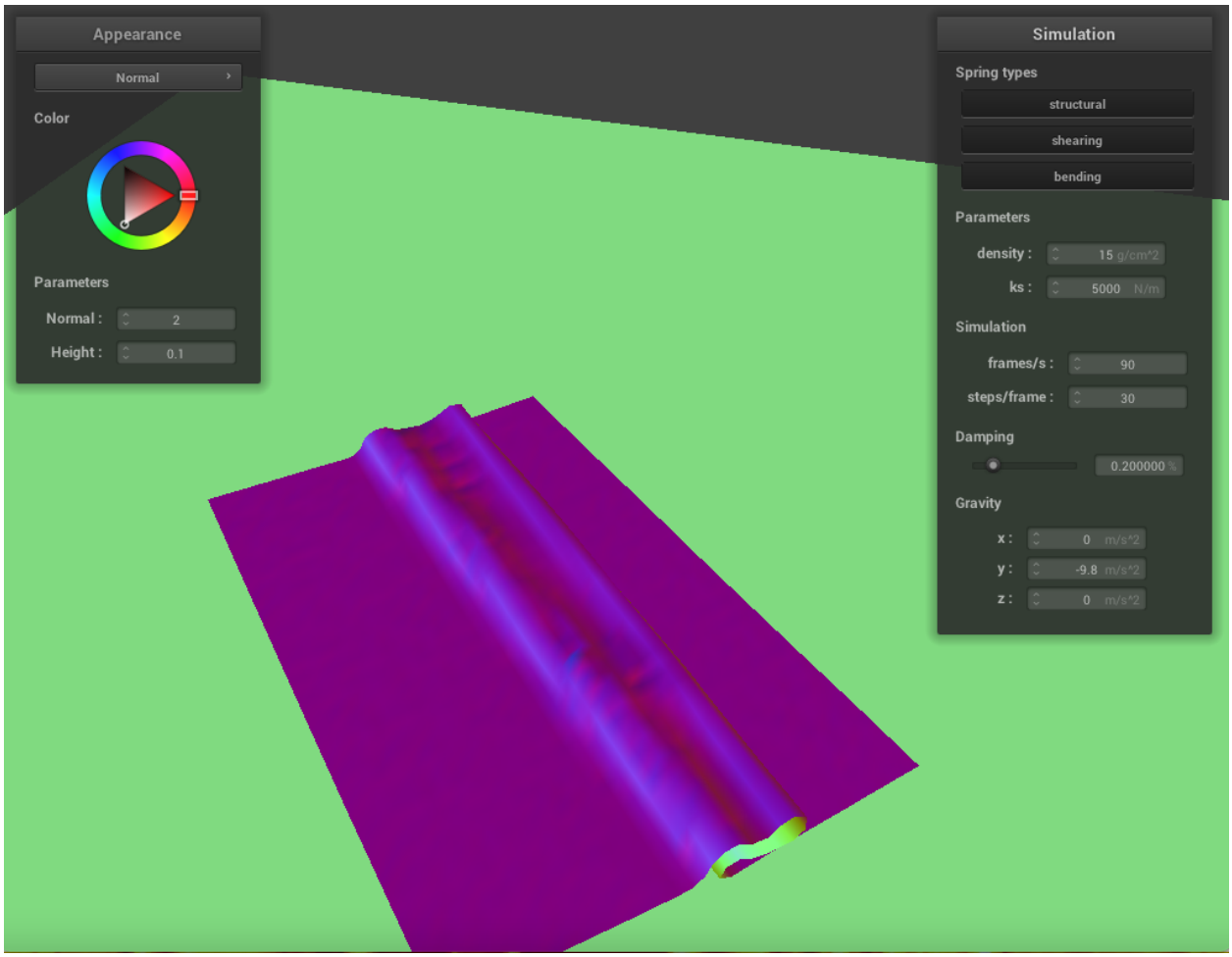


density = 1500, all other parameters are set to default setting:

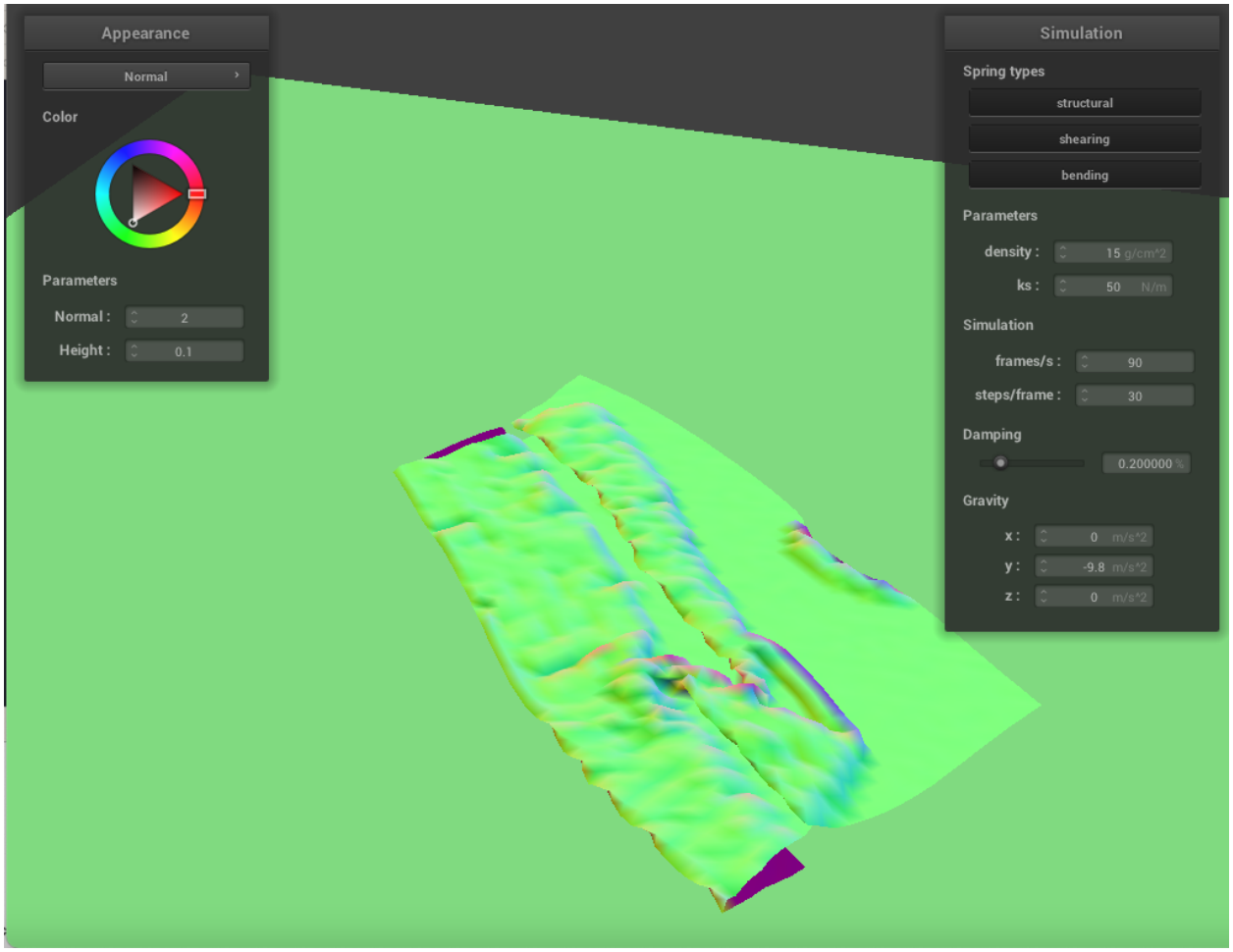


As density increases, there are more jumps and wrinkles as the cloth falls down. Also, the cloth is closer to the plane when in a restful state. This is due to a heavier point mass. As mass increases for each point mass, the gravitational force exhibited on each point mass increases. Thus, each point mass would be pulled more towards the plane, so the cloth overall is closer to the plane. As each point mass is pulled further down, the spring's rebounding force is also increased. Since the jumps are caused by the rebounding force of the springs, you can see more jumps in the image that has a higher density.

K_s = 5000, all other parameters are set to default:



K_s = 50, all other parameters are set to default:



As K_s decreases, we can see more bumps and wrinkles on the cloth. This is due to Hooke's law.

$$F_s = k_s * (||p_a - p_b|| - l)$$

It takes less force and is thus easier to deform a spring with a lower K_s. Since the bumps are caused by the deformation of springs, the cloth with a lower K_s would have more bumps and wrinkles.

Part5:

THIS PART IS PROBABLY THE MOST INTERESTING PART IN THIS PROJECT!!!!!!!

Brief Implementation:

We implemented the diffuse shading by setting the out_color to be the calculated light, using the formula in the lecture. We implemented texture-mapping by setting the out_color to be a sampling of the given texture at the given uv position. Everything went well until we implemented Bling-Phong shading, which requires quite a bit of tuning in its coefficients. We didn't find the best k_d, k_s, k_a, I_a for Bling-Phong to replicate the result shown in the spec. But we are sure that the implementation is correct as all effects from specular, diffuse, and ambient lighting can be shown clearly in the results (screenshots). We then implemented bump mapping, which interpolates the normal vector of the vertex by a small amount to create those little bumps, and we applied Bling-Phong shading to change the color. We implemented Displacement-mapping by copying the code from bump mapping, except that we now also change the position of the vertex (not just modifying the normal vector by a small amount). This essentially "displaces" the underlying geometry of the cloth and the sphere. Lastly, we implemented mirroring by calculating the w_in and setting the out_color to be a sampling of the given texture at the direction of w_in.

What is a shader program? How vertex and fragment shaders work together to create lighting and material effects?

A shader program essentially does the following things:

1. Calculating the position / normal of a vertex so that we know where we should render that vertex on the screen. In other words, it calculates the geometry of a primitive.
2. Calculating the color of that vertex so that we know which color to render at that position on the screen. In other words, it shades the primitive with colors.

To accomplish this process, the shader is broken into two parts: the vertex shader and the fragment shader. The vertex shader computes the geometry of the primitives, and the output from the vertex shaders is fed to be the input of fragment shaders, which can then calculate the light / color at that vertex given its position. The output of the fragment shaders are used for rendering. Changing vertex positions in the vertex shader and changing the way we calculate the light received by that vertex in the fragment shader gives us various beautiful lighting and material effects.

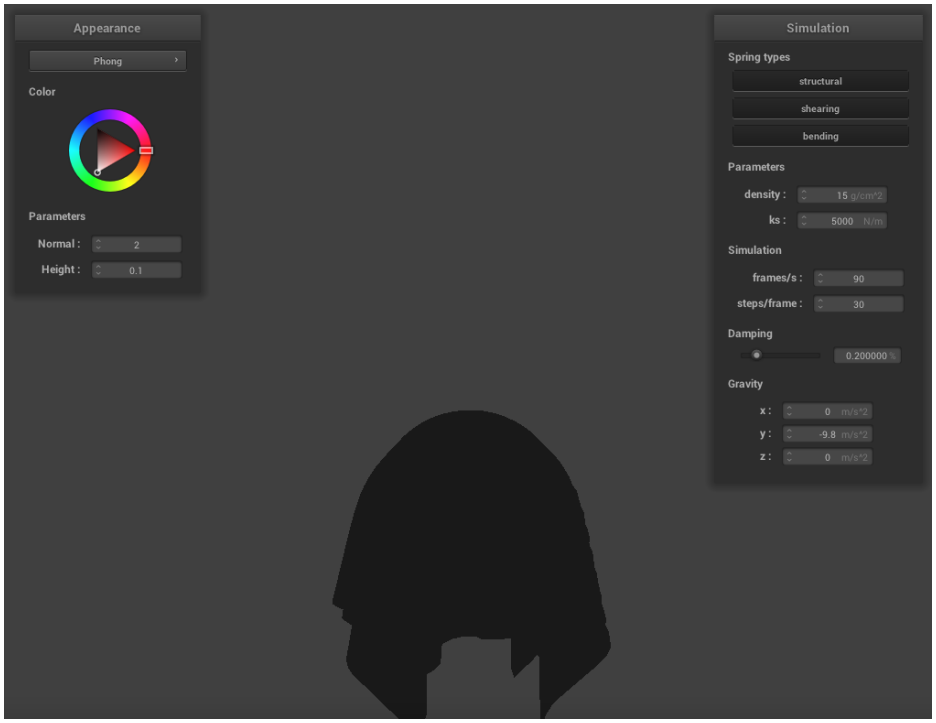
Explain the Blinn-Phong shading model in your own words.

Blinn-Phong shading model takes into account three types of lighting: diffuse lighting, ambient lighting, and specular lighting. It computes the final lighting by adding these three lightings together. Diffuse lighting represents the diffused light that can be seen no matter where the camera is (as long as the angle between the light and the normal

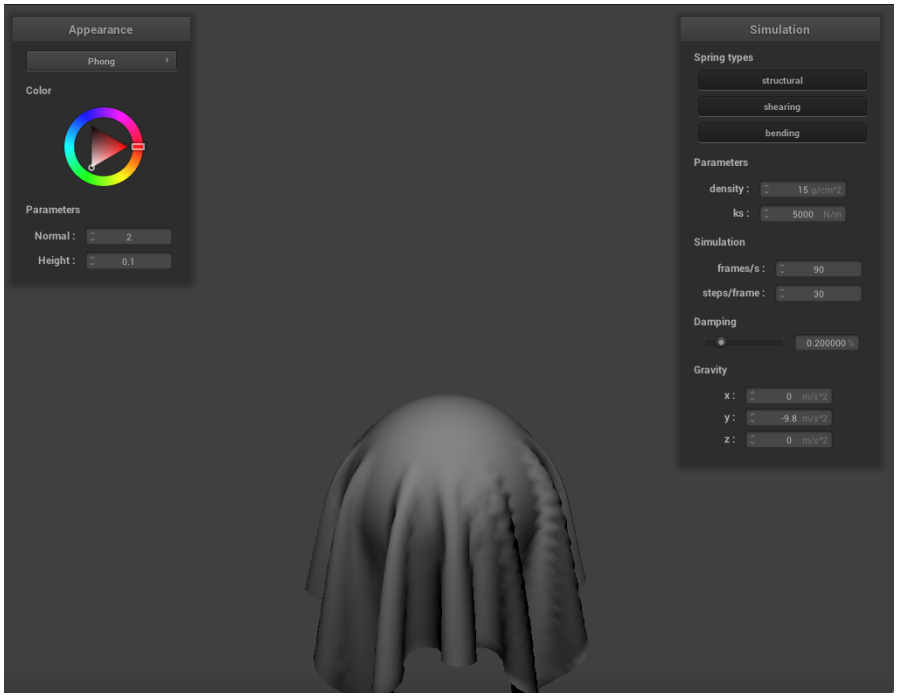
vector of the plane is smaller than 90 degrees). Specular lighting gives shiny effects only on certain areas because $cos(\alpha) = 1$ only when h and n overlap each other, in which case the angle between the camera view and the normal vector is the same as the angle between the incoming light and the normal vector. In all other cases, $cos(\alpha)$ would be smaller than 1 and therefore, with a very high p, it would make this term negligible. This explains why the shiny effects are “rare.” The ambient light gives the same amount of lighting to all vertices seen by the camera. Adding all these three effects gives the Blinn-Phong shader.

Show a screenshot of your Blinn-Phong shader outputting only the ambient component, a screen shot only outputting the diffuse component, a screen shot only outputting the specular component, and one using the entire Blinn-Phong model.

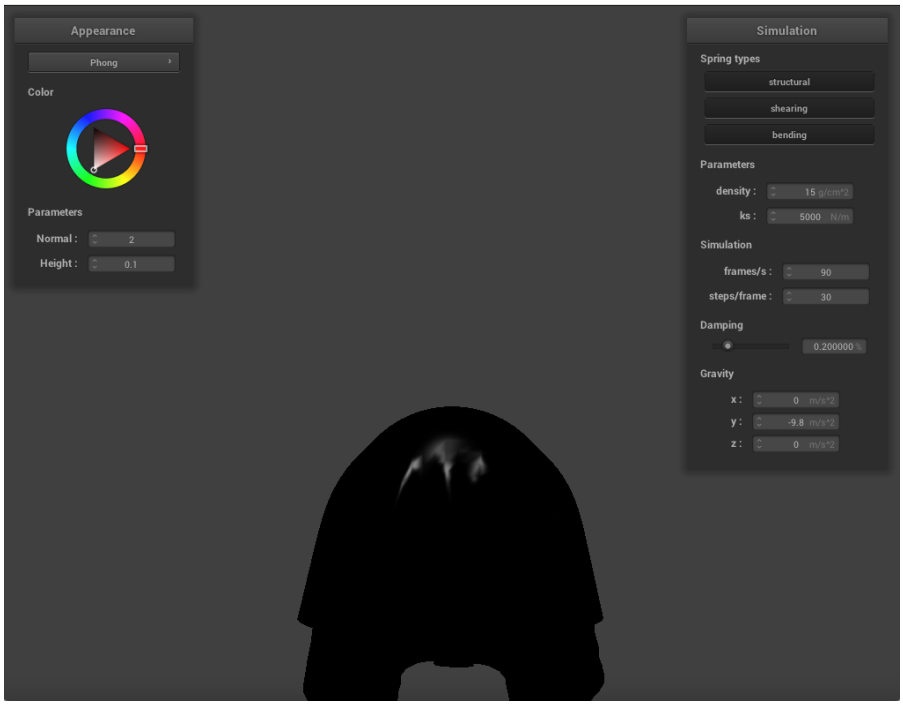
Ambient only:



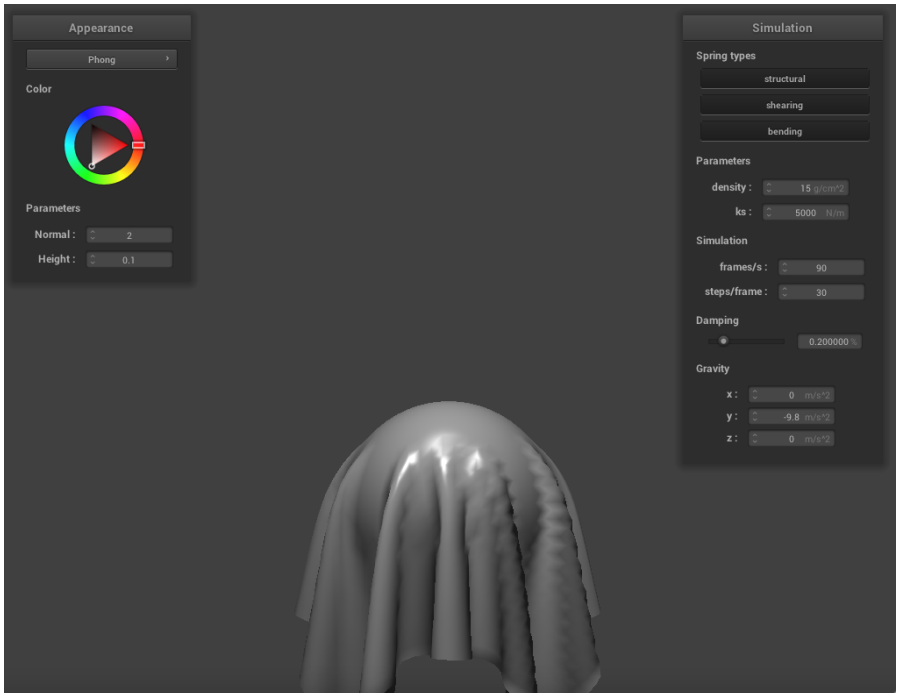
Diffuse only:



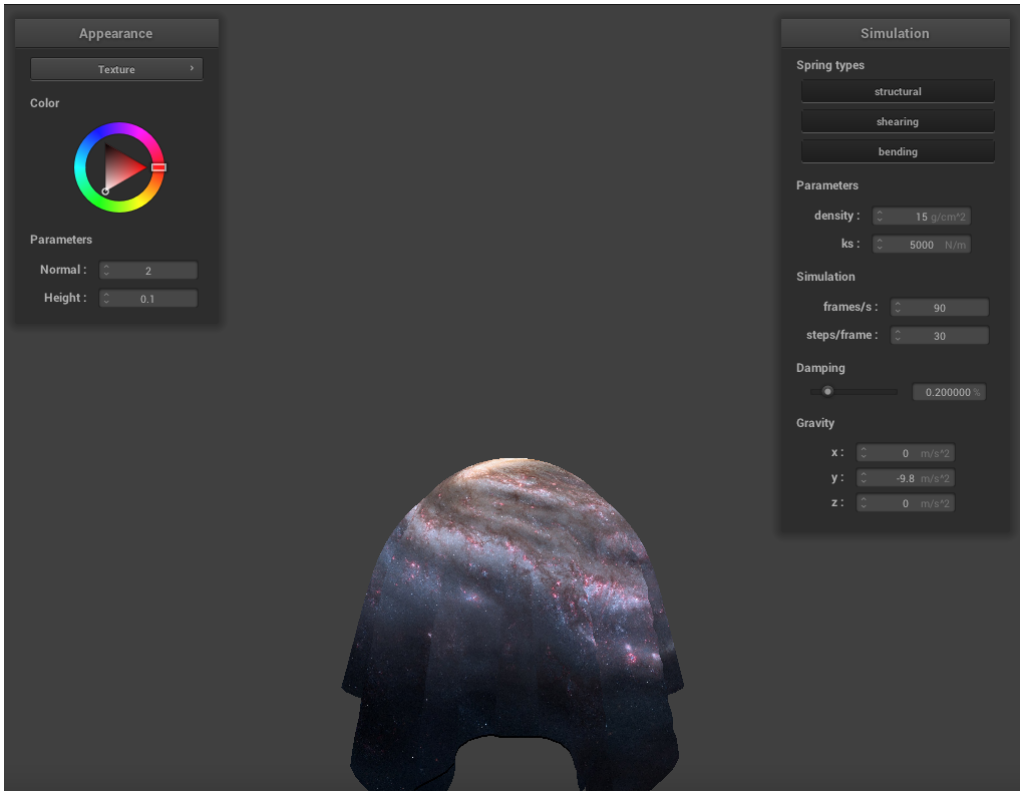
Specular only:



Entire Blinn-Phong model:



Texture mapping shader using the milky way texture!



Comparison between Bump and displacement:
In bump mapping, we interpolate the normal vectors by a small amount to change the amount of light received on that vertex, therefore creating the effects of a “bump” on the sphere and the cloth. This, however, doesn’t change the vertex position of the sphere or cloth. Displacement mapping not only interpolates the normal vector, but also changes the position of the vertices, displacing the geometry. This makes the sphere looks less “spherical.”

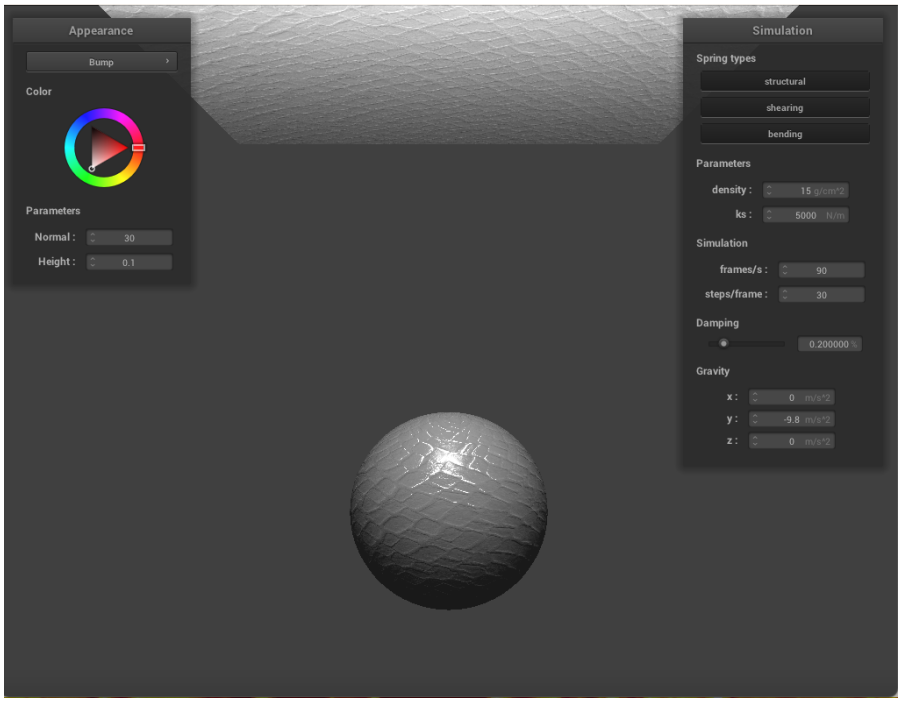
A note for our displacement mapping:
Our choice of coefficients for the Blinn-Phong isn’t that good. So in our displacement mapping, the bump effects, which are essentially changes in lighting, aren’t that clear when a small displacement occurs. But the bump is really there if you zoom in and observe it. We are really sorry about that!

Comparision of different coarseness:
Bump:
As shown in the image, we can see finer details of small bumps on the spheres and the cloth as we increased the resolution to 128x128. That is, the whole image looks less “coarse.”

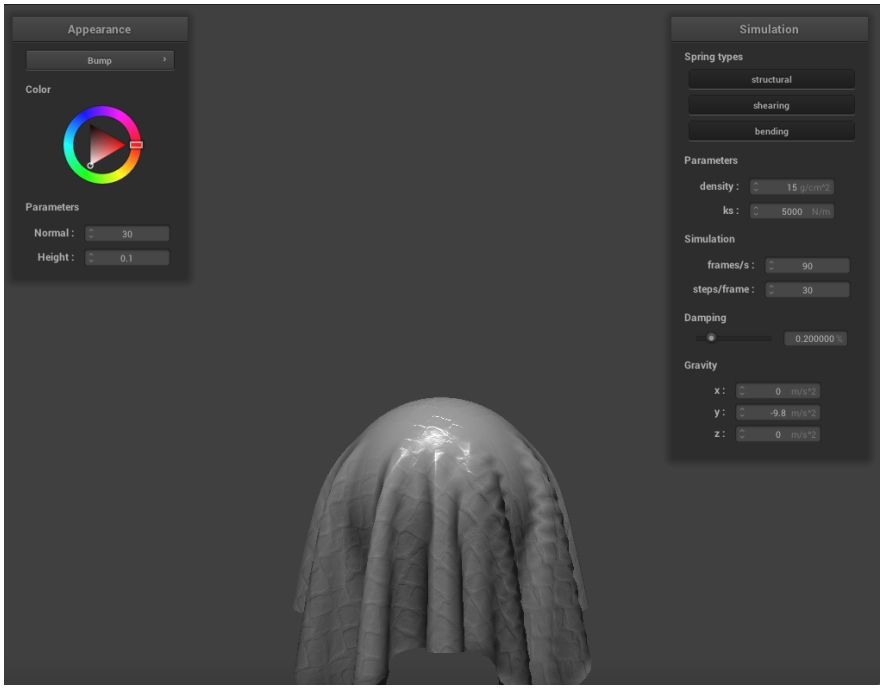
Displacement:
As we increase the resolution to 128x128, previous coarse large displacements on the sphere are replaced by a lot smaller displacements, making the whole image looking less “coarse” and more smooth!

Bump Mapping:

Default Setting
Sphere alone:

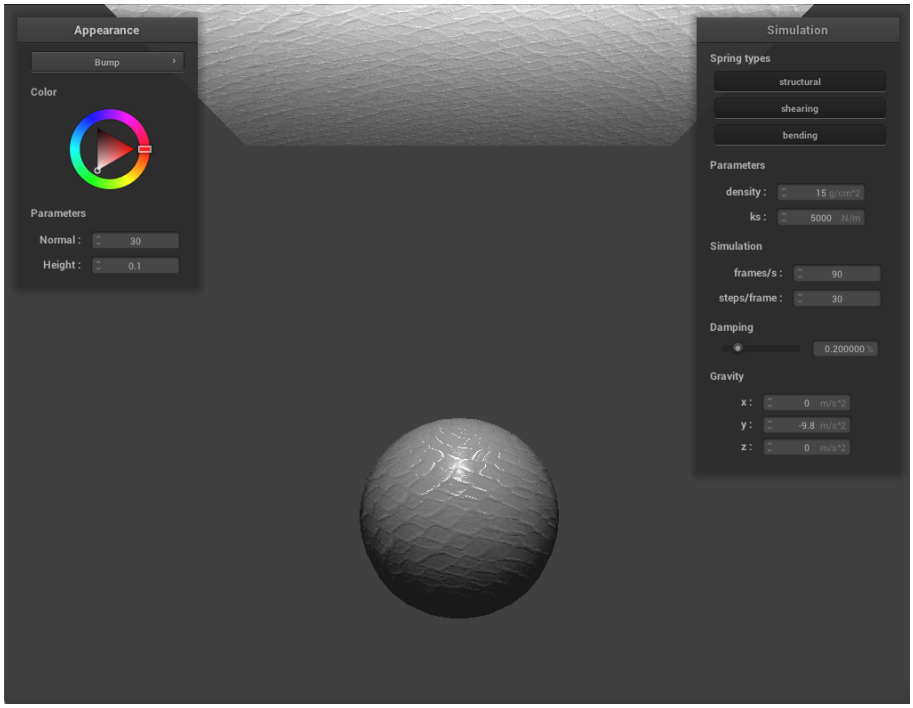


The cloth and the sphere:

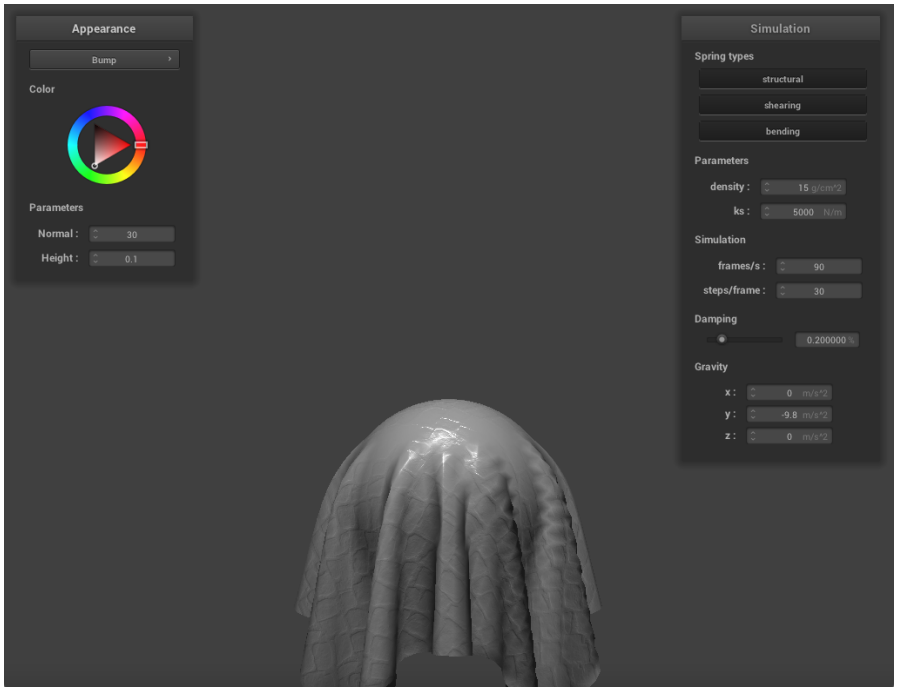


-o 16 -a 16:

Sphere alone:

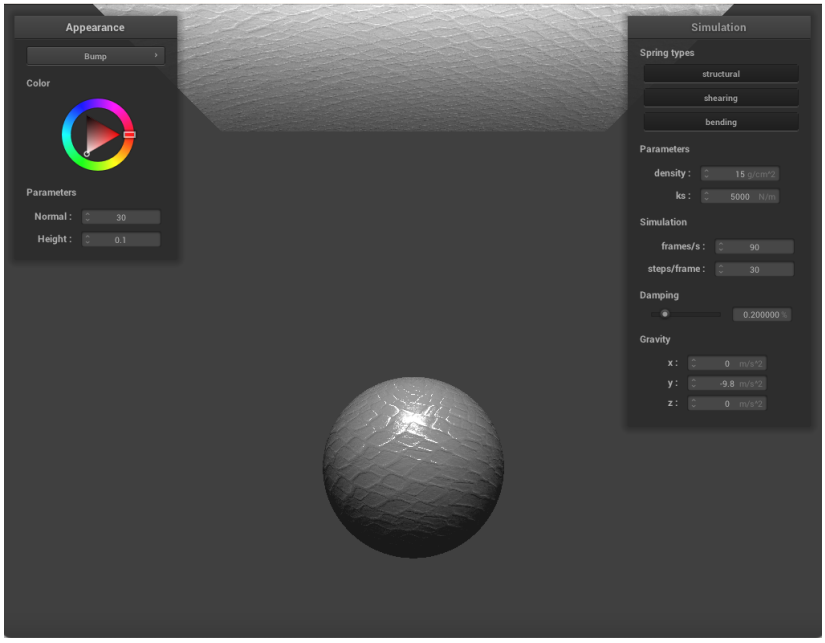


The sphere and the cloth:

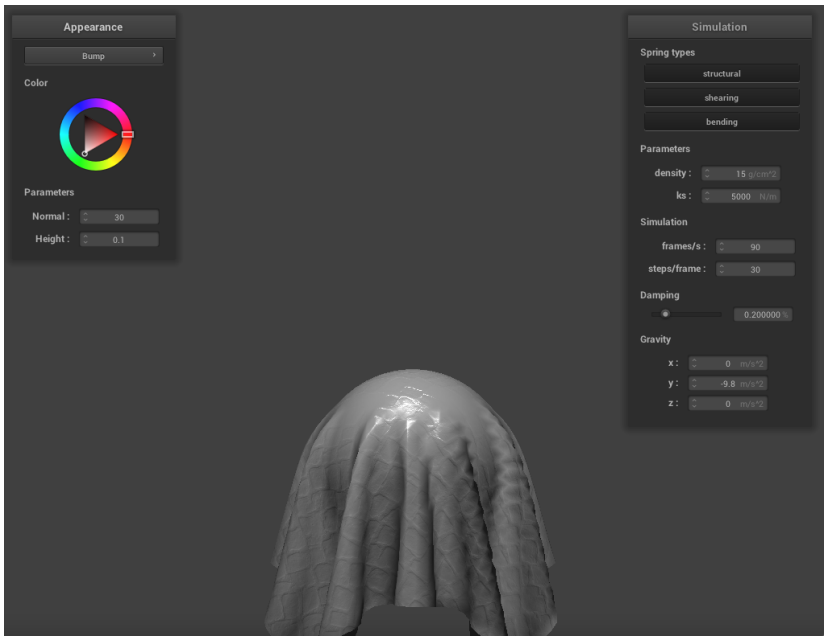


-o 128 -a 128:

Sphere alone:



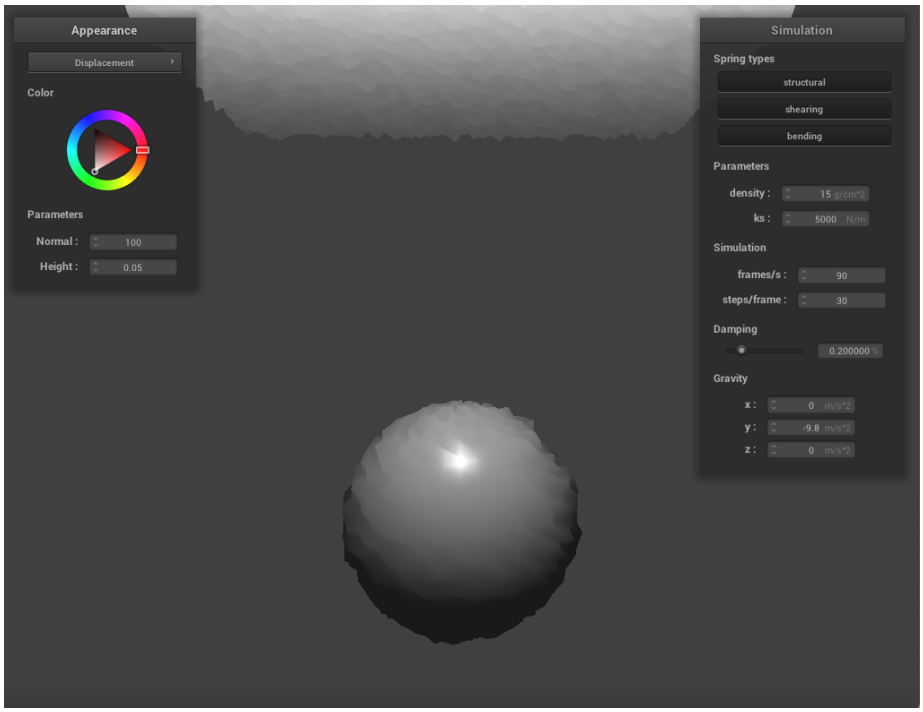
The sphere and the cloth:



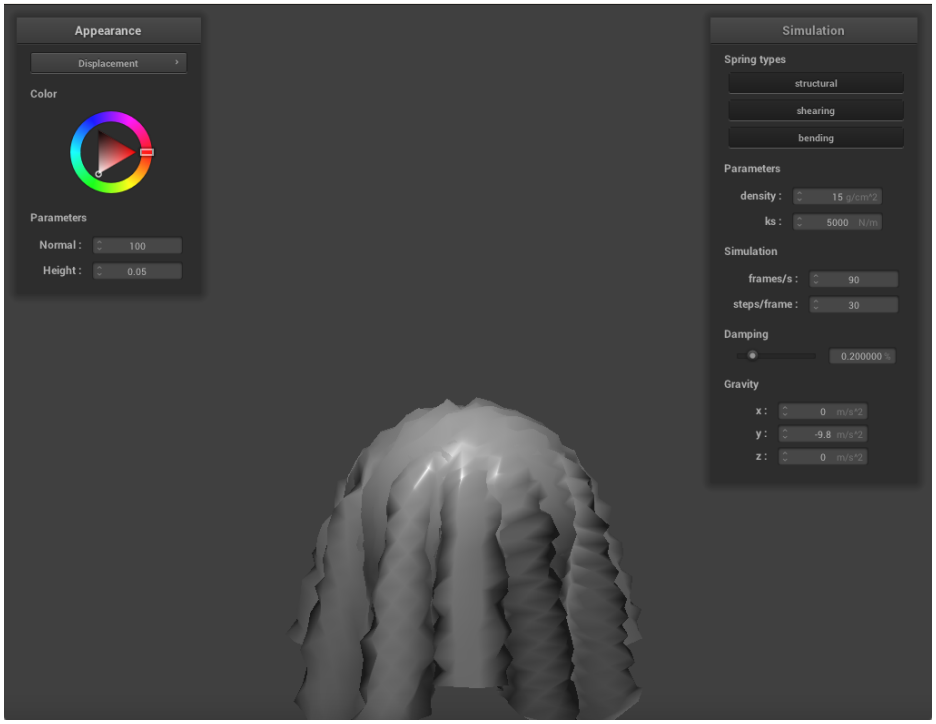
Displacement Mapping:

Default:

Sphere alone:

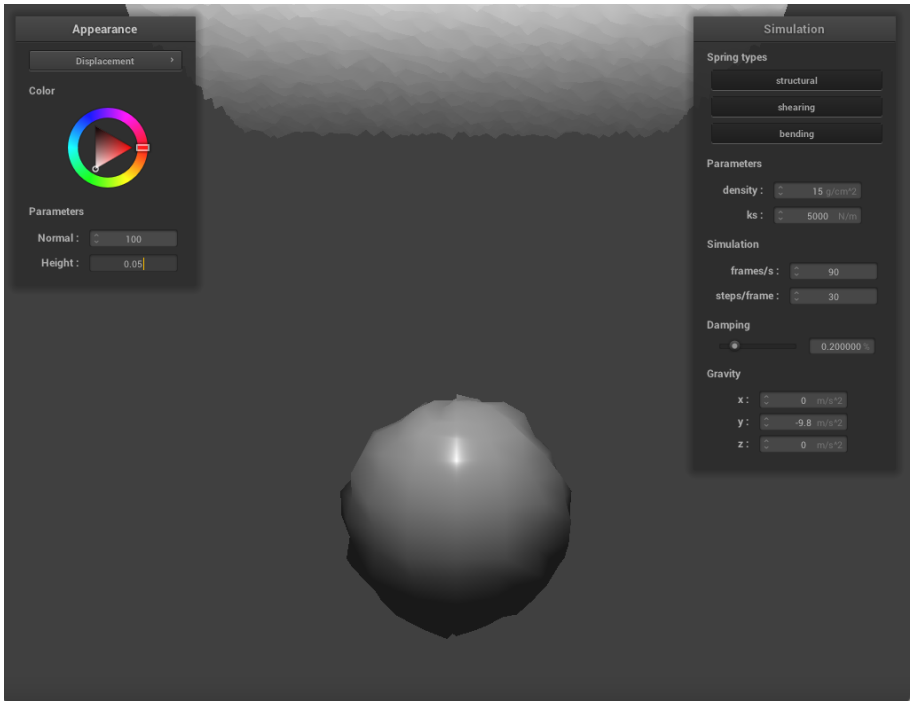


The sphere and the cloth:

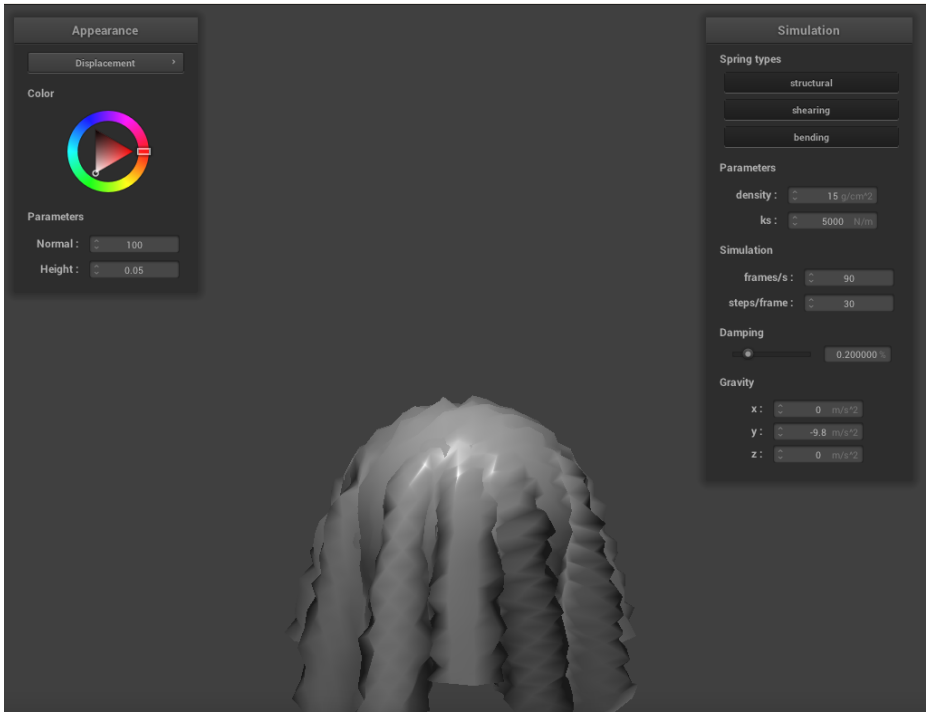


-o 16 -a 16:

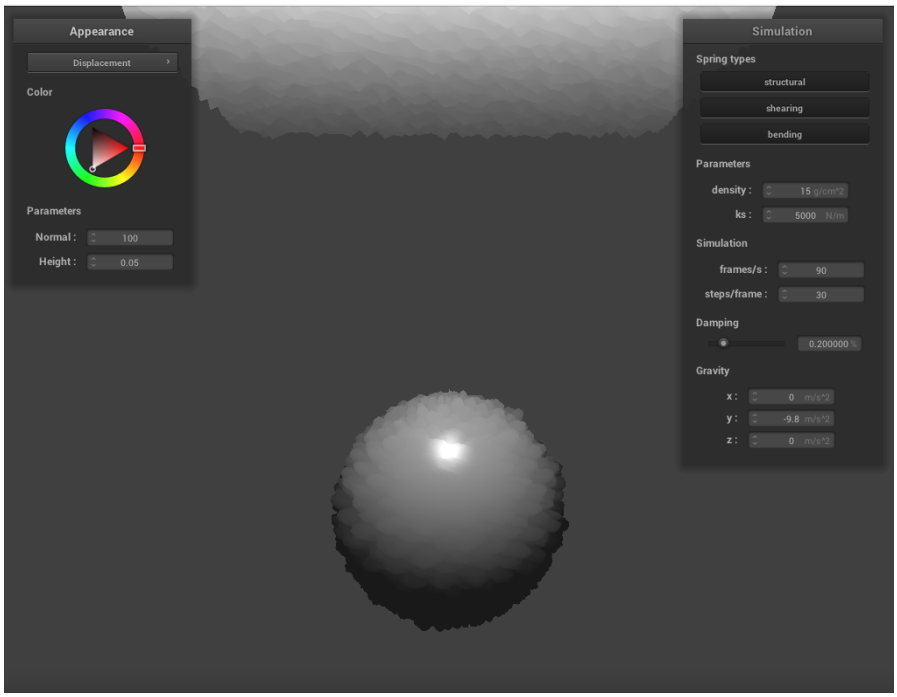
Sphere alone:

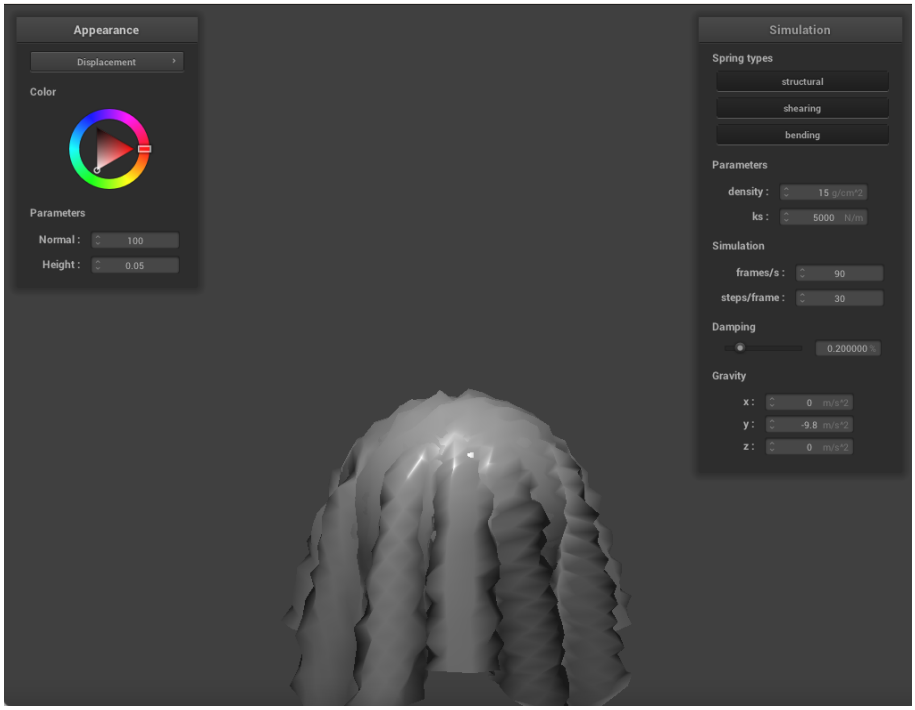


Sphere and the cloth:



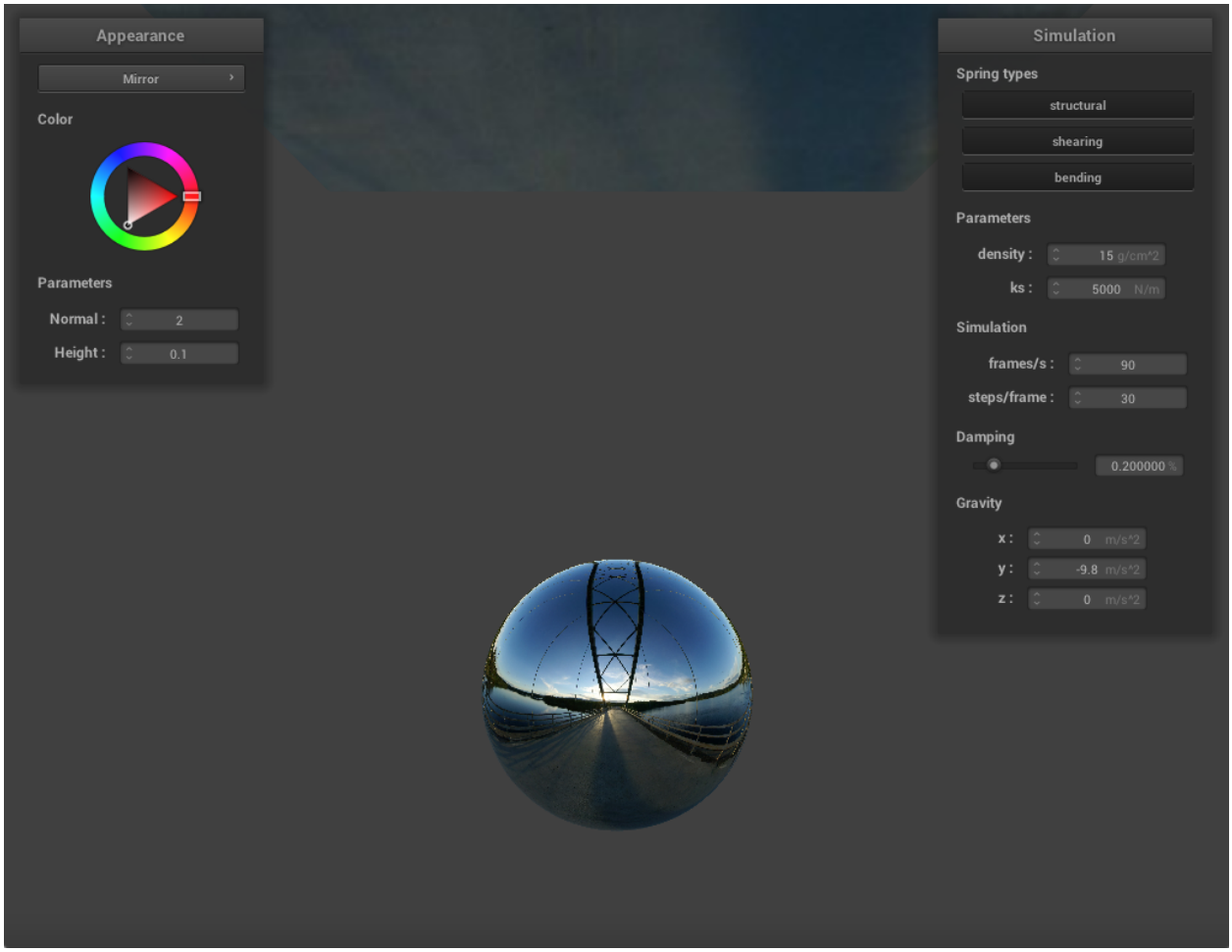
-o 128 -a 128:
Sphere alone:



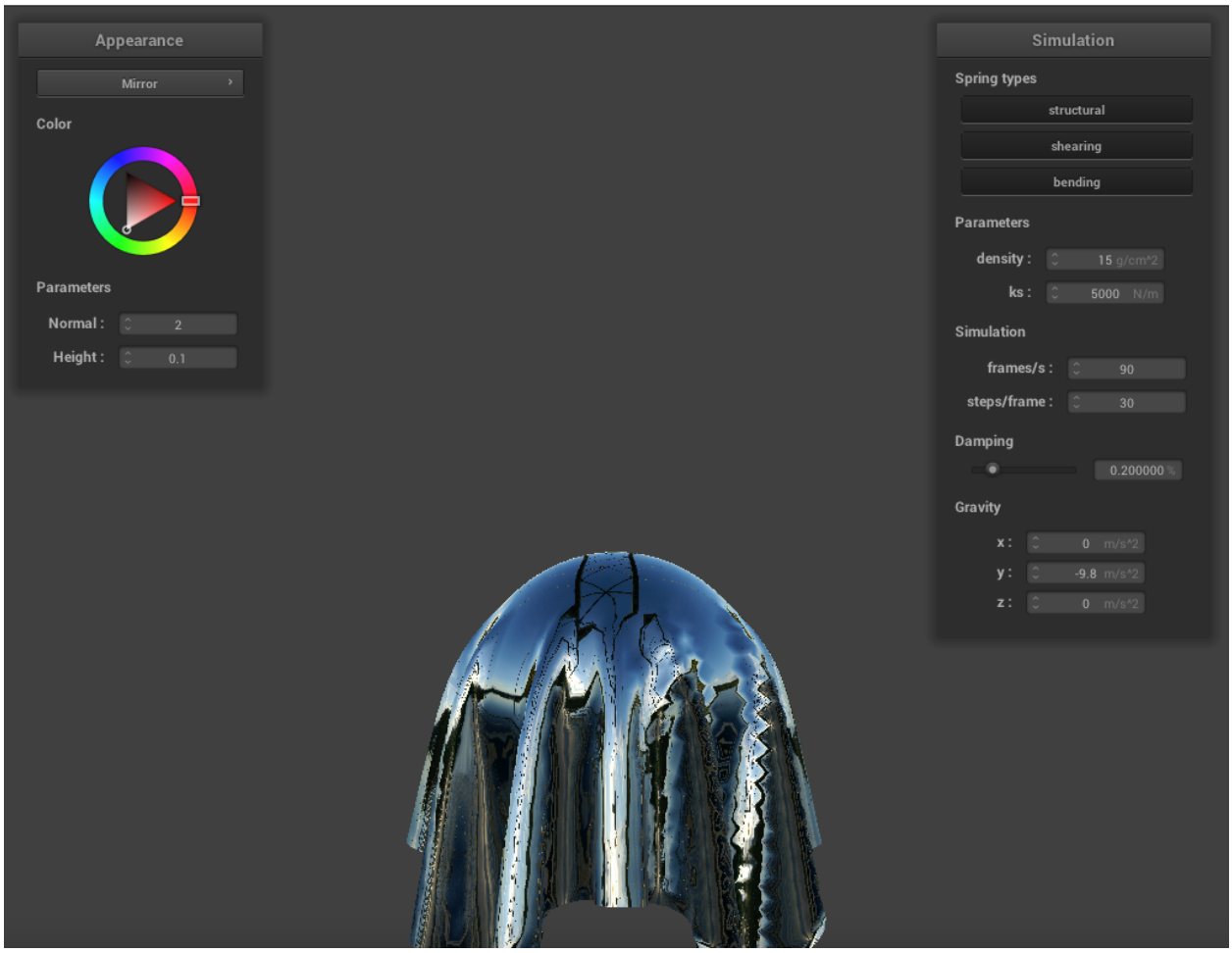


Show a screenshot of your mirror shader on the cloth and on the sphere.

Sphere:


Appearance
Mirror
Color
Parameters
Normal : 2
Height : 0.1
Simulation
Spring types
structural
shearing
bending
Parameters
density : 15 g/cm^2
ks : 5000 N/m
Simulation
frames/s : 90
steps/frame : 30
Damping
0.200000 %
Gravity
x : 0 m/s^2
y : -9.8 m/s^2
z : 0 m/s^2

Cloth:


Appearance
Mirror
Color
Parameters
Normal :
2
Height :
0.1
Simulation
Spring types
structural
shearing
bending
Parameters
density :
15 g/cm^2
ks :
5000 N/m
Simulation
frames/s :
90
steps/frame :
30
Damping
0.200000 %
Gravity
x :
0 m/s^2
y :
-9.8 m/s^2
z :
0 m/s^2