CS 184: Computer Graphics and Imaging, Spring 2022

Project 4: Cloth Simulator

Ethan Gnibus, CS184-antartica

Overview

Give a high-level overview of what you implemented in this project. Think about what you've built as a whole. Share your thoughts on what interesting things you've learned from completing the project.

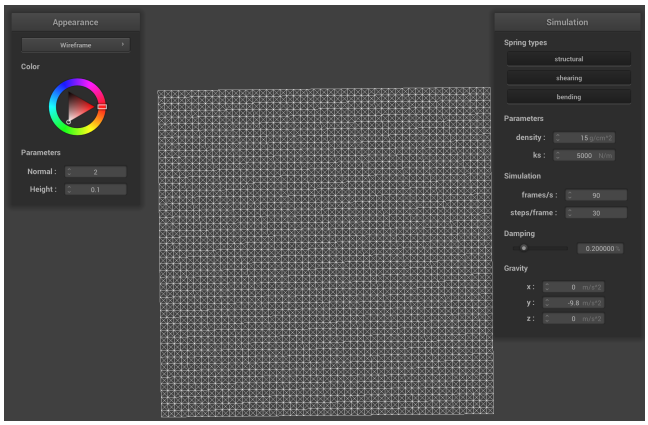
In this project, I implemented cloth simulation by following the PointMass and Spring method. This required building the PointMass-Spring grid itself, applying various forces to said PointMasses to simulate things like gravity, colliding the cloth with primitives such as spheres and planes, and colliding the cloth with itself. I also implemented GLSL Shaders that run in parallel on the GPU so that my simulation could look pretty without taking hours to render!

Part 1: Masses and springs

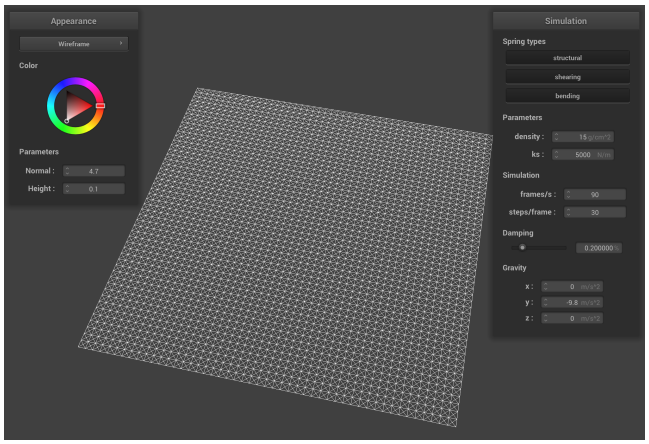
Describe what you did in Part 1. etc...

In Part 1, I wrote the code that represents a grid of masses and springs. To do this I first made num_width_points x num_height_points PointMasses, oriented horizontally or vertically them based on the orientation variable, then emplaced them to the back of the point_masses vector. Next, for every PointMass in the point_masses vector, I created it's structural, shearing, and bending springs and added them to the springs vector if they should exist.

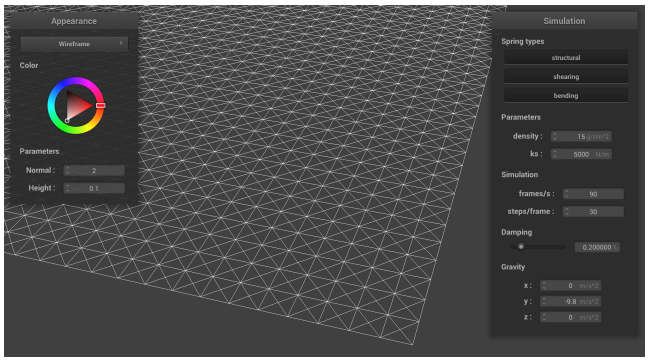
Take some screenshots of scene/pinned2.json from a viewing angle where you can clearly see the cloth wireframe to show the structure of your point masses and springs.



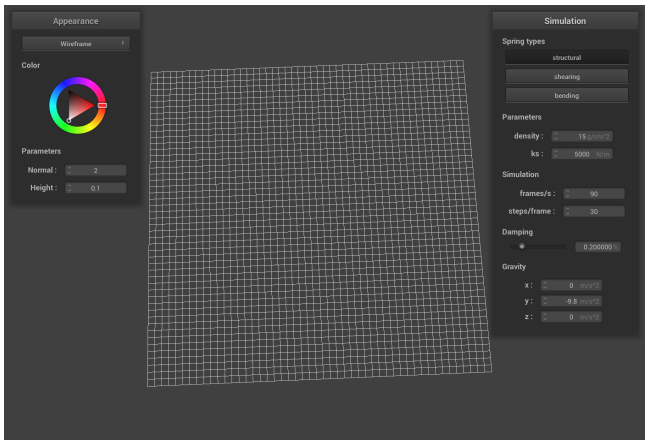
Bird's eye view of pinned2.



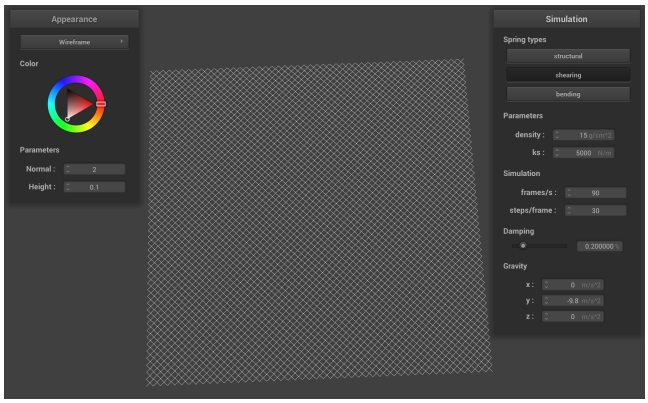
pinned2 viewed from an angle.



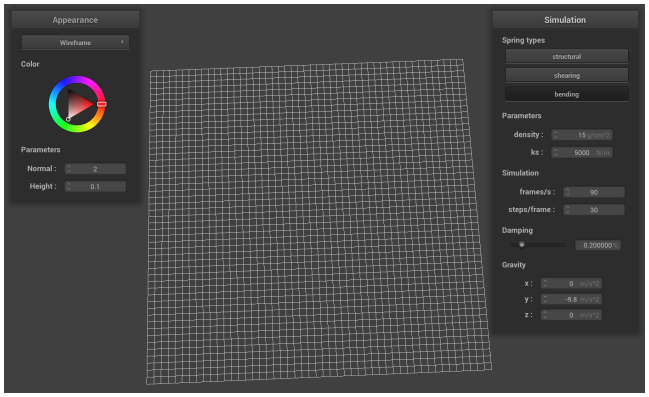
Zoomed in shot of pinned2.



pinned2 with only structural springs.

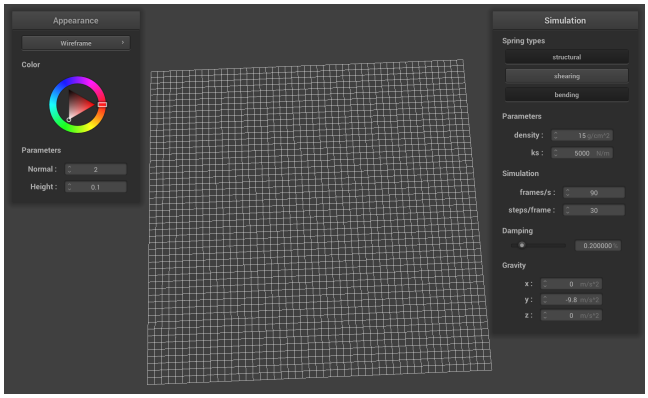


pinned2 with only shearing springs.

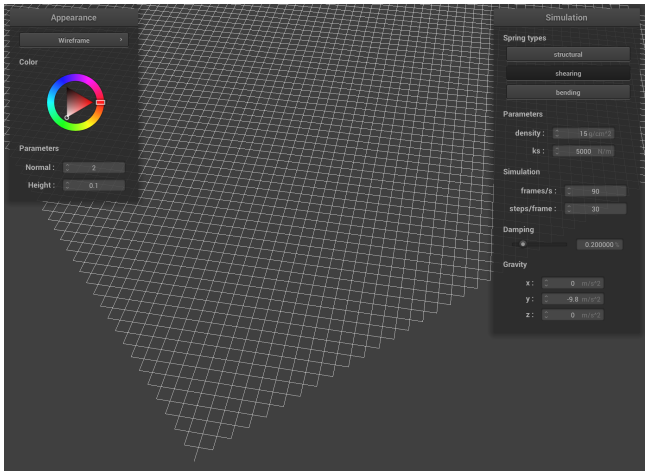


pinned2 with only bending springs.

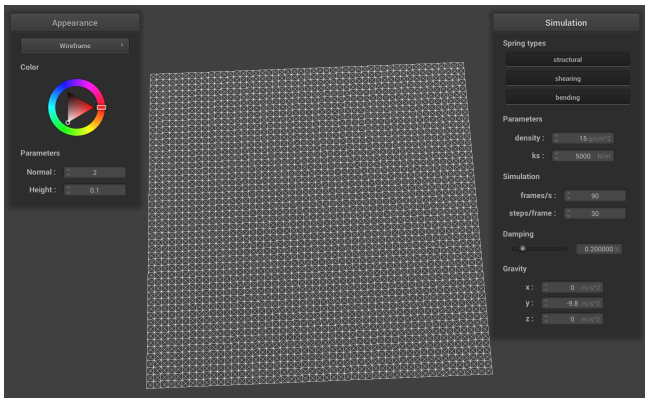
Show us what the wireframe looks like (1) without any shearing constraints, (2) with only shearing constraints, and (3) with all constraints.



(1) without any shearing constraints.



(2) with only shearing constraints.



(3) with all constraints.

Part 2: Simulation via numerical integration

Describe what you did in Part 2. etc...

In Part 2 I simulated external and spring correction forces to act on the PointMasses that make up our cloth. I then used Verlet integration to find the new positions of the PointMasses after the forces are applied to them. To better mimic real world behavior, I limited how far each spring could stretch by constraining the distance each pair of connected PointMasses could be apart from one another.

Experiment with some the parameters in the simulation. To do so, pause the simulation at the start with P, modify the values of interest, and then resume by pressing P again. You can also restart the simulation at any time from the cloth's starting position by pressing R.

Describe the effects of changing the spring constant ks; how does the cloth behave from start to rest with a very low ks? A high ks?

With a low ks (less than 100) the cloth seems to vibrate after swiftly falling to a resting position. With a high ks (above 10000000) the cloth falls to a resting position very rigidly. While in a resting position, it is much less flexible than the sub 100 or 5000 counterparts. It seems that the cloth goes from a simulating a free-flowing and light material like netting to a dense and stiff material such as construction paper and beyond as ks goes from low values to high values.

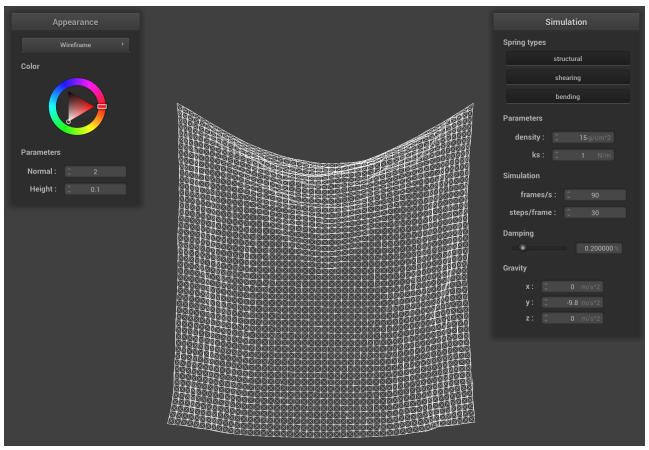
What about for density?

A low density value like 1 seems to reflect a material that doesn't weigh much like netting. It holds it's top edge at almost a straight line, which means that there is not much weight pulling the material down between the pinned PointMasses. On the other hand, a high density value such as 10000 visually looks very heavy as if it could be simulating chainmail. The top edge sinks down about an eighth of the way down the entire mesh! We can also notice that PointMasses jiggle around unlike the low density case. This tells us that there is lots of stress on the springs when there is high density. We can conclude that as density goes from low to high, the material simulated goes from one that doesn't weigh much to one that weighs a lot.

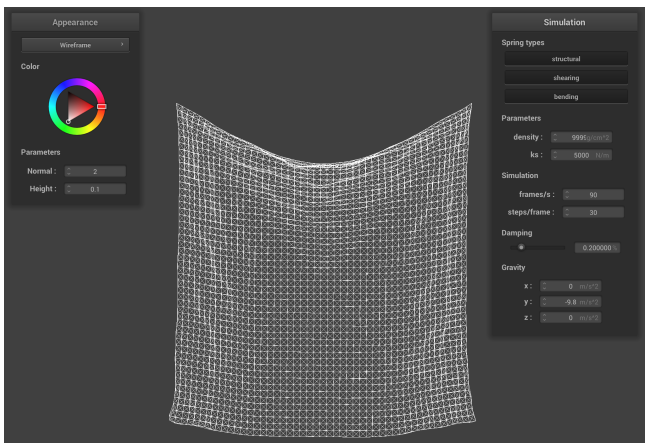
What about for damping?

In our simulation, low values of damping like 0 appear to accerate the cloth's movement while higher values of damping like 1 appear to slow the cloth's movement down. From lecture we know that this is a product of a flaw in our simulation approach, so we can discard that feature and look to the others. At low values of damping, we can observe that the cloth swings about rather than falling to it's resting position. It is wavy and jittery almost like a sheet drying in the wind. As the damping factor increases, we can observe the cloth rest to a halt. Replaying the simulation with a damping factor of 1 shows the cloth falling down to it's resting position and immediately stopping. It appears that the damping factor controls how powerful springs in our model are at constraining the movement of the PointMasses. Low values of damping mean that springs have little control over the forces exerted on the cloth other than holding the cloth together. As damping values rise, springs get better and better at making PointMasses less affected by other forces.

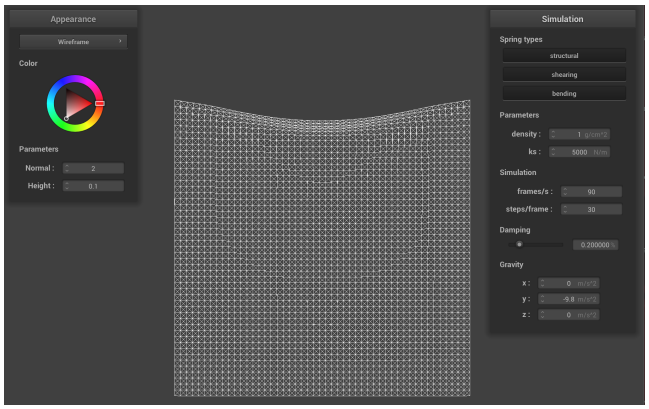
For each of the above, observe any noticeable differences in the cloth compared to the default parameters and show us some screenshots of those interesting differences and describe when they occur.



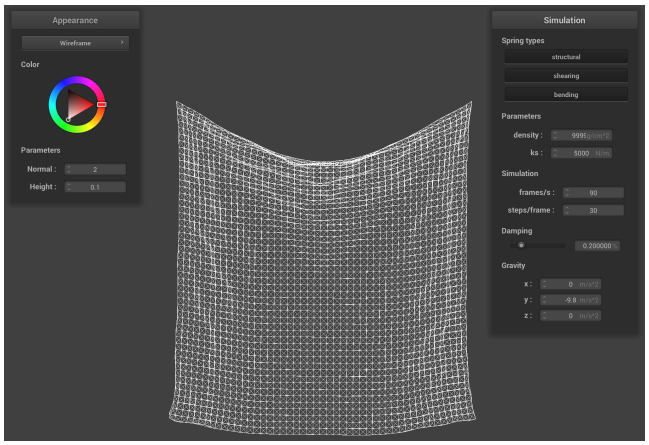
When ks = 1 N/m, the material simulated is wiggly like a waterbed.



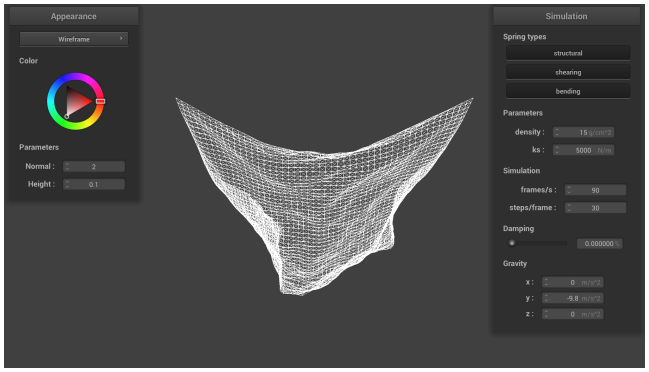
When ks = 100000 N/m, the material simulated is stiff like construction paper.



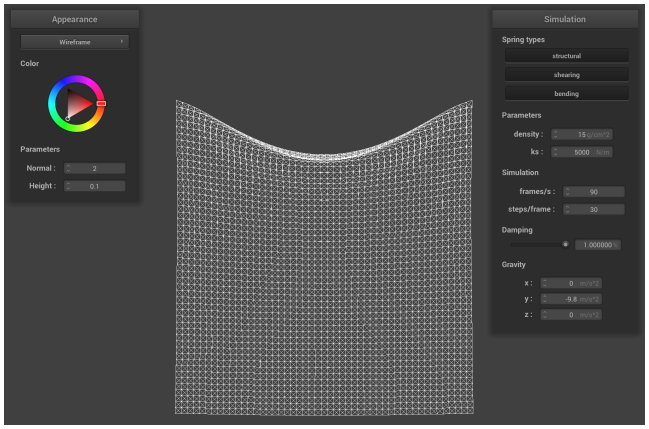
When density = 1 gm/cm^2, the material simulated is light like netting.



When density = 99999 gm/cm^2, the material simulated is heavy like like chainmail but jittery as if it's under lots of stress.



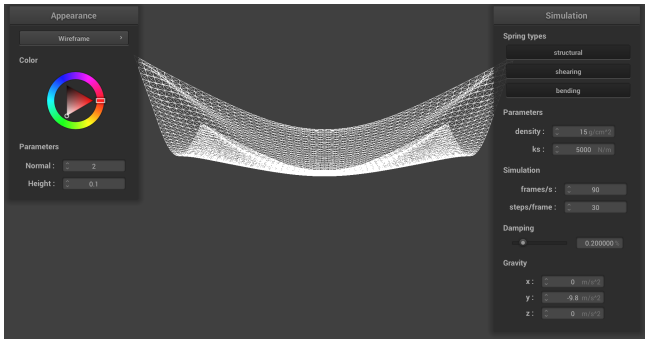
When damping = 0, the cloth swings rather than going to a resting position.



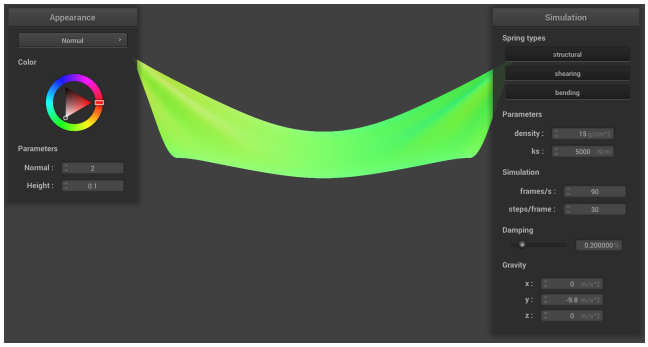
When damping = 1, the cloth falls to its resting position and sits still almost immediately after it stops falling.

Show us a screenshot of your shaded cloth from scene/pinned4.json in its final resting state! If you choose to use different parameters than the default ones, please list them.

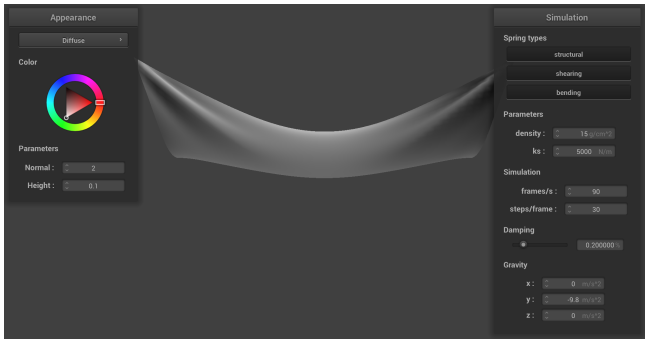
Below I show shaded versions of scene/pinned4.json in it's final resting state with default parameters.



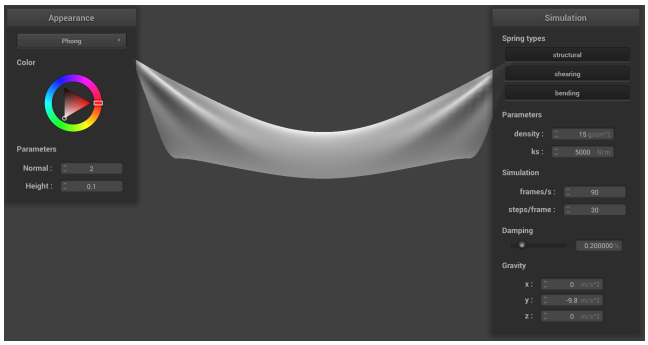
Wireframe.



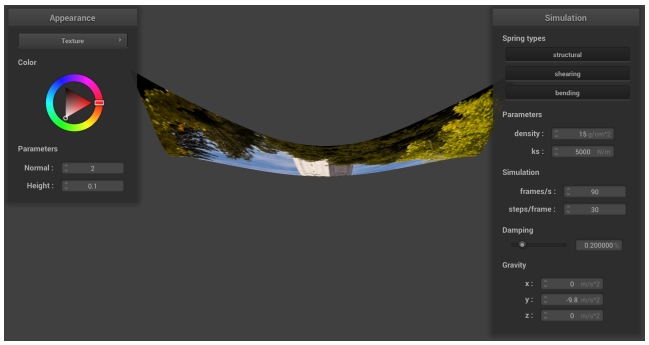
Normal Shading.



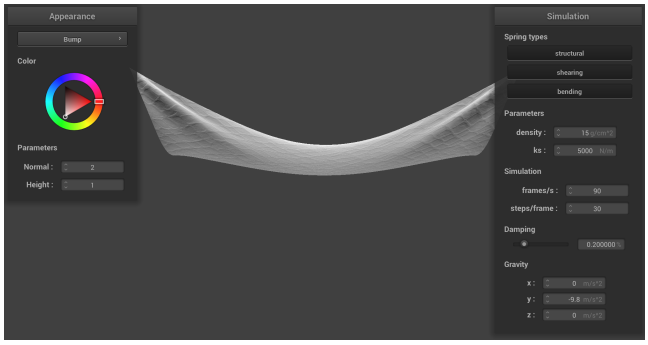
Diffuse Shading.



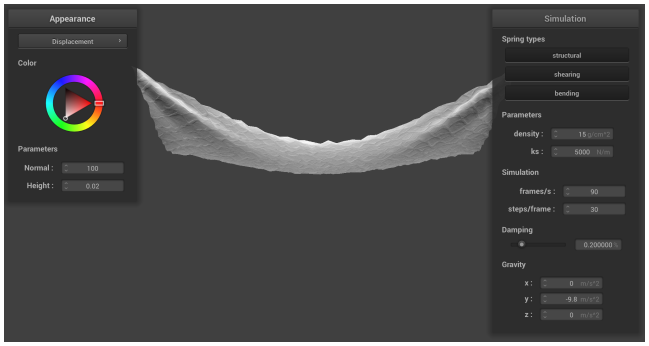
Blinn-Phong Shading.



Texture Shading.



Bump Shading.



Displacement Shading.

Mirror Shading.

Custom Shading (Cell Shader).

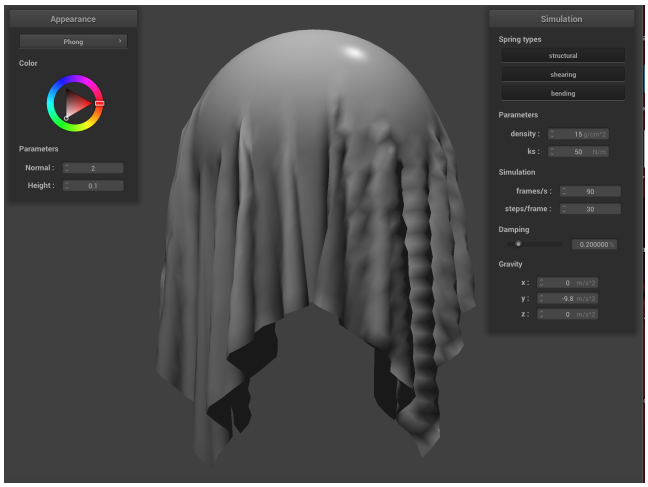
Part 3: Handling collisions with other objects

Describe what you did in Part 3. etc...

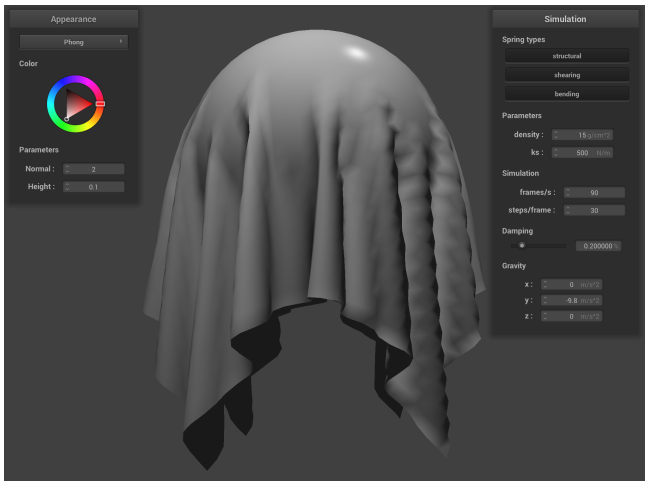
In Part 3, I added collision to my cloth simulation. First, I added collision with spheres by bumping a PointMass a little above the sphere if it ever crosses over the sphere's surface or rests on the surface. I also implemented collisions with planes by doing the same if it a PointMass crosses from one side of the plane's surface or rests on the surface. Both of these involved finding the intersection point, finding an offset point a little on the outside of the object's intersection point, and applying a correction vector scaled by friction to correct the every PointMass's position that intersects.

Show us screenshots of your shaded cloth from scene/sphere.json in its final resting state on the sphere using the default ks = 5000 as well as with ks = 500 and ks = 50000. Describe the differences in the results.

I simulated the following with Blinn-Phong Shading.

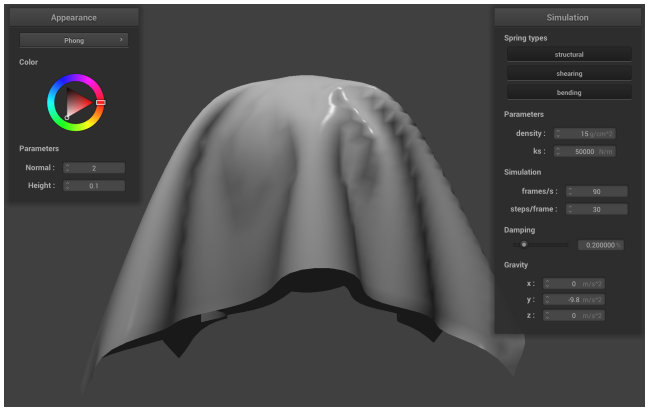


ks = 50.



ks = 500.

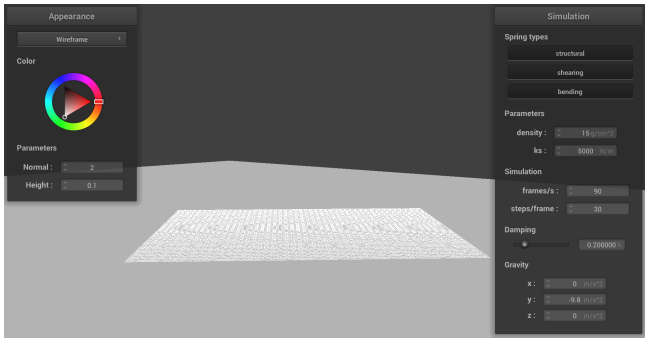
ks = 5000.



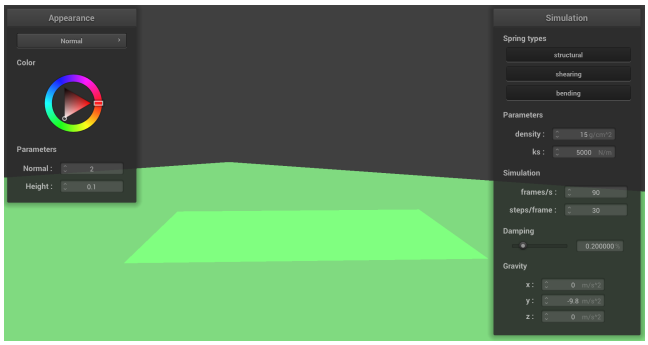
ks = 50000.

As ks goes from a low value to a high value, the cloth seems to go from simulating a thin material to simulating a thick material. At lower values, more folds appear and the cloth falls into place more swiftly. At higher values, less folds appear and the cloth slowly pulls itself into place. When ks = 50, the cloth looks like it is simulating a bandana. When ks = 500 the cloth looks like it is simulating a tablecloth. When ks = 5000, the cloth looks like it is simulating a towel. When ks = 50000, the cloth looks like it is simulating a sheet of rubber.

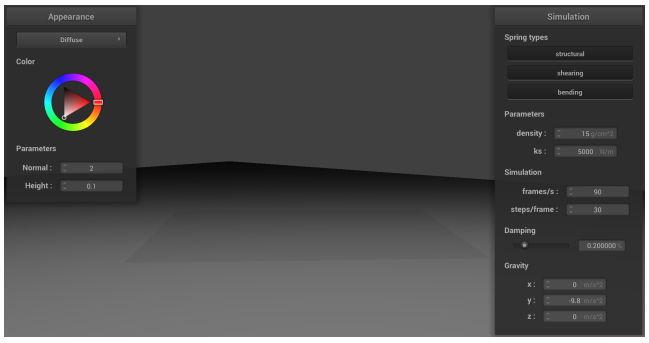
Show us a screenshot of your shaded cloth lying peacefully at rest on the plane. If you haven't by now, feel free to express your colorful creativity with the cloth! (You will need to complete the shaders portion first to show custom colors.)



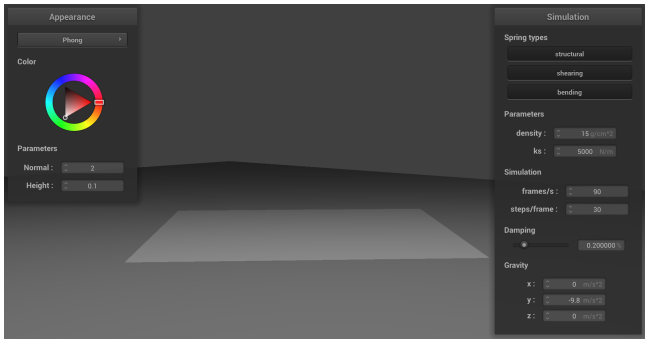
Wireframe.



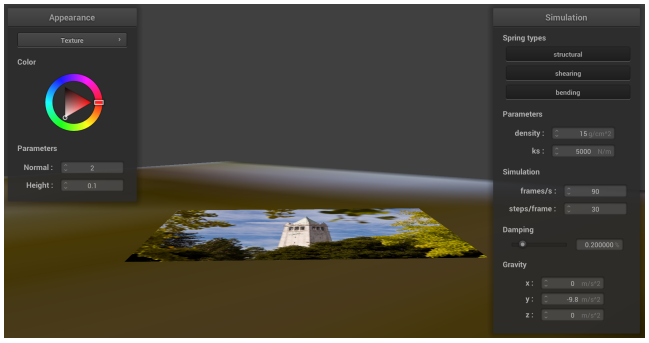
Normal Shading.



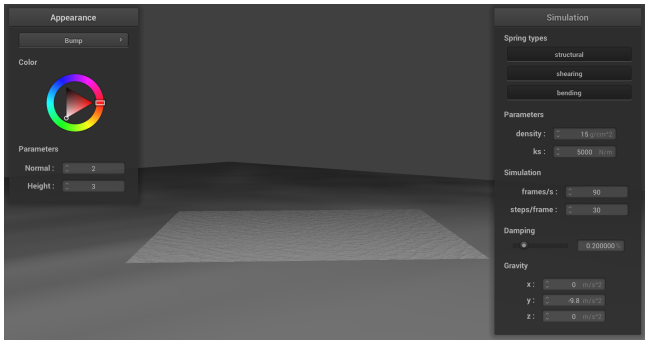
Diffuse Shading.



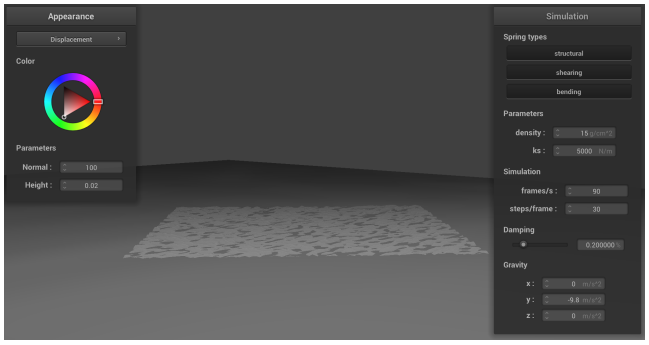
Blinn-Phong Shading.



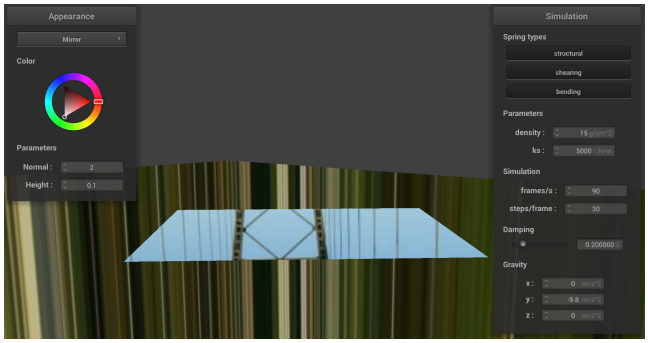
Texture Shading.



Bump Shading.



Displacement Shading.



Mirror Shading.

Custom Shading (Cell Shader).

Displacement shading looks weird because the displaced plane vectors clip into the mesh's vertices or vice versa. Cell shading (my extra credit shader) looks weird because the light hitting the flat plane and the flat cloth fall into the same bucket.

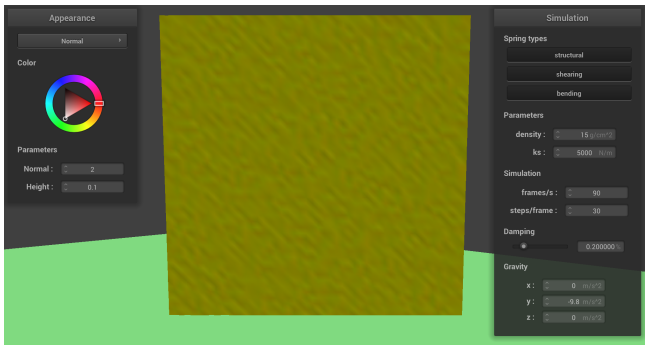
Part 4: Handling self-collisions

Describe what you did in Part 4. etc...

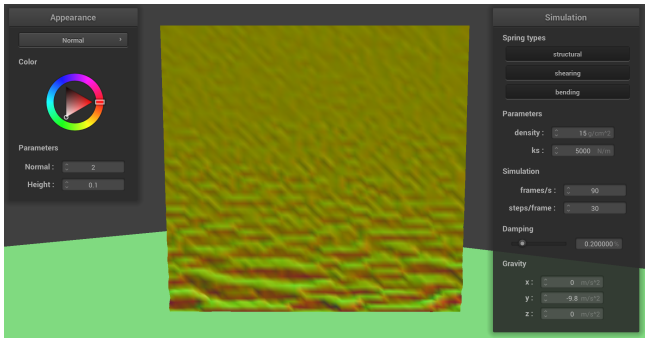
In Part 4 I created and accelerated collisions between the cloth and itself. To do this, I assigned PointMasses hashes based on their position in the scene so that PointMasses with the same hash are in the same partition of the scene space. Then I used this hash to put PointMasses that are close in proximity into the same bin in a spatial map. Then I loop through all PointMasses, see if the current PointMass collides with any other PointMass in it's spatial map bin, and apply a correcting force to it if it does collide.

Show us at least 3 screenshots that document how your cloth falls and folds on itself, starting with an early, initial self-collision and ending with the cloth at a more restful state (even if it is still slightly bouncy on the ground).

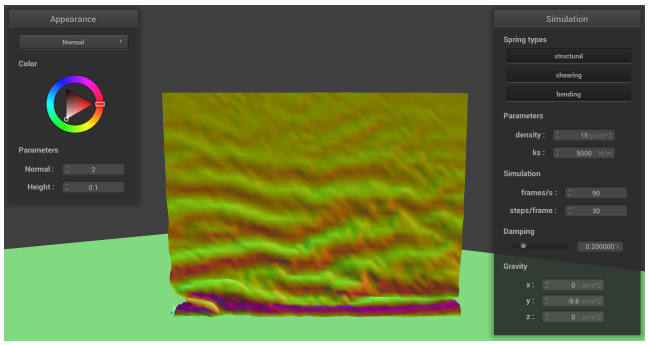
The following images were simulated using normal shading.



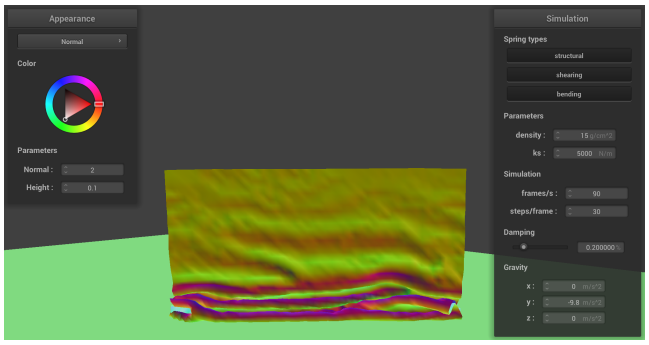
Step 1.



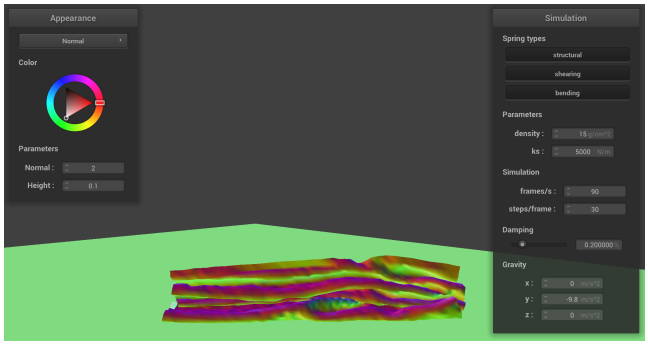
Step 2.



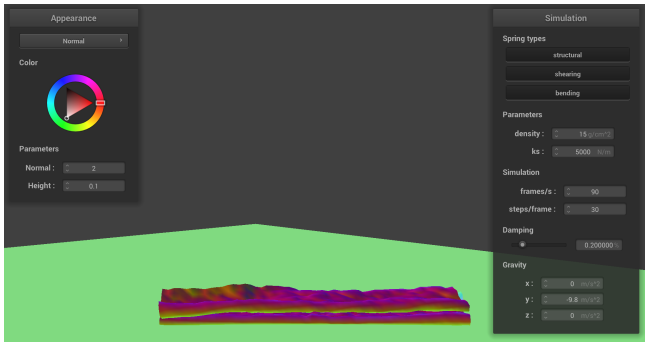
Step 3.



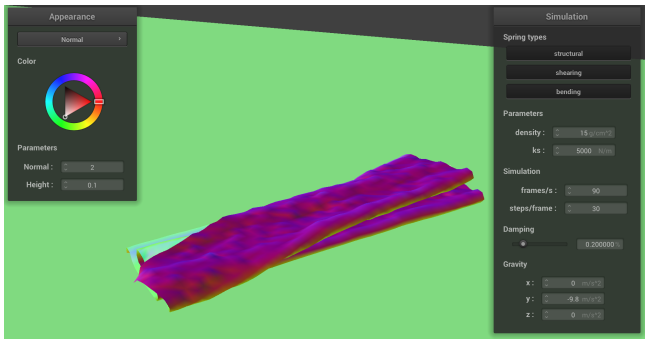
Step 4.



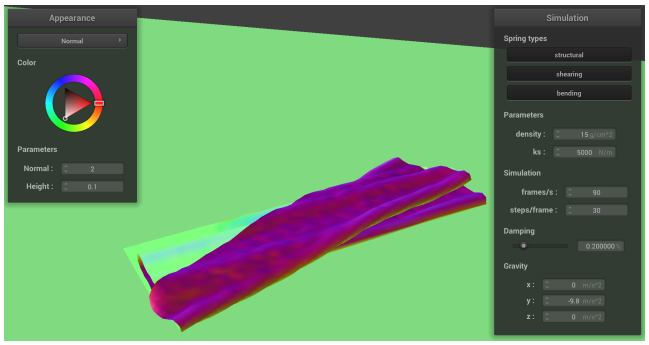
Step 5.



Step 6.



Step 7.

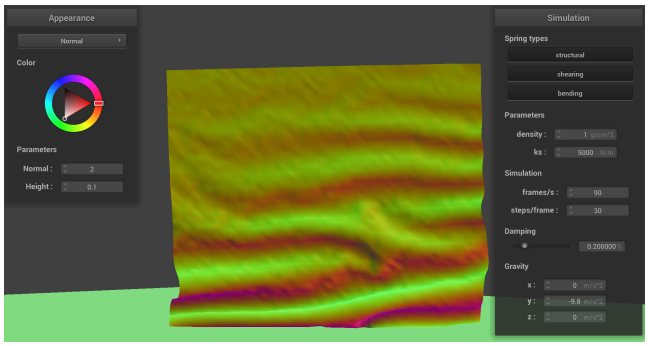


Step 8.

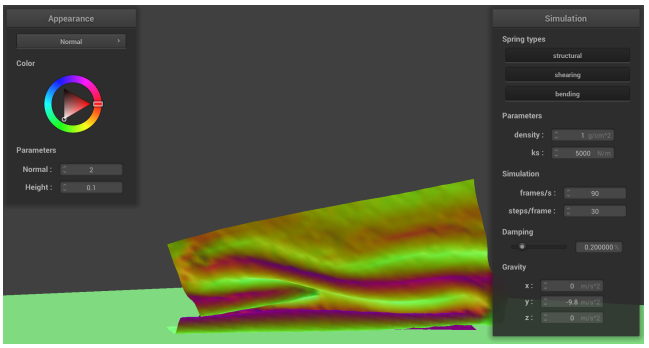
As observed, the cloth folds over itself nicely.

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.

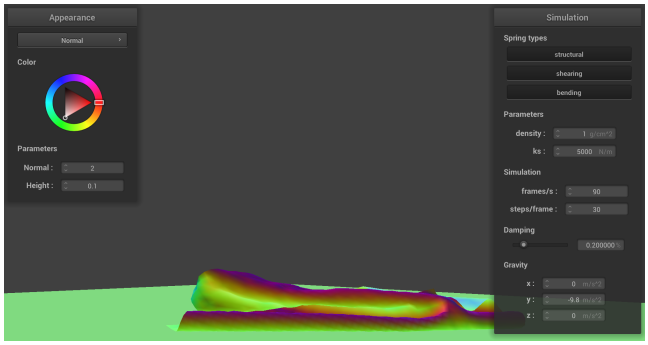
Below I will describe self collisions with low density (1 g/cm^2) using normal shading.



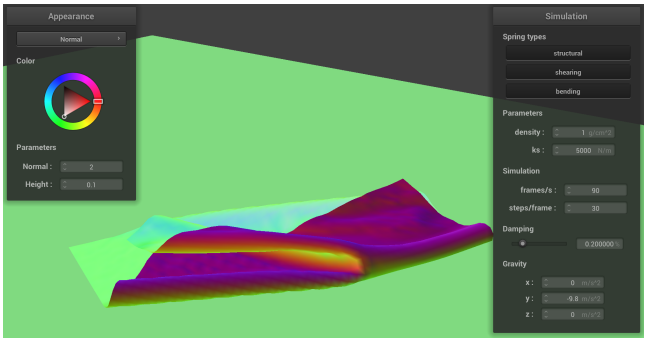
Low density causes the cloth to fall in a way that makes big ripples.



It folds back an forth over itself like lasagna.

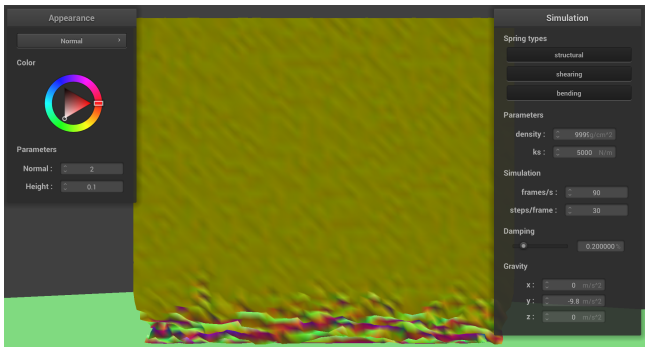


When it falls it is relatively still and moves smoothly.

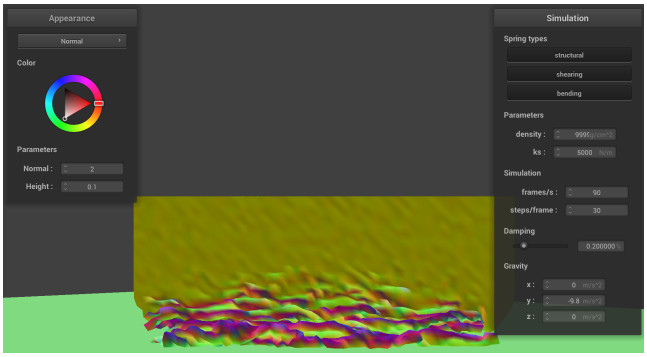


It rests in a sleek, smooth way like a wet paper towel.

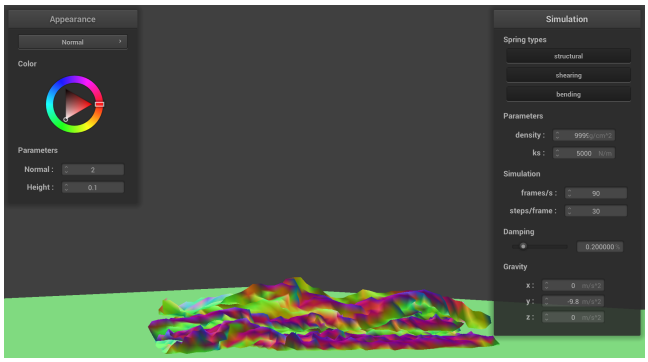
Below I will describe self collisions with high density (99999 g/cm^2) using normal shading.



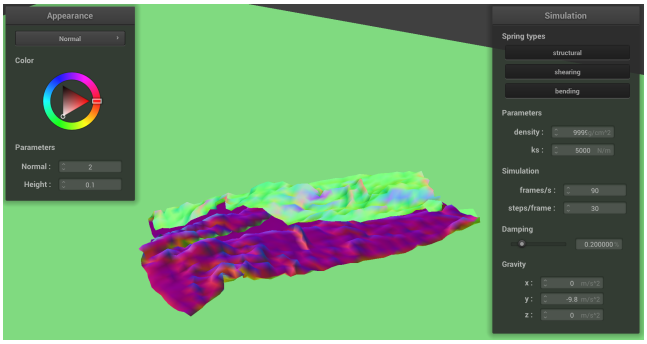
High density causes the cloth to crash on itself and make a bunch of tiny folds.



It clumps up like dropping chainmail.

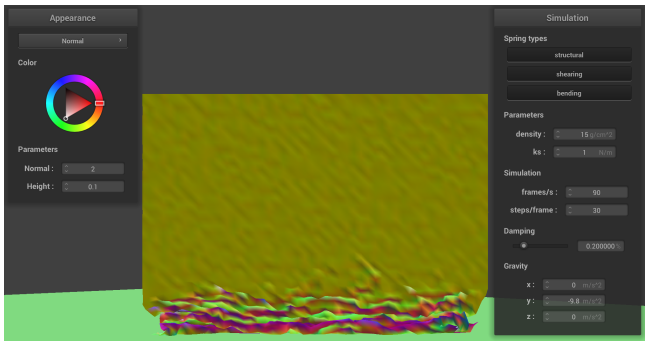


When it falls it lays to rest in a big pile.



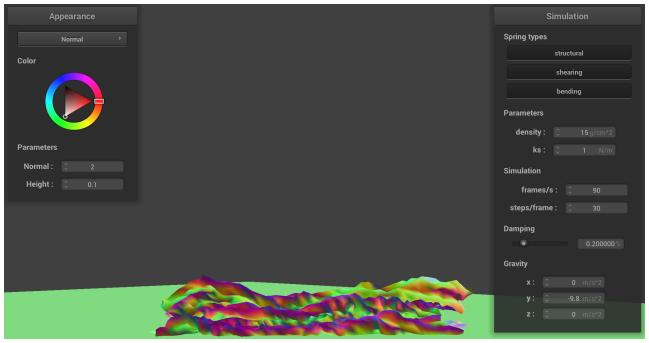
It eventually spreads out more, but has tiny ripples that are tight-knit.

Below I will describe self collisions with ks = 1 using normal shading.

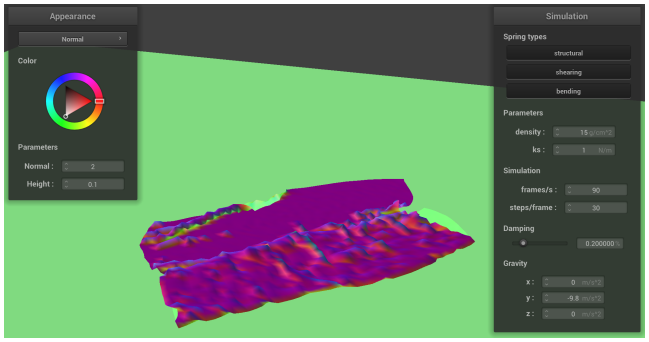


Like above, a low ks makes the cloth fold into tiny ripples.

This time they are more evenly dispersed rather than clumped around the center of mass.

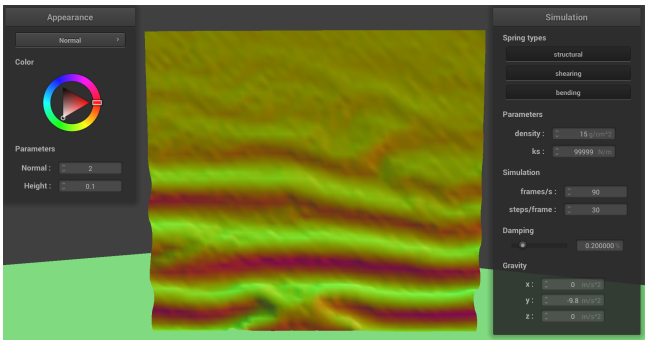


The cloth falls into a springy pile.

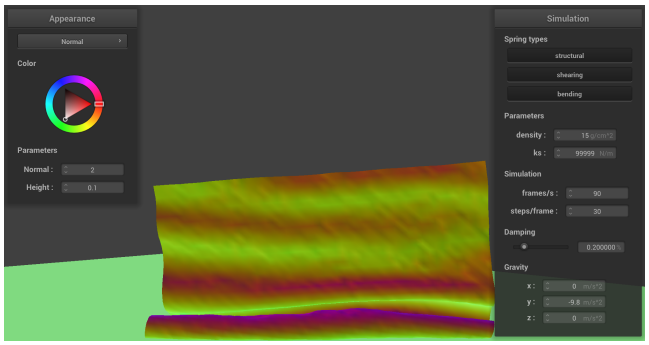


When it lays out flat, it is bubbly like a windy tarp or lava.

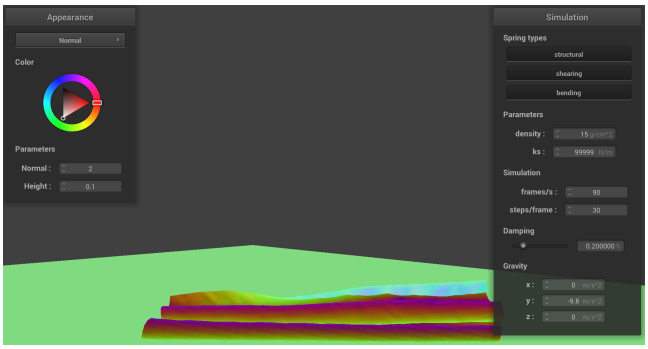
Below I will describe self collisions with ks = 99999 using normal shading.



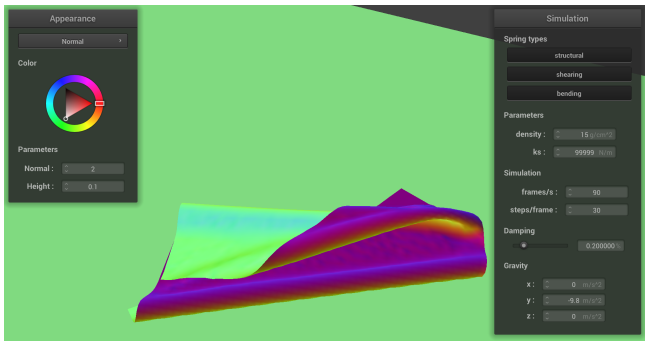
When there is a high ks, the cloth falls and creates big ripples.



The folds seem to be more uniform than the low density case.



The cloth slides into a nice overlapping pile like thin vinyl.



The cloth moves smoothly and falls in place with barely any ripples.

High ks and low density are similar. Low ks and high density are similar. The main difference between low ks and high density seems to be that low ks is much more bubbly and "springy" than the cloth with high density. For the low ks case, think of steamy boiling water. For the high density case, think of bubbling stew. The main difference between high ks and low density seems to be that high ks PointMasses are constrained more by adjacent PointMasses than the low density case.

Part 5: Shaders

Describe what you did in Part 5. etc...

In Part 5 I was able to make my cloth simulation look pretty in real time by implementing GLSL Shaders! GLSL Shaders run in parallel on the GPU, so they can do lots of computationally intensive work to make simulations look pretty without affecting preformance too much. The mandatory shaders I implemented were Diffuse Shading, Blinn-Phong Shading, Texture Mapping, Bump Mapping, Displacement Mapping, and Environment-mapped Reflections (similar to a mirror). I also did extra credit to make my own shader... A Cell Shader!

Explain in your own words what is a shader program and how vertex and fragment shaders work together to create lighting and material effects.

A shader program is a program that runs parallel on the GPU so that real-time renders can look good while still running fast. While the CPU calculates things such as collisions and mesh building that have to be done while knowing what everything else in the scene is doing, a shader program can run without needing to know those interactions. Because of that, we can essentialy pixel sample in parallel super fast and render an impressive scene at a quick speed while keeping a similar framerate. To write shaders in this project, I used GLSL. This parallelization feature allows shaders to do almost the same thing as the illumination functions I implemented in Project3, but with a different workflow.

To create lighting and material effects, a shader program uses vertex and fragment shaders. Vertex shaders change the geometric properties of vertices retrieved from the program running on the CPU. They can update the positions, normals, etc. of vertexes before sending that info over to fragment shaders. After handling and sending these vertex values, the values are interpolated over the polygon they will appear on. Fragment shaders take these interpolated values sent from the vertex shader and use them to change how the output pixels are colored. To keep things simple, if you want to edit the features in a scene you'd write a vertex shader. If you want to change how the features in that scene are colored, you'd want to write a fragment shader.

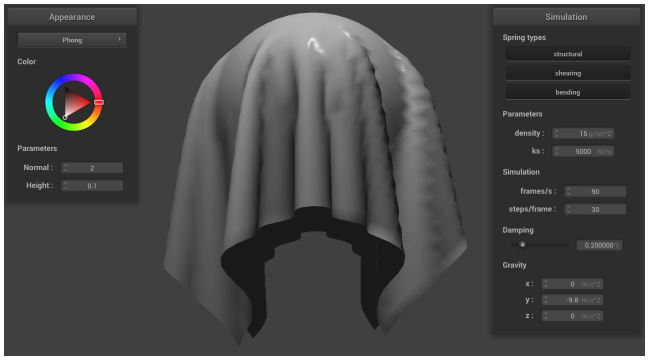
Explain the Blinn-Phong shading model in your own words. Show a screenshot of your Blinn-Phong shader outputting only the ambient component, a screen shot only outputting the diffuse component, a screen shot only outputting the specular component, and one using the entire Blinn-Phong model.

The Blinn-Phong shading model is a variation of the Phong shading model that is faster for directionally lit scenes. This shading model has components of diffuse shading, ambient shading, and specular shading. I'll give examples of the three below. The combination of these three means that we could make it so our objects look globally illuminated really quickly. The algorithm is as follows: separately compute ambient color, diffuse color, and specular color. Use the Diffuse Shading equation to get diffuse color. For specular color (the part the relects the shiny light), if the dot product between the direction to the light and the vertex normal (both normalized) is positive, then find the halfangle and put the halfangle to a power depending on how much you want the shine of the specular color to span.

Here I am only showing the ambient part of Blinn-Phong shading.

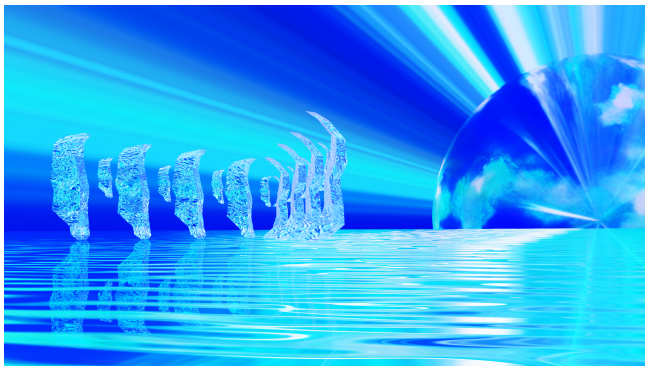
Here I am only showing the diffuse part of Blinn-Phong shading.

Here I am only showing the specular part of Blinn-Phong shading.

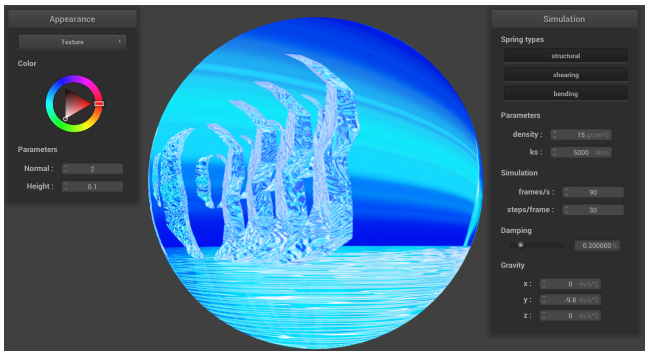


This is Blinn-Phong shading.

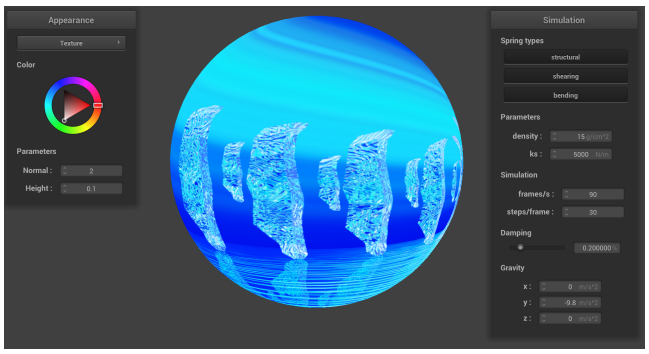
Show a screenshot of your texture mapping shader using your own custom texture by modifying the textures in /textures/.



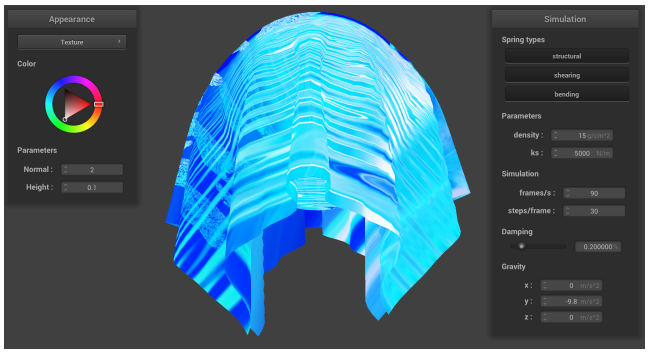
The texture I will be using.



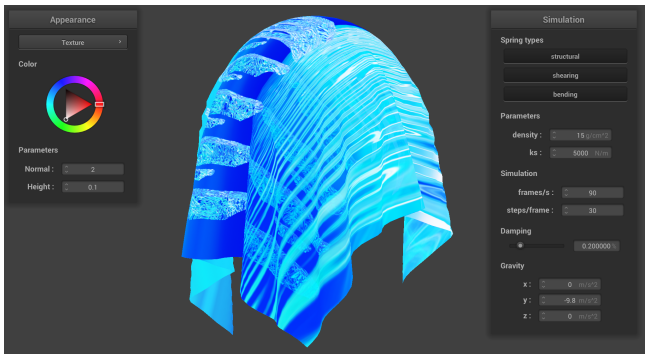
Front-on of sphere.



Side profile of sphere.

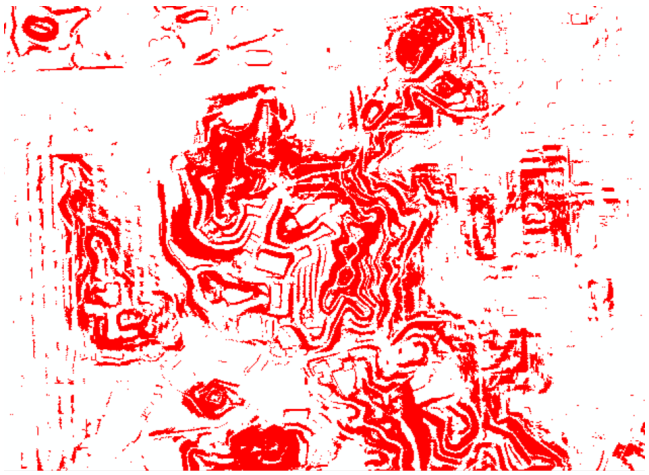


Front-on with cloth.

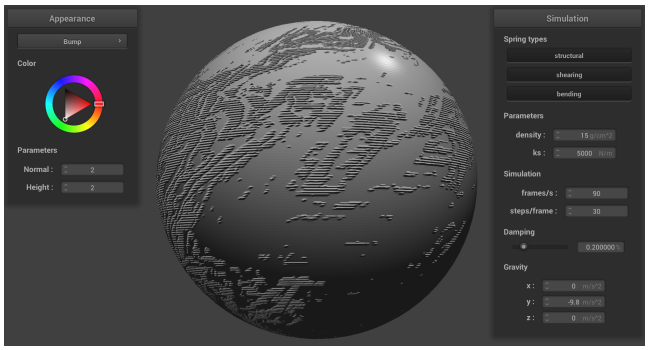


Side profile with cloth.

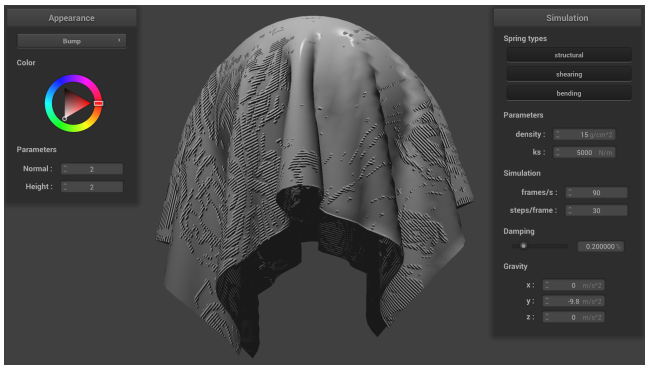
Show a screenshot of bump mapping on the cloth and on the sphere. Show a screenshot of displacement mapping on the sphere. Use the same texture for both renders. You can either provide your own texture or use one of the ones in the textures directory, BUT choose one that's not the default texture_2.png. Compare the two approaches and resulting renders in your own words. Compare how your the two shaders react to the sphere by changing the sphere mesh's coarseness by using -o 16 -a 16 and then -o 128 -a 128.



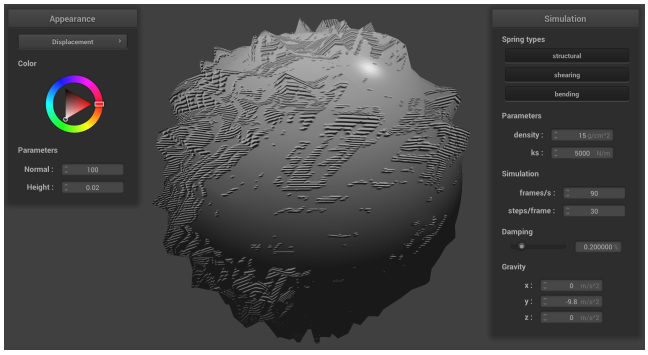
The texture I will be using.



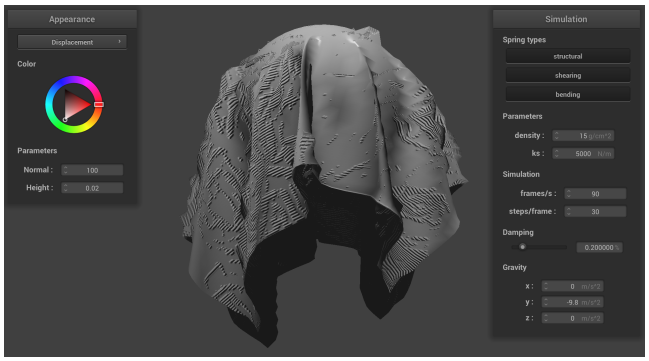
Bump Shading on sphere.



Bump Shading on cloth.



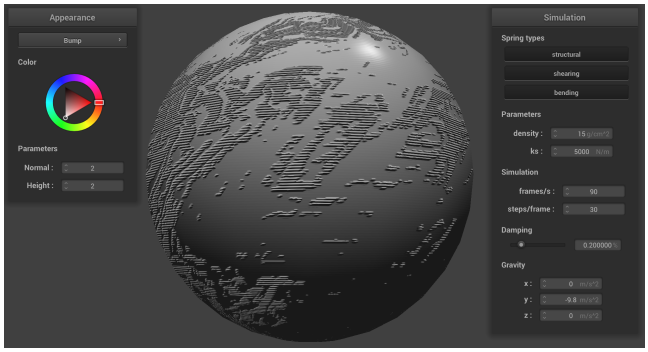
Displacement Shading on sphere.



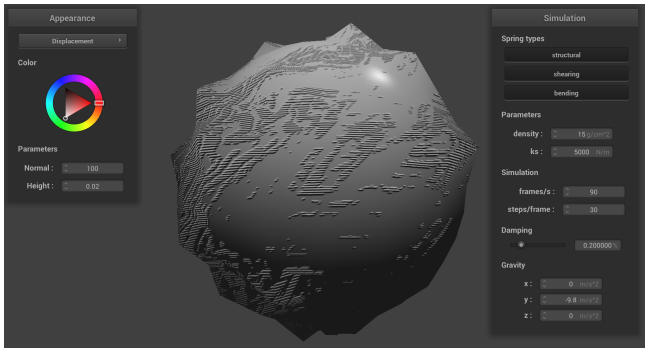
Displacement Shading on cloth.

Here it is very clear that Bump Shading "projects" features on to the surface of the objects to give the illusion that they extrude, while displacement shading actually moves the vertices. This is because in bump shading we only modify normals based on texture features, while in displacement shading we modify normals and vector positions. I chose to use a texture with high red contrast so it's easy to differentiate vectors that have been moved. The sphere clearly shows extruding vertices at areas that are red from the texture. It's harder to tell on the cloth, but it's still there.

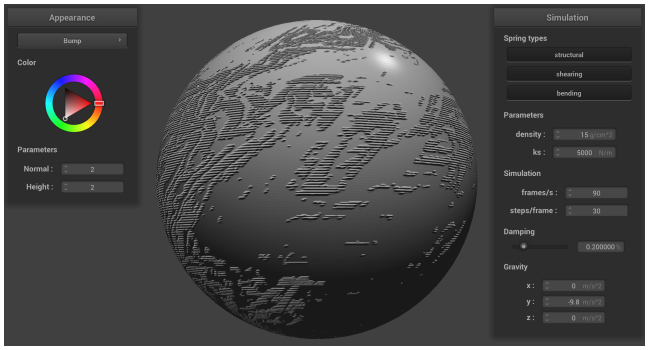
Below we will compare bump and displacement when we change the mesh's coarseness.



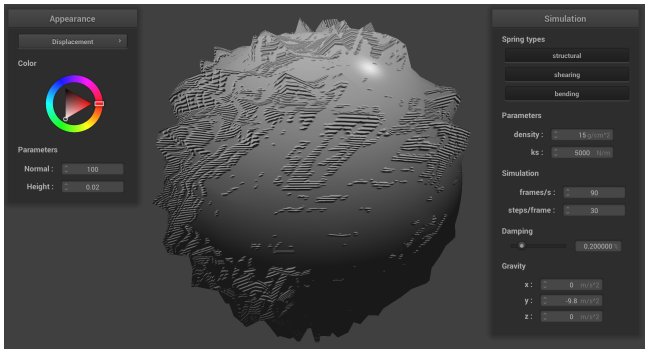
Bump Shading on sphere with -o 16 -a 16.



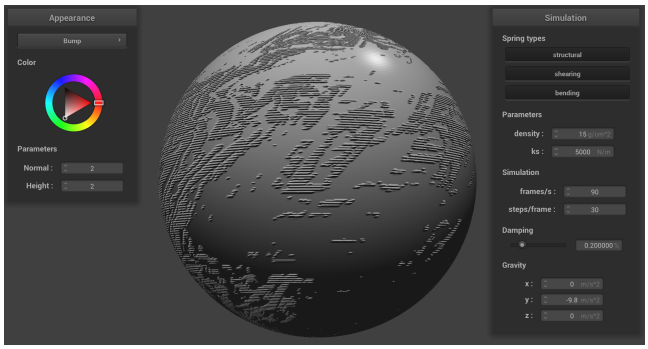
Displacement Shading on sphere -o 16 -a 16.



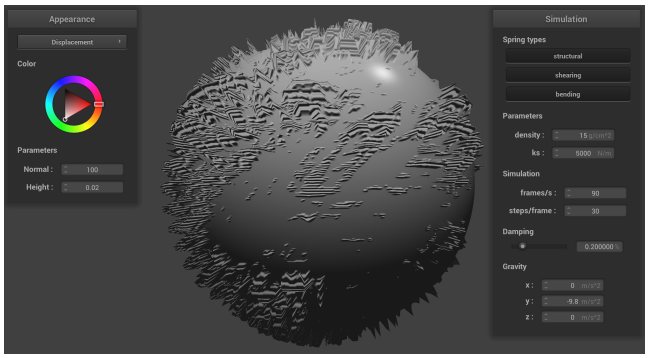
Bump Shading on sphere with default settings.



Displacement Shading on sphere with default settings.



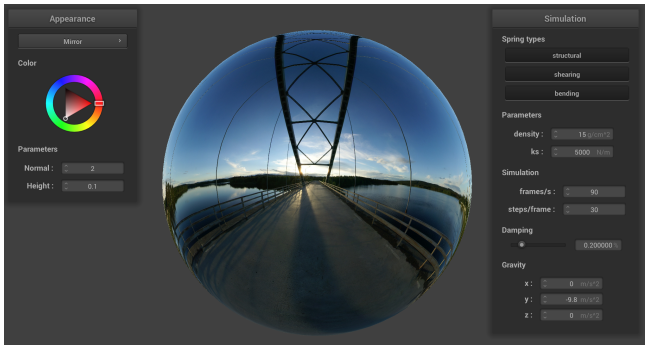
Bump Shading on sphere with -o 128 -a 128.



Displacement Shading on sphere with -o 128 -a 128.

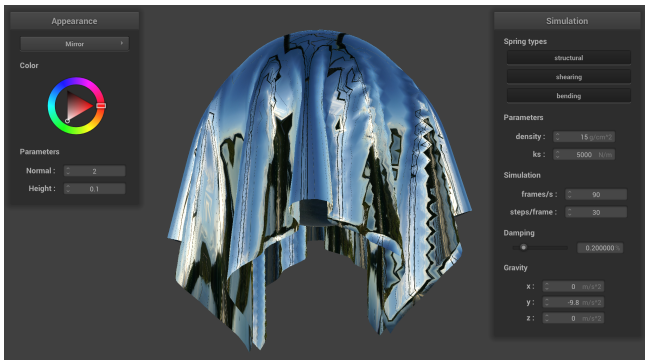
As the coarseness of the mesh increases, there is little to no quality increase on Bump Shading other than the roundness of the sphere. Displacement Shading, however, sees massive quality improvements. The introduction of more vertices means that displacements can be more precise. This is why the sphere goes from looking like a spiky ball with a texture projected onto it at -o 16 -a 16, to a planet with mountains at default settings, to a ball with hairs at -o 128 -a 128. In total, Bump Shading looks about the same on objects with varying coarseness, but Displacement Shading gets far more detailed as coarseness increases.

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

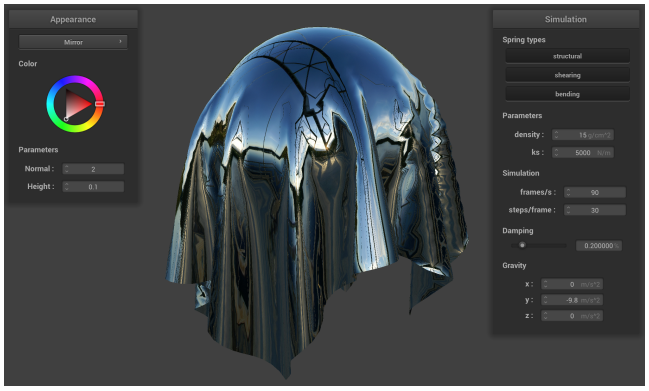


Front-on of sphere.

Side profile of sphere.



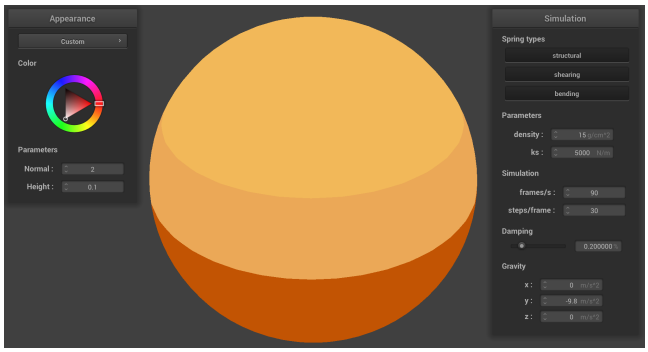
Front-on with cloth.



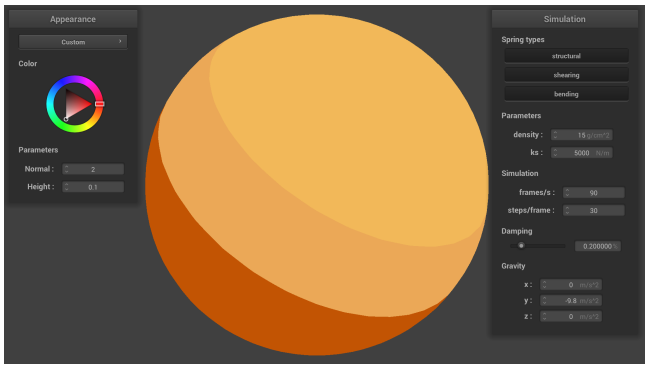
Side profile with cloth.

Explain what you did in your custom shader, if you made one.

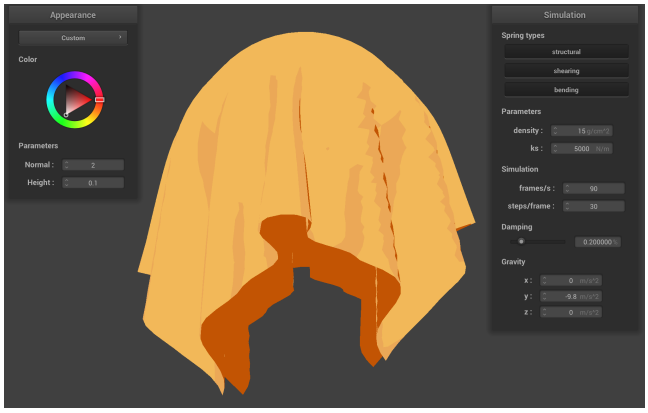
For my custom shader, I chose to make a cell shader. The idea behind a cell shader is that it bins similar color values into groups, then chooses one color to represent that entire group of colors. This simulates how colors are often used in cartoons and gives the world a comical feel. To find these bins, I decided to use the dot product between the direction to light and the normal.



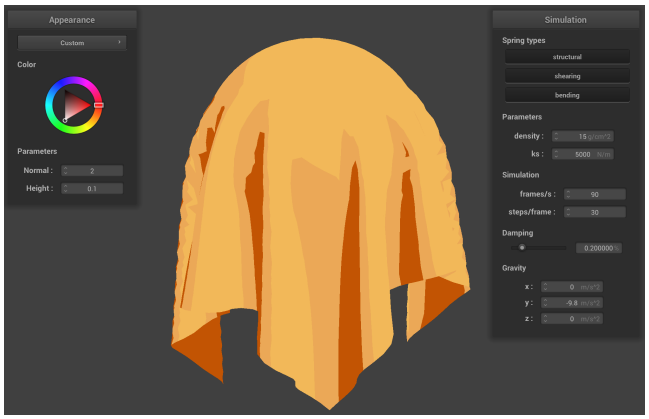
Front-on of sphere.



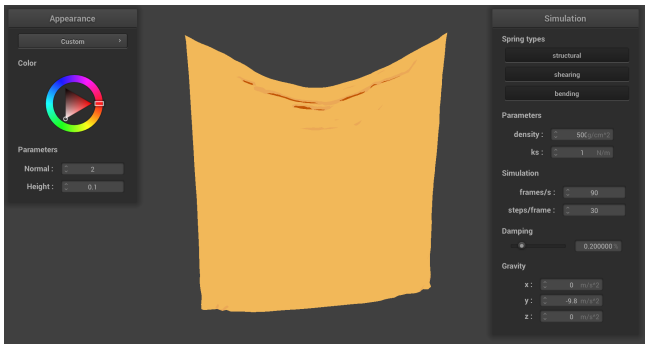
Side profile of sphere.



Front-on with cloth.



Side profile with cloth.



pinned2.

pinned4.

Unrealistic!

Part 6: Extra Credit

If you implemented any additional technical features for the cloth simulation, clearly describe what you did and provide screenshots that illustrate your work. If it is an improvement compared to something already existing on the cloth simulation, compare and contrast them both in words and in images.

I implemented a custom shader to simulate cartoony artsyles in real time! This shader is called a cell shader. Refer to the Part 5 section to see it!

The website for my writeup is https://cal-cs184-student.github.io/sp22-project-webpages-ethangnibus/