Cloth Sim

CS184/284A Spring 2025

By Edward Gilmore and Jovin Valdez

Live webpage: nekolocation.github.io/portfolio/cloth-sim

GitHub repository:

https://github.com/cal-cs184-student/sp25-hw4-lightning-mcqueen-crocs-by-lebron-james

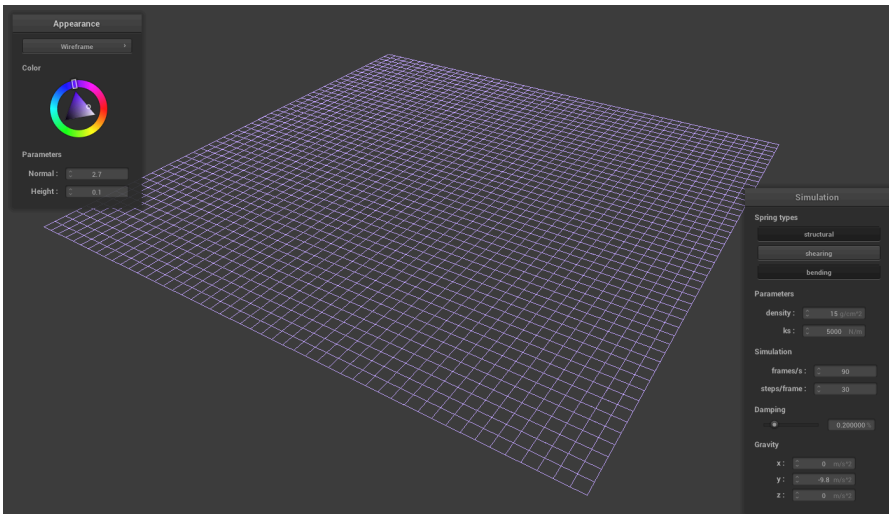
Overview

In this project we created an engine that simulates and renders real time physics acting on a cloth. We accomplished this by building out a grid of point-masses and applying different external forces (such as gravity) to those point masses– but we also simulate spring physics on the point masses, giving a feeling of elasticity to the cloth, allowing us to simulate collision with other objects, and self-collision with other parts of the cloth. This allows us to do things like drop cloth on spheres and let the cloth wrap around and cover it– or dropping cloth on the ground and letting the cloth crumple up on itself. Finally, we added the ability to render textures onto the cloth, allowing for various surfaces to be mapped to the contours of the cloth. It was really interesting playing around with the different parameters in the simulation (like the spring constant, density of the cloth, and external forces in different directions) and see how these changes impacted the way the cloth interacted with its environment.

Part 1: Masses and springs

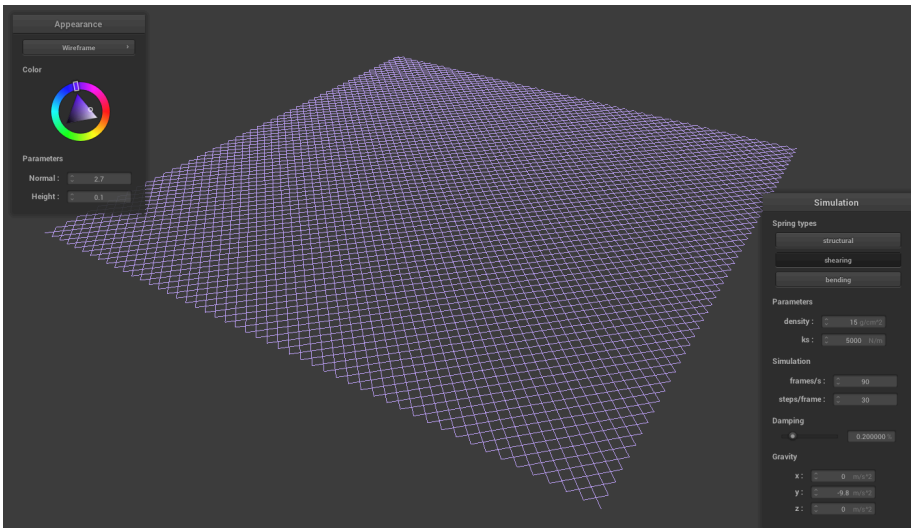
Take some screenshots of scene/pinned2.json from a viewing angle where you can clearly see the cloth wireframe to show the structure of your point masses and springs.

Show us what the wireframe looks like (1) without any shearing constraints



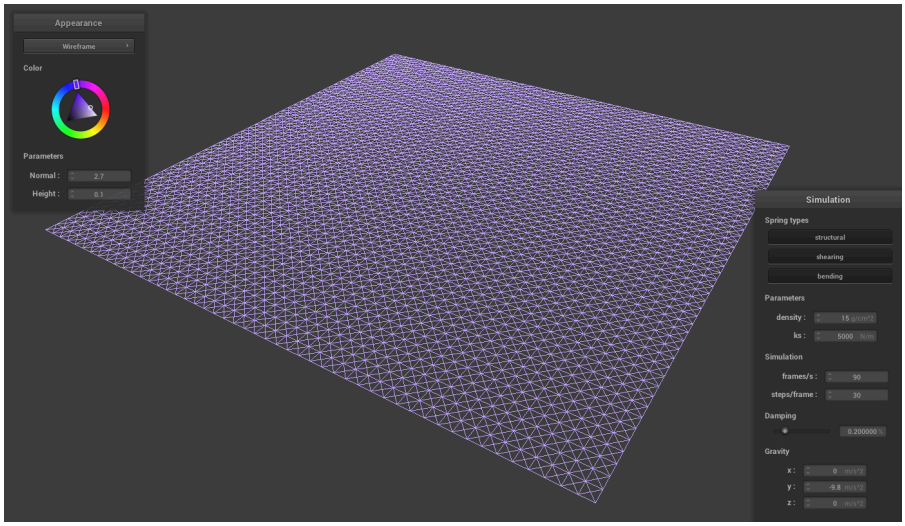
Wireframe without shearing constraints

(2) with only shearing constraints



Wireframe with only shearing constraints

and (3) with all constraints



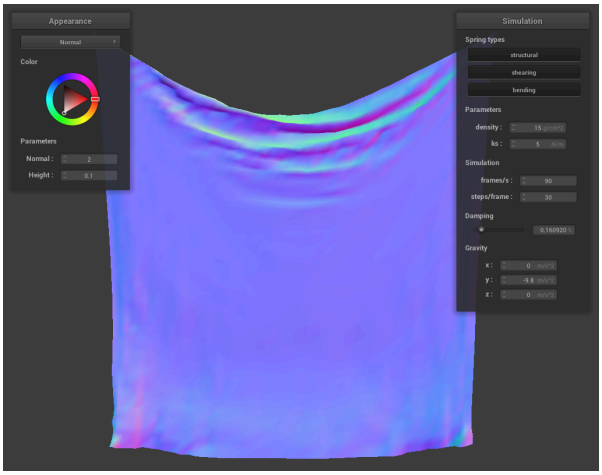
Wireframe with all constraints

Part 2: Simulation via numerical integration

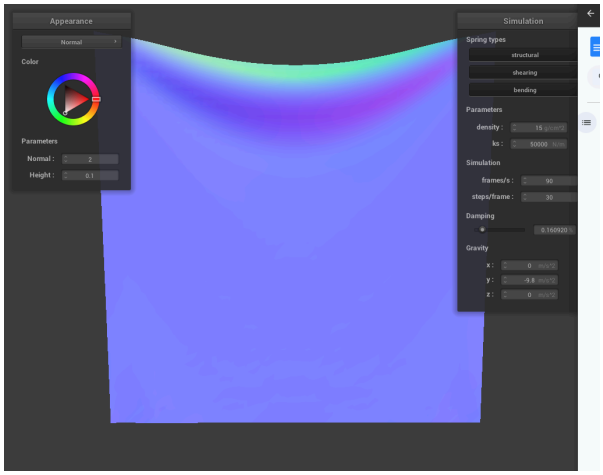
Experiment with some the parameters in the simulation. To do so, pause the simulation at the start with P, modify the values of interest, and then resume by pressing P again. You can also restart the simulation at any time from the cloth's starting position by pressing R.

- **For each of the below, 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.**
- **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 very low ks, the cloth falls faster and appears to bend and sag more, while with high ks the cloth appears much more rigid and stiff.



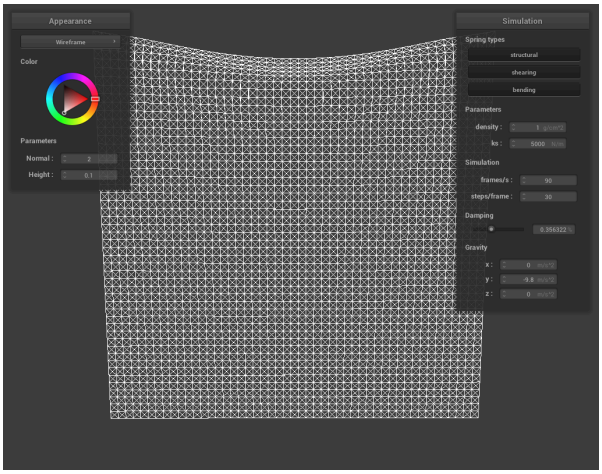
Low ks Value (5 N/m)



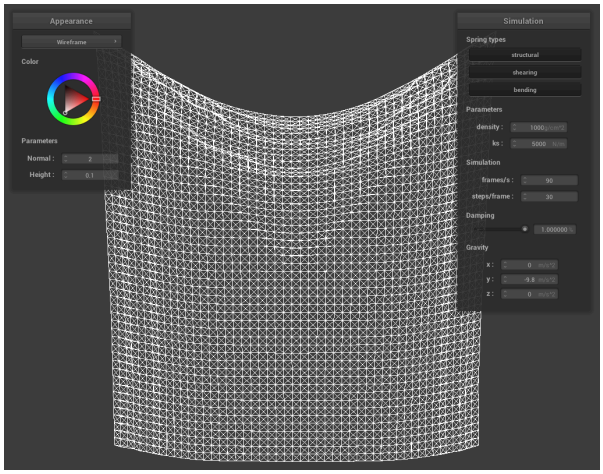
High ks Value (50000 N/m)

- **What about for density?**

For density, behavior appears to be the opposite of ks. With lower density the cloth appears more stiff , while with higher density the cloth sags more and is more bouncy and bendy.

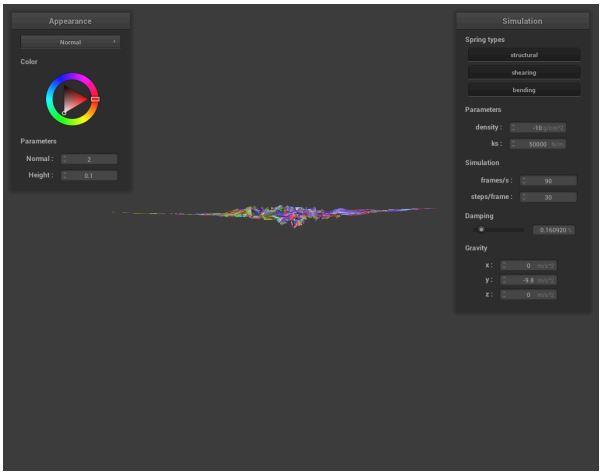


Low Density (1 g/cm^2)



High Density (1000 gm/cm^2)

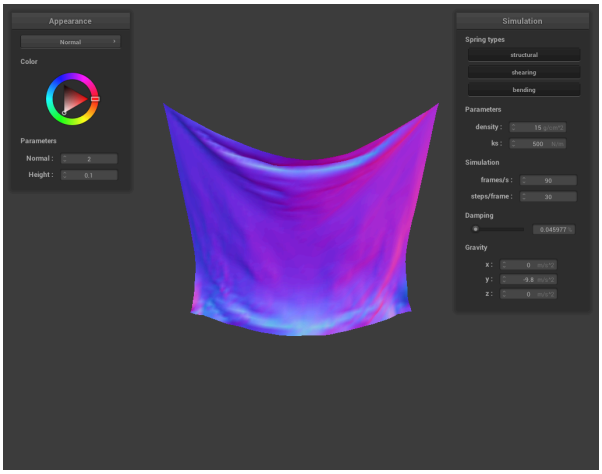
We can also set the density to a negative value to see beyond the limits of the universe:



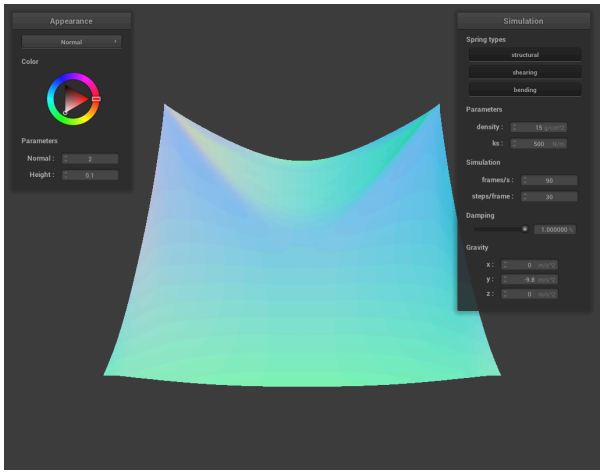
Density at -10 g/cm^2

- **What about for damping?**

With high damping the cloth slowly falls and comes to rest quickly. With much lower damping, the cloth falls faster and takes a longer time to come to rest since it sways back and forth.

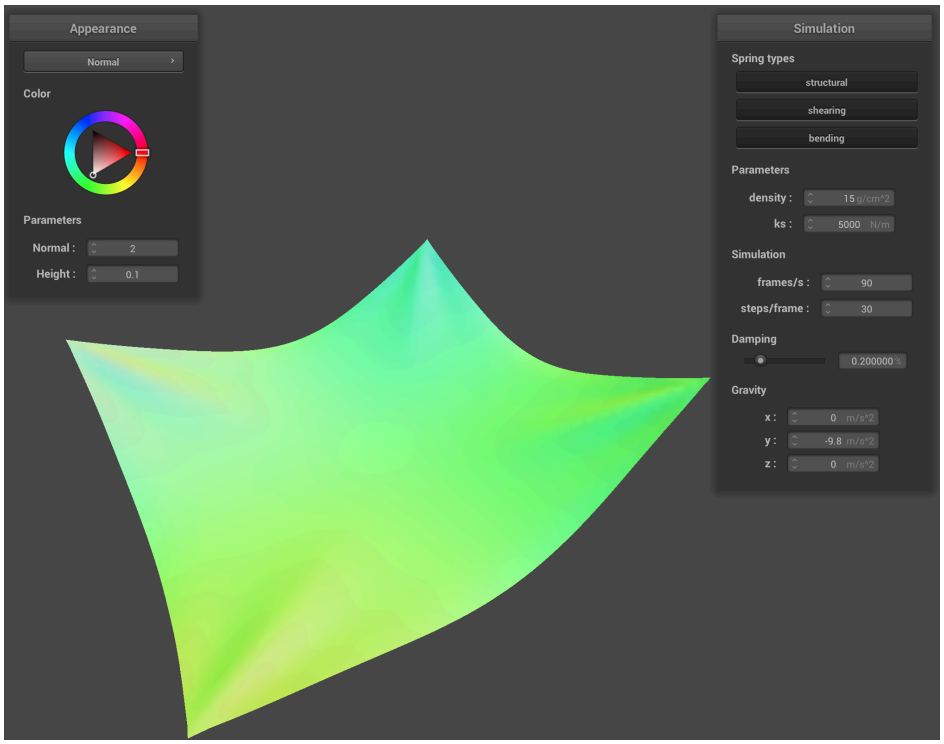


Swinging Caused by Low Damping (0%)



Graceful Swing in High Damping (100%)

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.

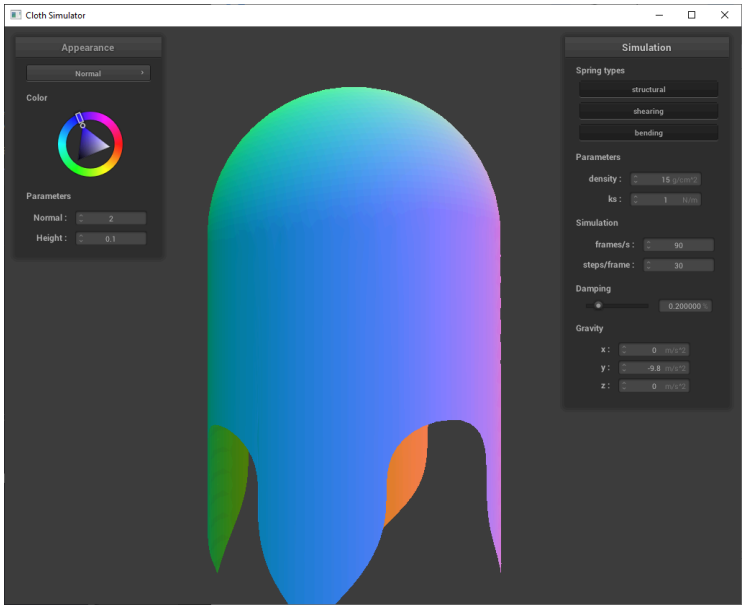


Part 3: Handling collisions with other objects

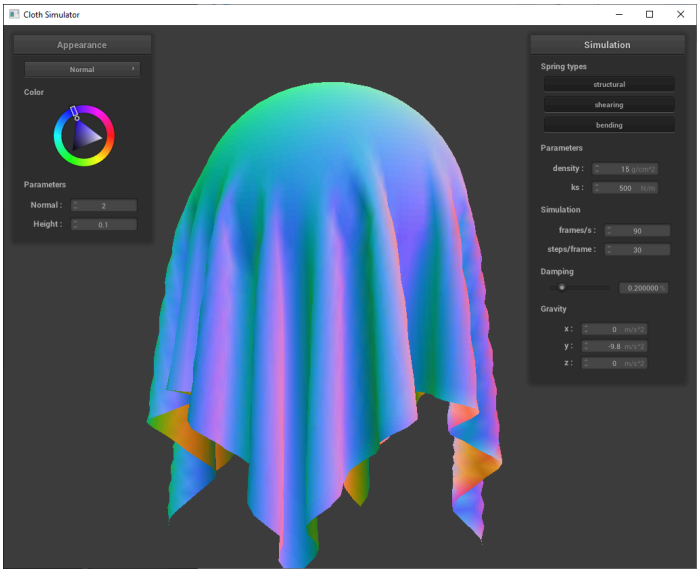
Describe your implementation of handling collisions with spheres and planes.
For spheres, we check if a point mass is inside the sphere by getting the distance from the sphere's origin to the point mass and checking if it's less than the radius. If so, we find the tangent point by getting the closest point on the surface, calculate a correction vector towards the tangent point, scale it by 1 - friction, and finally update the position by applying the correction to the last position.
For planes, we check collisions by seeing if the point mass crossed the plane between time steps. This is done by comparing signed distances of the current and last position relative to the plane's normal. If there is a collision (the current position is less than the surface offset), we get the depth of how far the point has penetrated the surface, and then a target point using that penetration depth. Finally, just like the sphere we get a correction vector and update the position accordingly.

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.

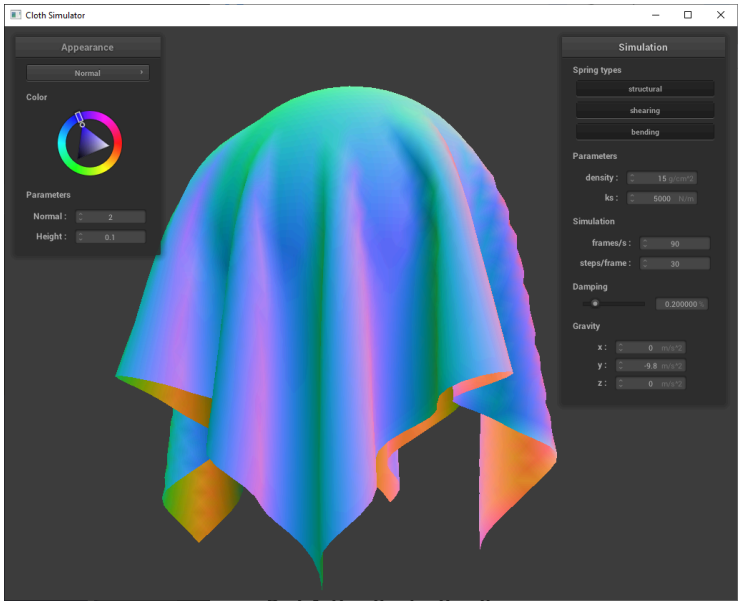
As we adjust the value of ks within our simulation, the rigidity of the cloth appears to change, since the spring constant has varying levels of affect. The lower the value of ks, the more the cloth fits to the contours of the sphere, and the less we see wrinkles. However, as the value of ks increases we see wrinkles appear with increasingly pronounced shapes as the cloth takes on a stiffer structure.



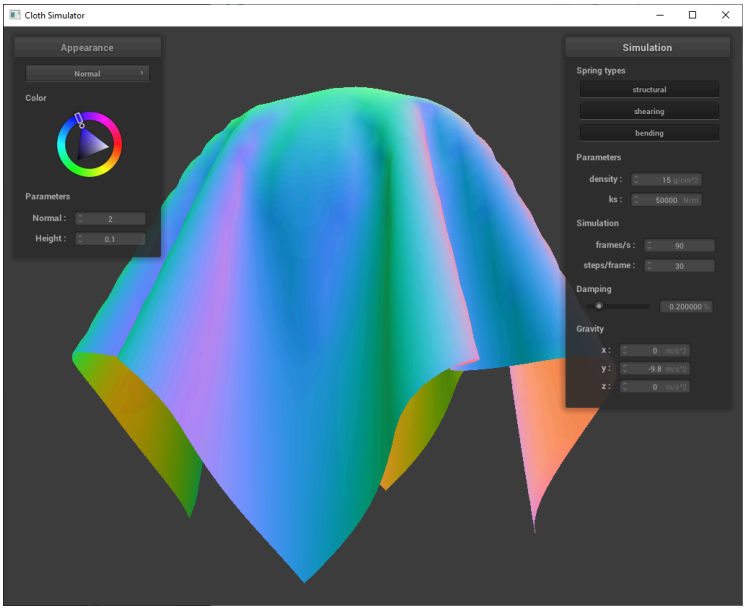
ks = 1



ks = 500



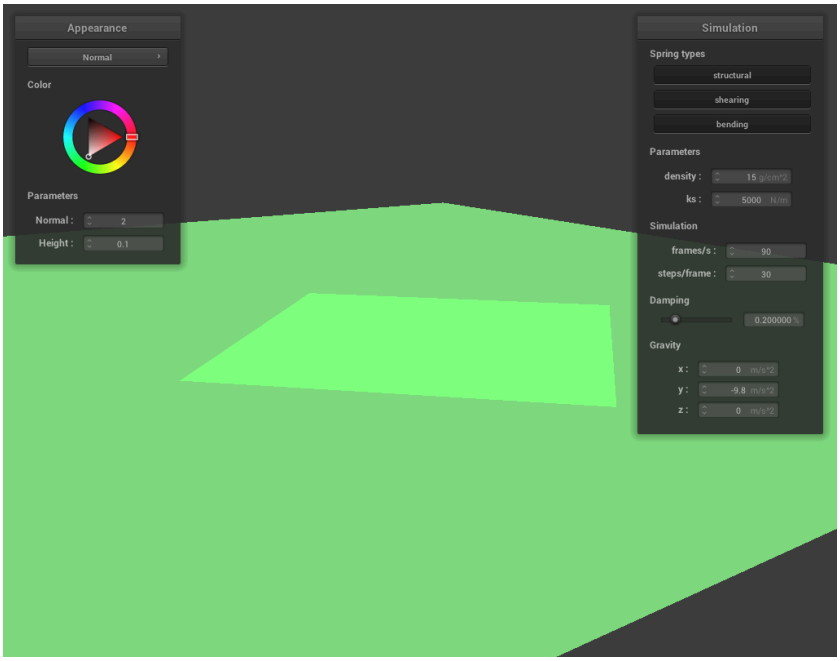
ks = 5000



ks = 50000

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.)

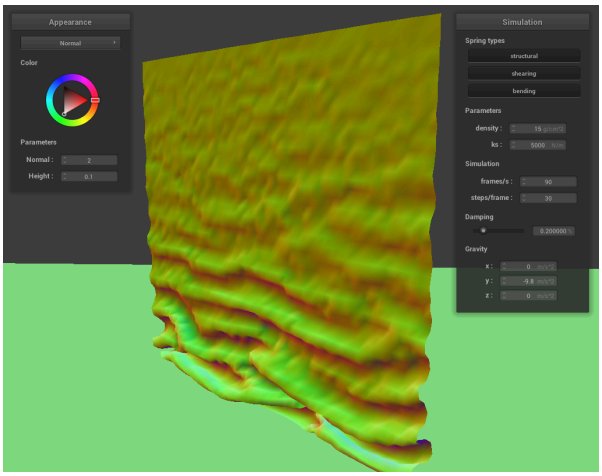
we can also make the cloth simulate how i looked all day after my last interview:



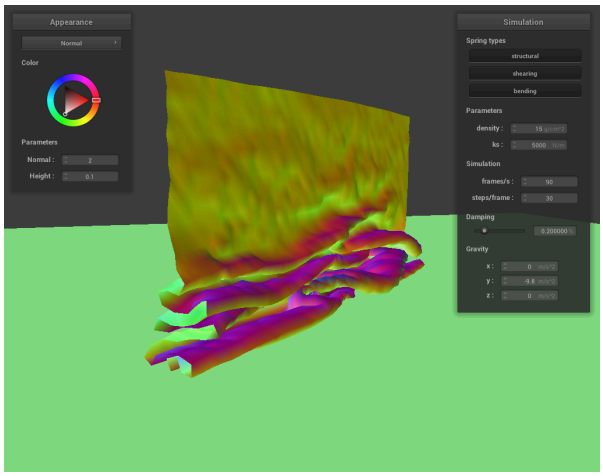
Part 4: Handling self-collisions

Describe your implementation of handling self-collisions.
To handle self-collision we implemented spatial hashing. We partition the 3D space into blocks that populate a hashmap, where keys are those block volumes and their values are the point masses within that block. The self_collide function then goes through the point masses and gets candidate point masses from the corresponding block in the hashmap, checking the distance to each candidate (excluding itself). If the distance is smaller than the threshold distance of 2 * thickness, a correction vector is made to push the point mass away from the candidate. These correction vectors are averaged and scaled down by simulation_steps before being applied to the point mass, ensuring that the points maintain separation.

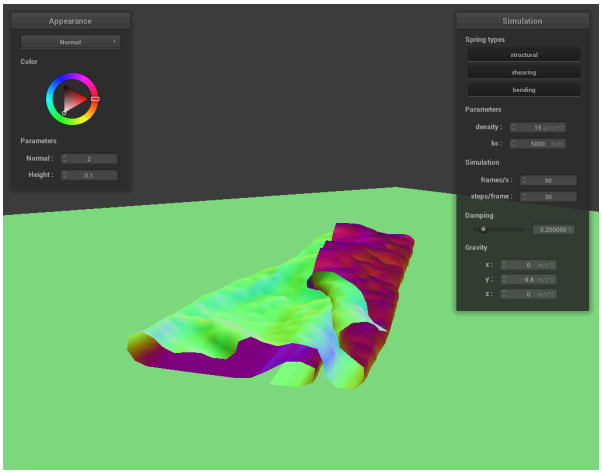
Show us at least 3 screenshots that document how your cloth falls and folds on itself, starting with an early, initial self-collision and ending with the cloth at a more restful state (even if it is still slightly bouncy on the ground).



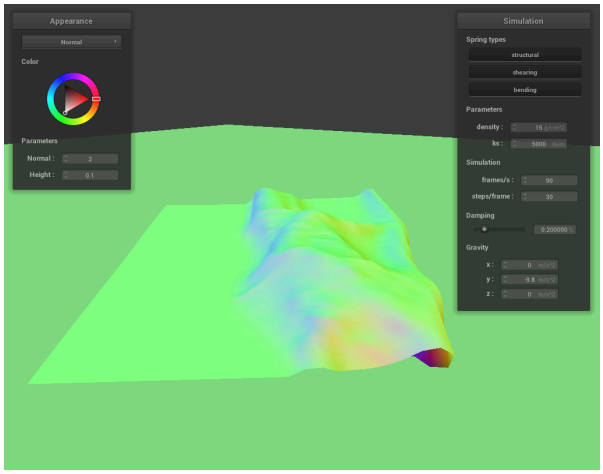
Early cloth self-collision



Cloth mid-fall



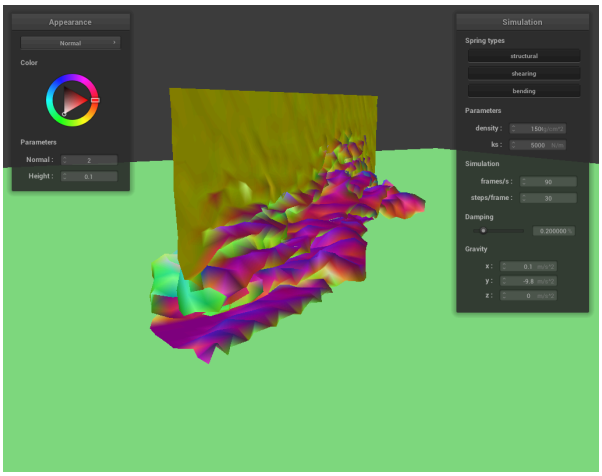
Cloth piling on the ground



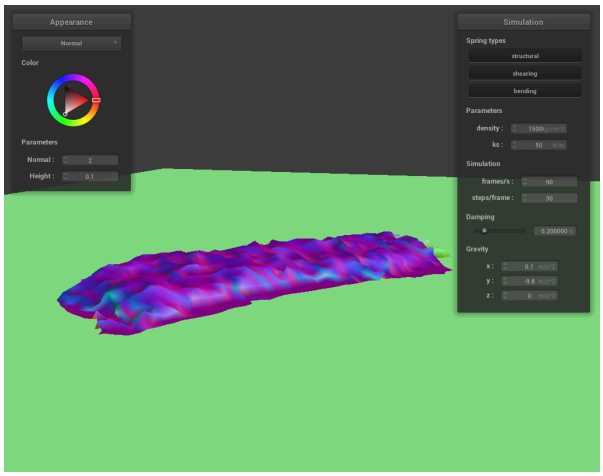
Cloth in a more restful state

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.

As we increase the density of the cloth, it falls to the ground much faster. This happens because the acceleration of the cloth is based on its mass (and as the density adjustment makes no changes to the volume of the cloth, the density adjustment is only affecting the mass). We can see below that the cloth piles in a more compact and chaotic shape on the ground, and the part of the cloth that has not yet hit the ground keeps a more flat shape. This is probably because the forces pulling the cloth directly towards the plane are stronger than the other spring constraints. It also becomes more flat more quickly than with lower density values, as we'll later see.

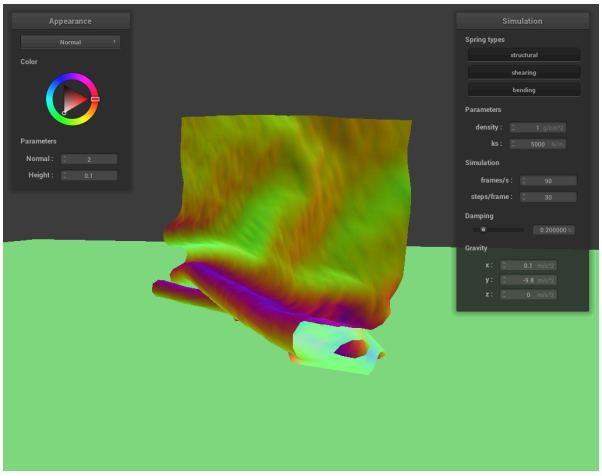


Cloth mid fall - High Density

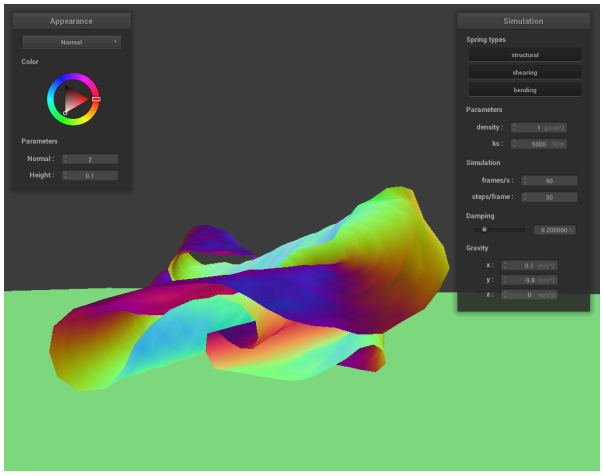


Cloth in restful state - High Density

With lower density values, the shape of the cloth contorts much more quickly and in a more impactful way. As well, there's more bounce and air-time as the cloth hits the ground. As before, this is probably because the lower acceleration force means that the impact of the spring forces in the cloth are more evident.

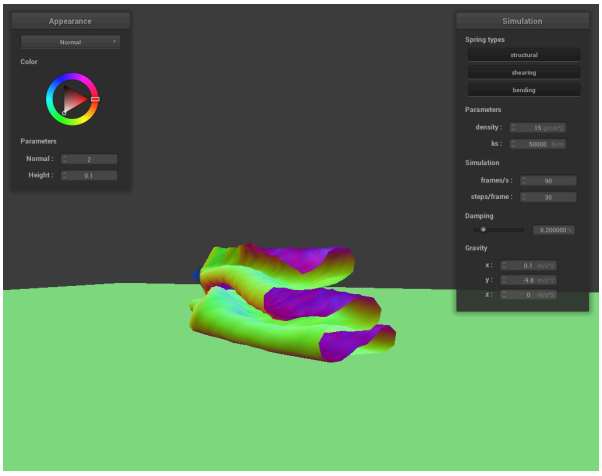


Cloth mid fall — Low Density

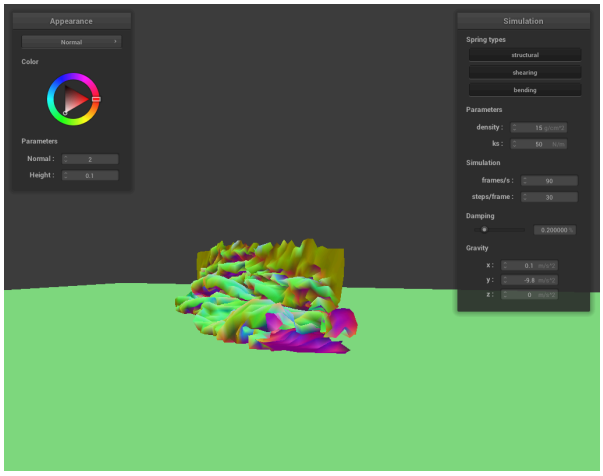


Cloth piling on ground - Low Density

When changing the ks parameter, a notable affect is that is the ks value increases, the rigidity of the shape is increased and the folds are more pronounced and rounded. With lower ks values, the shape is less rigid and takes on almost a gelatinous shape and movement.



High ks value (50000 N/m)



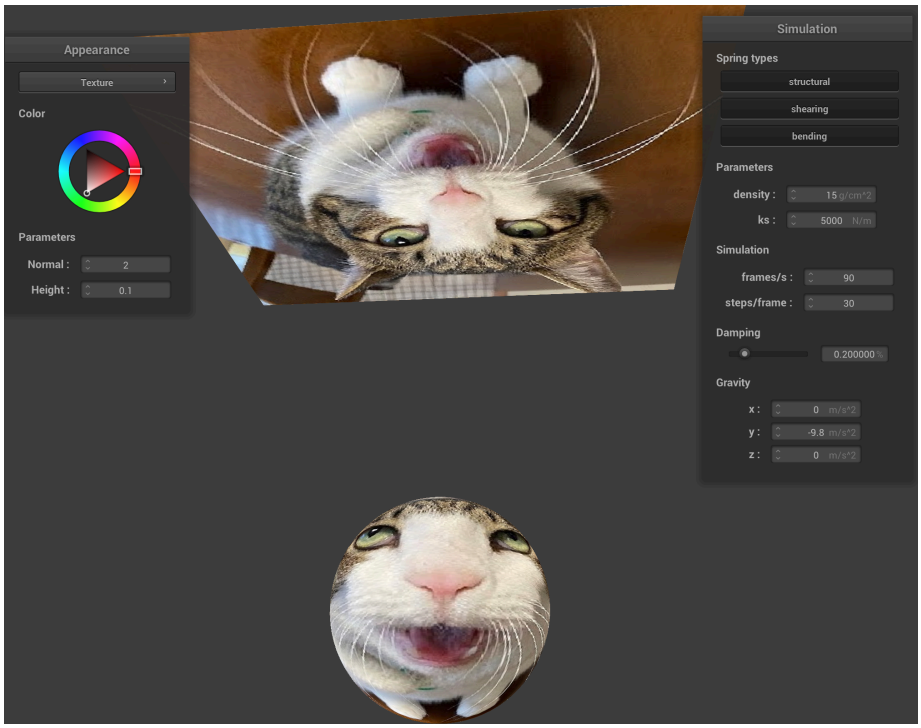
Low ks value (50 N/m)

Part 5: Cloth Sim

Explain in your own words what is a shader program and how vertex and fragment shaders work together to create lighting and material effects.
Shaders are programs that the GPU executes, which allow for more realtime high-performance graphics. A vertex shader runs for every vertex, transforming its position from world to screen space, and passing varying outputs like texture coordinates and world space normals. Then, fragment shaders are run on pixel fragments, using these passed varying data to calculate final colors of the fragments based on lighting and material properties.

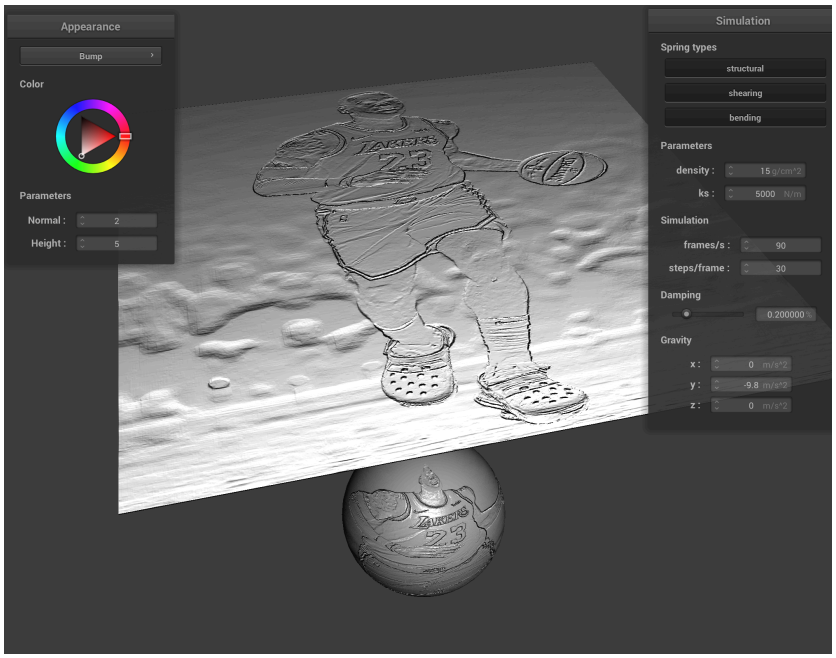
Explain the Blinn-Phong shading model in your own words. Show a screenshot of your Blinn-Phong shader outputting only the ambient component, a screen shot only outputting the diffuse component, a screen shot only outputting the specular component, and one using the entire Blinn-Phong model.
The Blinn-Phong shading model computes the color of a point on a surface in order to simulate realistic lighting, combining components of ambient light, diffuse reflection, and specular reflection. Ambient light represents indirect light providing global illumination, diffuse reflection simulates light scattering on matte surfaces depending on hitpoint angles, and specular reflection provides shiny highlights. The final result color is the sum of these three components, weighted by material and lighting coefficients.

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

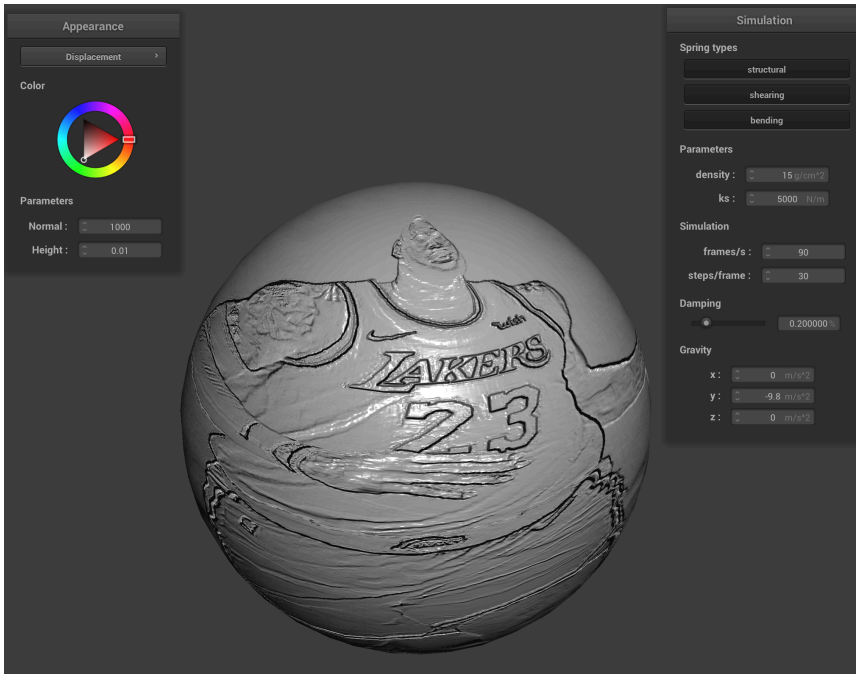


Show a screenshot of bump mapping on the cloth and on the sphere. Show a screenshot of displacement mapping on the sphere. Use the same texture for both renders. You can either provide your own texture or use one of the ones in the textures directory, BUT choose one that's not the default texture_2.png. 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.

Bump mapping and displacement mapping both enhance the detail of surfaces using textures. Bump mapping purely uses shading, modifying only the surface normals based on a heightmap texture's slopes. This doesn't actually modify the geometry, it only gives the illusion of texture through lighting. With displacement mapping vertex positions are actually changed, being moved along their normals according to their height map value. This actually modifies the geometry, resulting in more realistic lighting.



Bump mapping with the goat in crocs



Displacement mapping

With bump mapping, switching between -o 16 -a 16 and then -o 128 -a 128 doesn't really provide a significant difference of the texture appearance. However, with displacement mapping the texture was more noticeably changed, the triangle angles of the sphere were much more pronounced with the lower resolution. This makes sense since the actual normals of the surface with displacement mapping is altered.

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