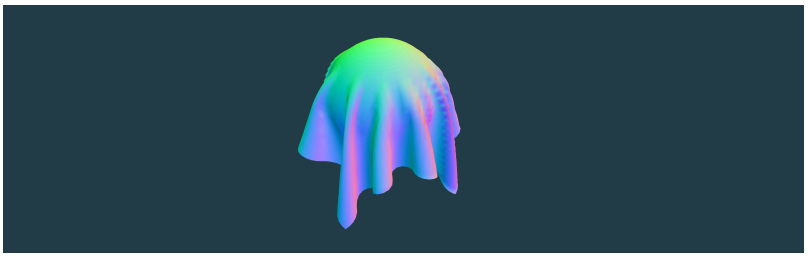
CS 184: Computer Graphics and Imaging, Spring 2024

Homework 4: Cloth Simulator

Ethan Tam



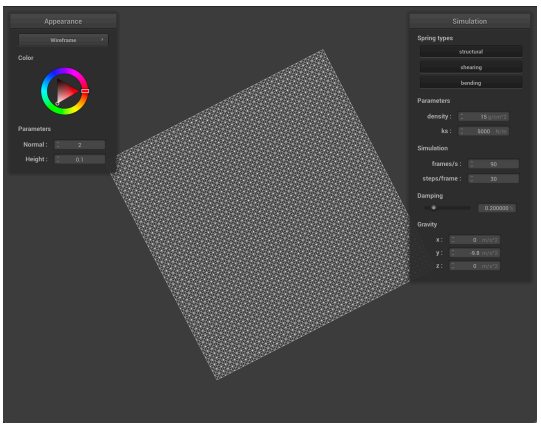
Overview

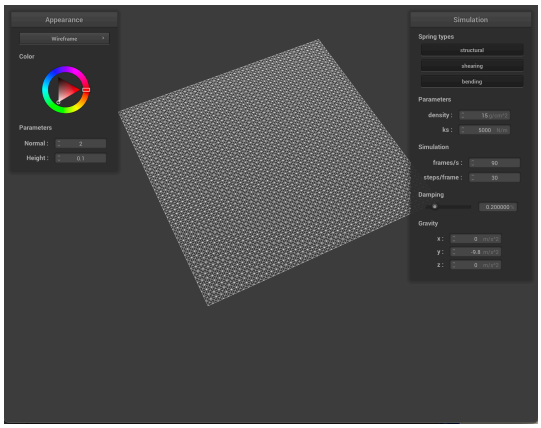
In this assignment, I created a real-time cloth simulation system by integrating point masses, springs, and physics principles to portray and control the cloth's behaviors/movements. I used Verlet integration to calculate the new time steps and adjust point mass positions. At the same time, I implemented constraints to stop undesirable deformations in and during the simulation. In addition, I improved the system by including and implementing collision handling mechanisms, allowing physical and realistic interactions with other objects like spheres and planes. I also address self-collision through using a hashtable to create efficient detection methods. To make the simulation more realistic, I optimized rendering with special shaders tailors to fit a multitude of cloth materials. In the end, I achieved and created a fully functional real time cloth simulator that is capable of simulating cloth behaviors in an accurate way through with the help of physics and by taking into account different material characteristics.

Part I: Masses and springs

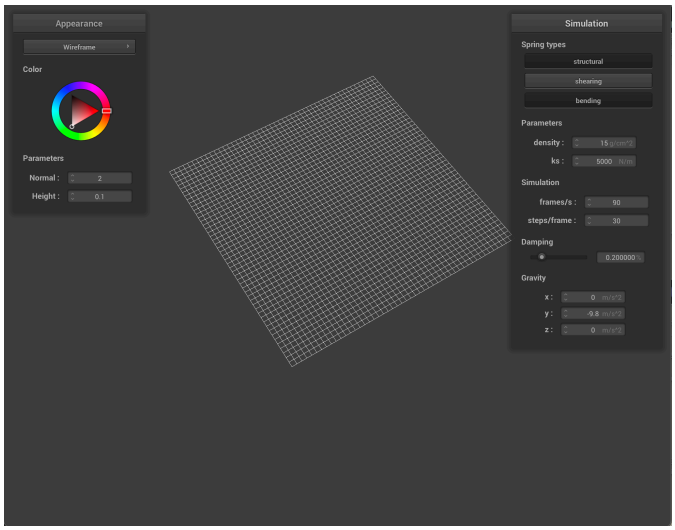
In this project, I used a system of point masses and springs to create the cloth model, dividing the cloth with evenly spaced point masses and creating connections between using specific spring types. In Cloth::buildGrid(), I initialize this system by creating a grid of evenly spaced point masses. Then by considering the cloth's orientation (horizontal or vertical) while iterating through num_height_points and num_width_points, I determine the total number of masses. In each calculated position, I create a new PointMass object. Its initial setting for its pinned is false and is then adjusted later. I then store all point masses in the point_masses vector using emplace_back() in row-major order. At the same time I iterate through the point_masses vector to determine whether each point mass should be pinned based on the indices contained in the pinned vector. Next I defined three types of springs. Shearing springs connect diagonally between point masses, while structural and bending springs connect adjacent and distant point masses, respectively. Once I determine the edge case, I iterate through all point masses to create springs with the appropriate point masses at each end. Then by specifying the type of spring, I store them in the springs vector using emplace_back().

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.

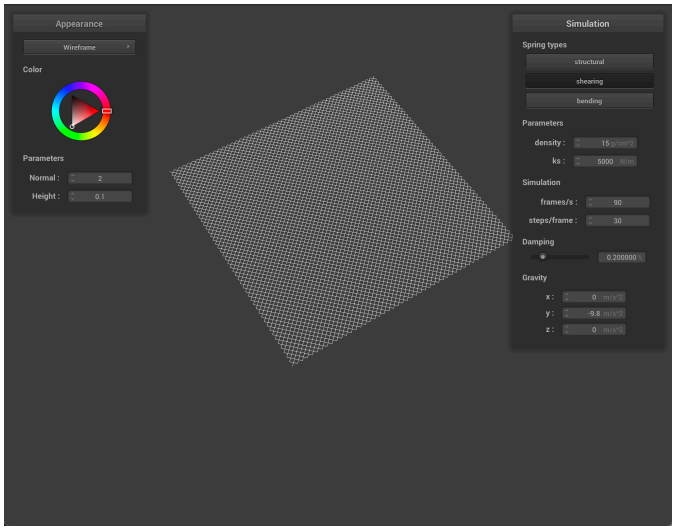




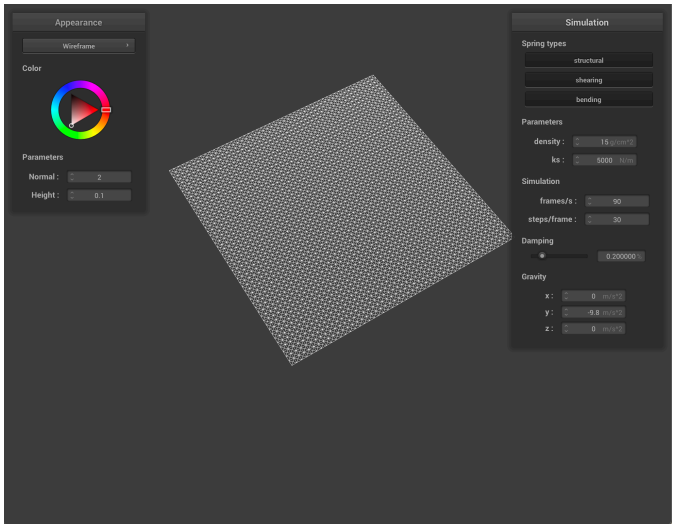
Show us what the wireframe looks like (1) without any shearing constraints, (2) with only shearing constraints, and (3) with all constraints.



Wireframe without any shearing constraints



Wireframe with only shearing constraints

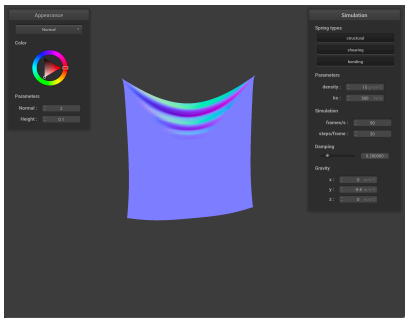


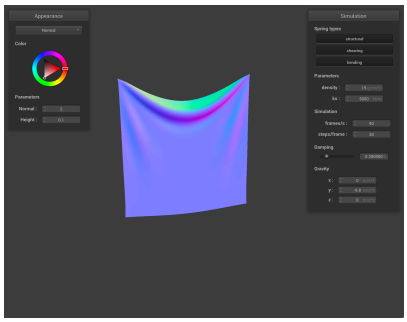
Wireframe with all constraints

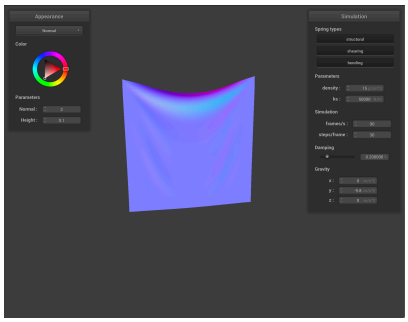
Part II: Simulation via numerical integration

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.

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?

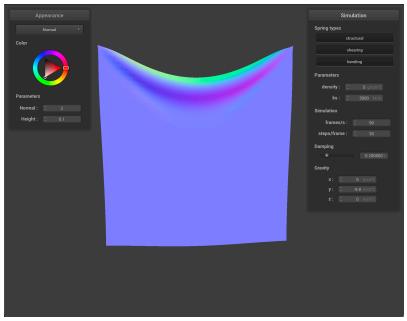

ks = 500 N/m

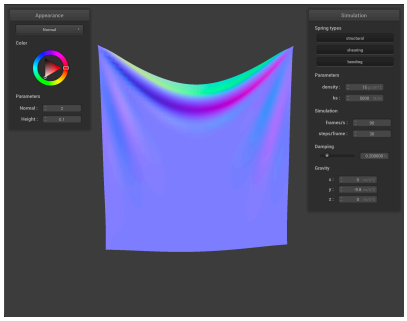

ks = 5000 N/m

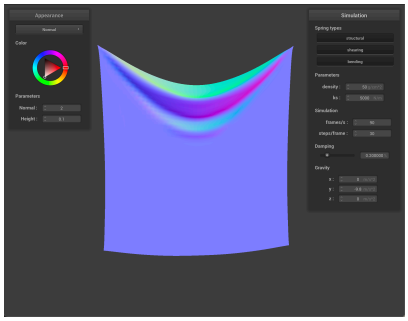

ks = 50000 N/m

When I use low ks, the cloth behaves like a thin piece of stretchy fabric. It's really slow when it comes to going to its final resting state and has a lot of ripples at its resting state. When it comes to high hs, the cloth is stiffer and has fewer ripples on the cloth in its final resting state. This makes sense because ks is the spring constant so it directly affects the stiffness of an object. I experimented with ks = 500 N/m, 5000 N/m, and 50000 N/m. For 500 N/m, there is the absence of wrinkles on the sides, just pronounced curving bends on the top. With ks = 5000 N/m, the fold at the top is bigger and there is only one. There are also side curves/folds in ks = 5000 N/m. With ks = 50000 N/m, the cloth is really stiff and there is little folding at the top. There are also faint wrinkles in the cloth. This makes sense because the bigger the ks the more stiff the cloth looks.

What about for density?

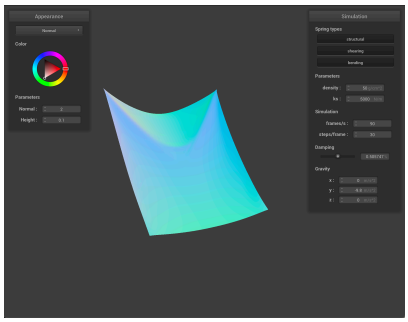

density = 5 g/cm^2

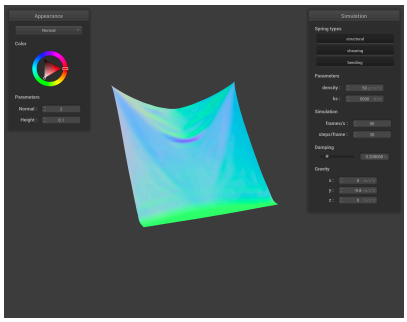

density = 15 g/cm^2

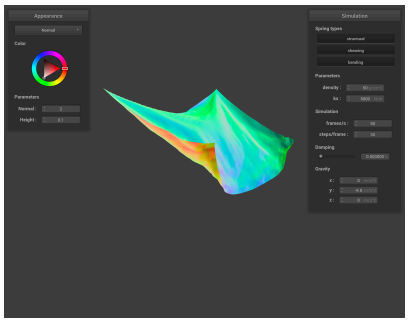

density = 50 g/cm^2

When the density is low, the cloth looks light and thin with small bounces whereas at high density, the cloth is heavier looking and looks more weighted down. This is because the density refers to the thickness of the cloth and when a cloth is more dense, there is more material and is thicker hence heavier. When experimenting with density, I used density = 5 g/cm^2, 15 g/cm^2, 50 g/cm^2. With 5 g/cm^2, the bend at the top is less curved and it seems like it's less heavy. With 15 g/cm^2, the fold at the top is more pronounced because the cloth is heavier. For 50 g/cm^2 the curve on top is even more curved that it created and extra fold because it's even heavier.

What about for damping?


damping = 0.5%

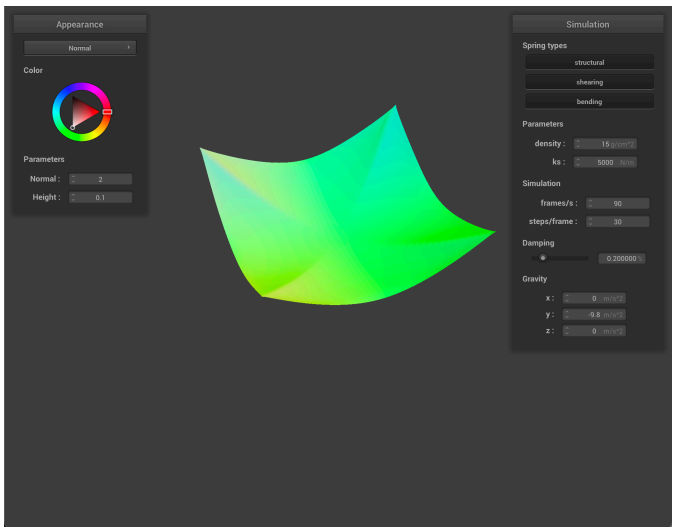

damping = 0.2%


damping = 0.0%

When the damping is low, the cloth takes a long time to converge to its final resting stage. However, the simulation is more wild as there were more ripples, bounces, and folding before it reached its final resting stage. When the damping is high the cloth has less

bouncing and rippling and goes to its final resting state quicker. This is because damping is the loss of energy from factors like heat loss or friction. Low damping means little energy lost which makes going to its final resting stage very long. Compared to high damping, the energy loss faster makes it reach its final resting stage quicker. I experimented with damping = 0.0%, 0.2%, and 0.5%. When looking at the results, there is no difference in final resting stage like the other factors we experimented with, however, the process to getting to the final state is very different. For damping = 0%, the cloth took a long time to reach the final state because there was a lot of bouncing back and forth. For the higher damping %, the more I increase it, the faster it reaches the final state with less bounce and less folding on itself.

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.

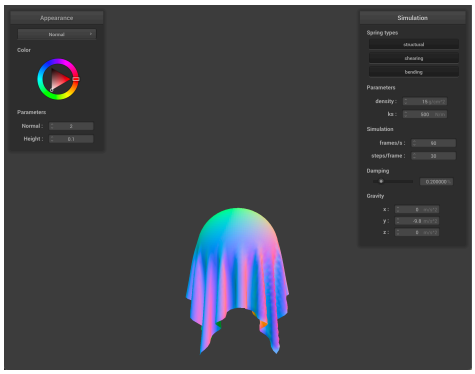


Default parameters

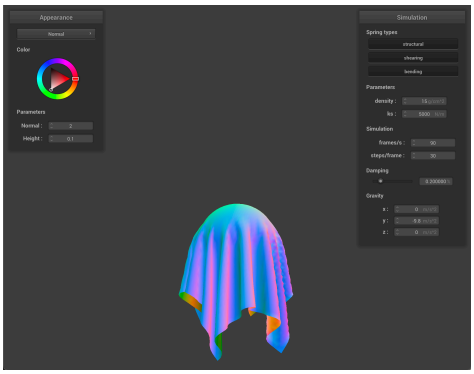
Part III: Handling collisions with other objects

In Cloth::simulate(), I integrate collision handling for cloth with spheres and planes by looking at the interactions between each point mass and objects in the collision. For sphere collision (Sphere::collide()), I check if the point mass intersects with the inside of the sphere and adjust its position to the surface to prevent it from going through the sphere. To do this, I compute the tangent point on the sphere's surface and derive the correction vector from the difference between this tangent point and the point mass's previous position. Then I scaled by the friction coefficient. For plane collision (Plane::collide()), I evaluate if the point mass is within or intersects the plane by comparing the dot product of the plane's normal with the vector between the point mass's previous position and a point on the plane. Similar to sphere collision handling, if it goes through or is crossing the plane, I compute the tangent vector using ray-plane intersection equations. I calculate the correction vector based on this tangent and the point mass's previous position, then I adjust by a small surface offset displacement and the plane's normal. Finally, I update the point mass's position accordingly.

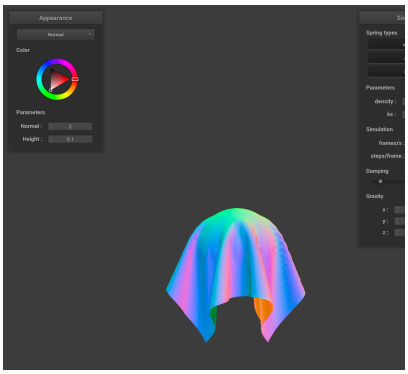
Show us screenshots of your shaded cloth from scene/sphere.json in its final resting state on the sphere using the default ks = 5000 as well as with ks = 500 and ks = 50000. Describe the differences in the results.



ks = 500



ks = 5000

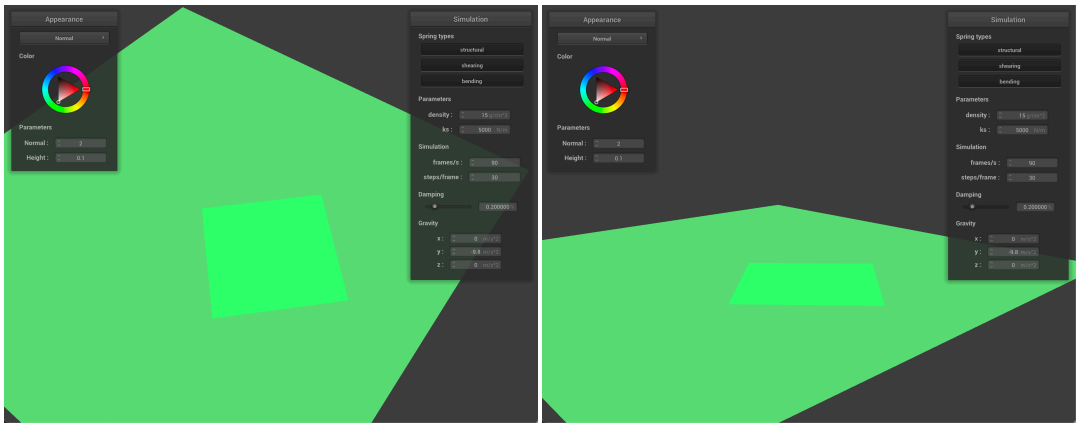


ks = 50000

When ks = 500, the thing that the cloth is covering is obviously a sphere as it wraps around 50 perfect of the sphere tightly. The cloth is also more elastic looking. When ks = 5000, the thing that the cloth is covering can be seen as a sphere if you look close because the cloth wraps the top 25% of the sphere perfectly. There is also some stiffness in the cloth but it is not too stiff. When ks = 50000, it is

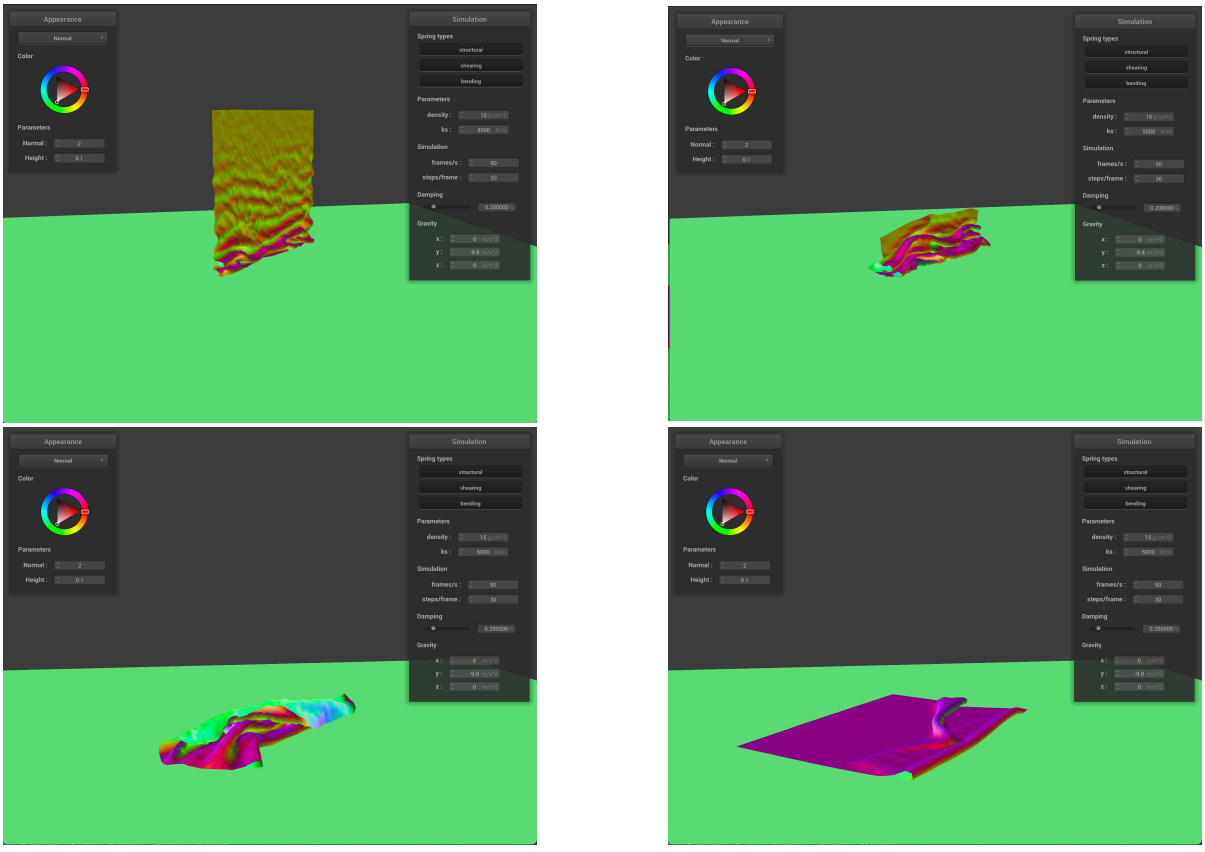
hard to tell the shape of the object because the cloth is super stiff and doesn't hug the object as tightly. As a result, it is seen that higher ks means the cloth is stiffer and lower ks means that the cloth is more stretchy and less stiff.

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.)

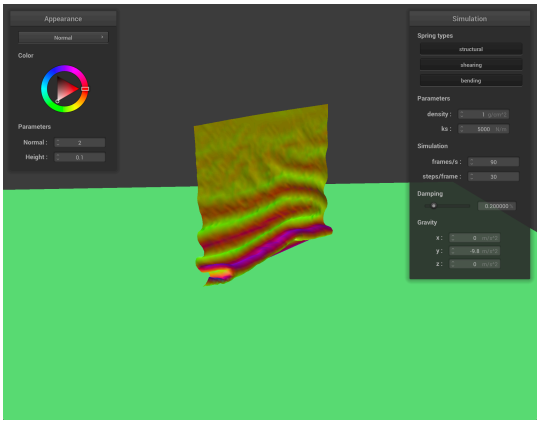


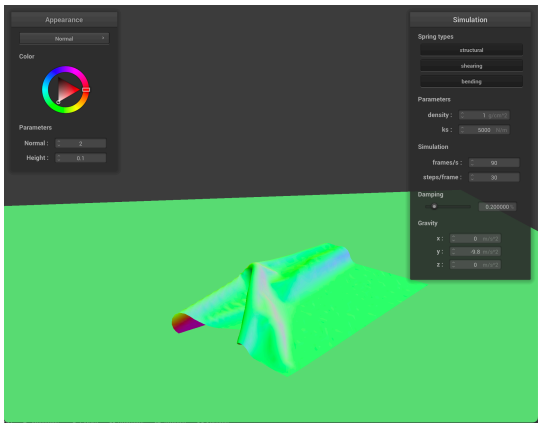
Part IV: Handling self-collisions

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).

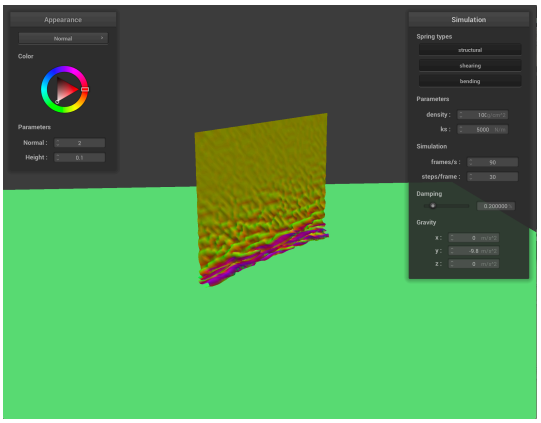


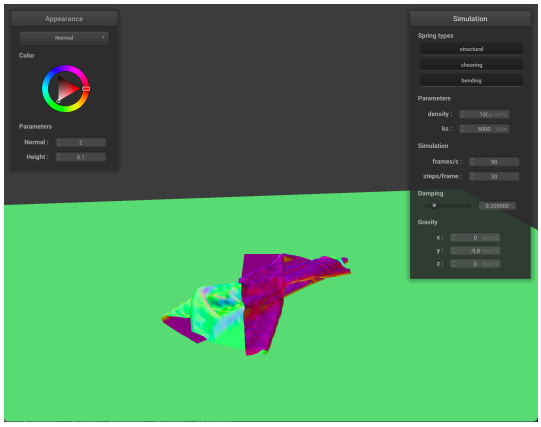
Vary the `density` as well as `ks` and describe with words and screenshots how they affect the behavior of the cloth as it falls on itself.



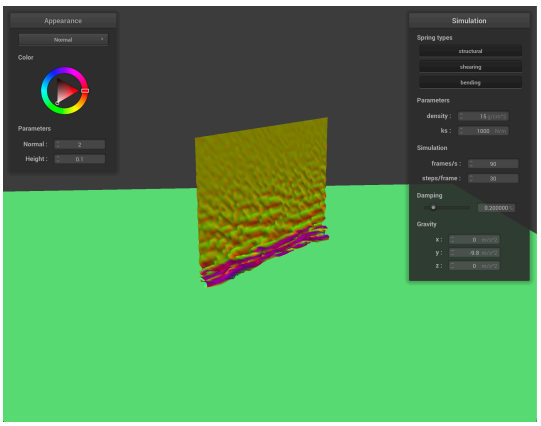


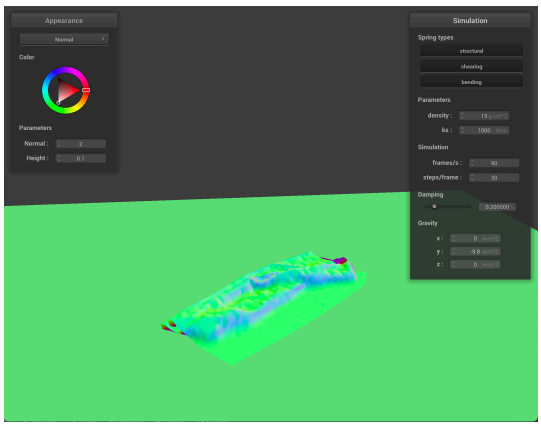
Cloth at density = 1



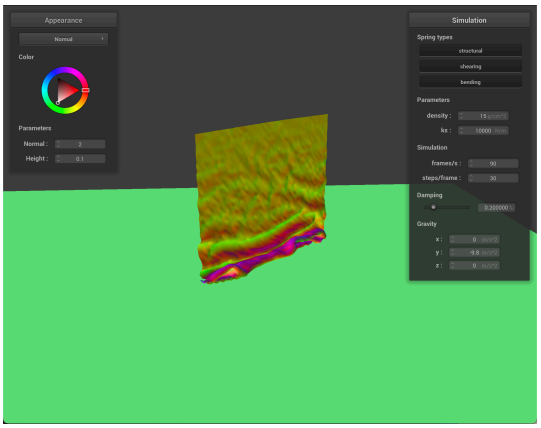


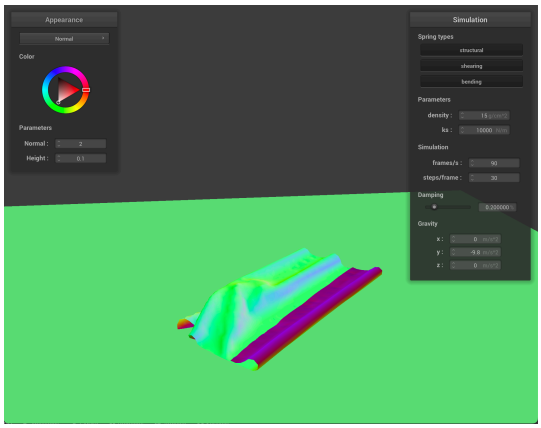
Cloth at density = 100





Cloth at ks = 1,000





Cloth at ks = 10,000

When the density is low, the cloth exhibits smooth folding without small wrinkles. When the density is higher, there are more small fold and wrinkles. When decreasing the spring constant values, it results in a more elastic and bouncy cloth with increased wrinkle formation. Whereas higher spring constants render the cloth more rigid, reducing the occurrence of folds and bounces when falling.

Part V: Shaders

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

Shader programs play a role in the graphics pipeline by functioning as parallel programs executed on the GPU to generate a singular 4-dimensional vector denoting the color at each specific input location. My implemented shaders, scripted in GLSL, comprise vertex shaders and fragment shaders. Vertex shaders primarily handle vertex properties such as position and normal vectors, applying transformations to them. On the other hand, fragment shaders operate on fragments generated post-rasterization, leveraging the geometric attributes of the fragment to calculate the ultimate color value.

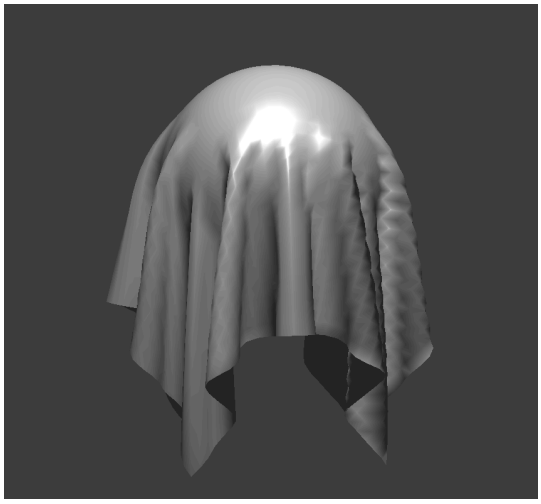
Explain the Blinn-Phong shading model in your own words. Show a screenshot of your Blinn-Phong shader outputting only the ambient component, a screen shot only outputting the diffuse component, a screen shot only outputting the specular component, and one using the entire Blinn-Phong model.

Blinn-Phong shading model encompasses a shading technique that considers three lighting aspects: ambient, which alleviates deep shadows; specular, responsible for reflection highlights and perceived shininess; and diffuse, defining the base color of the object. All three of these components are individually computed and then aggregated to determine the final reflected light value. In this model, several parameters can be adjusted to achieve the desired visual effect on a model. Each component features a constant that, when elevated, enhances its influence on the overall appearance of the model.

Ambient component

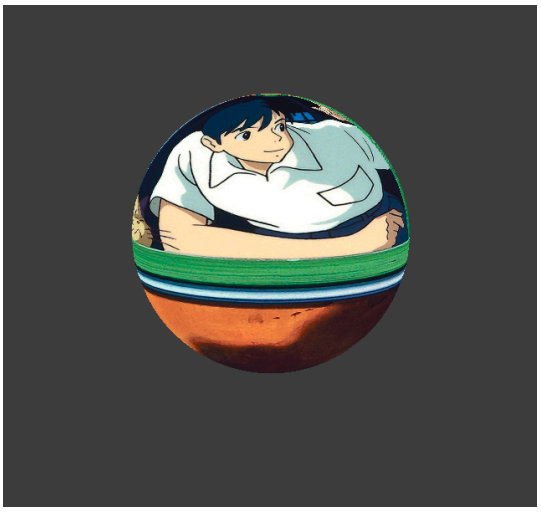
Diffuse component

Specular component

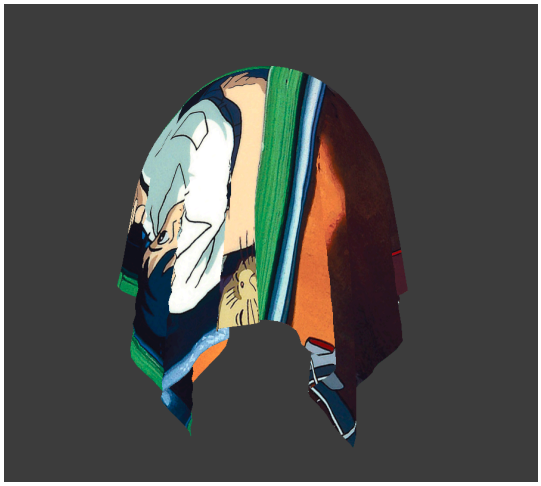


Blinn-Phong

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

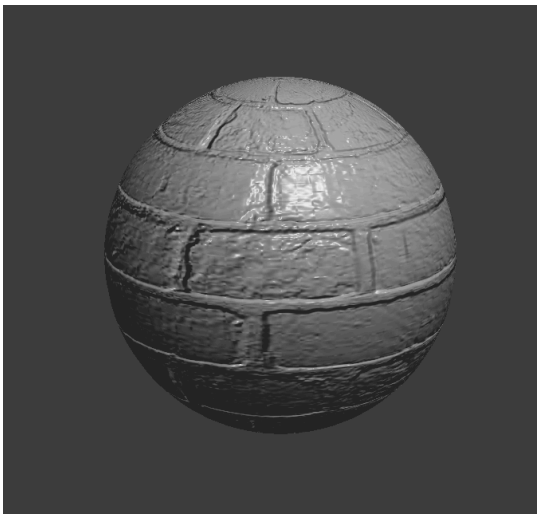


Custom texture



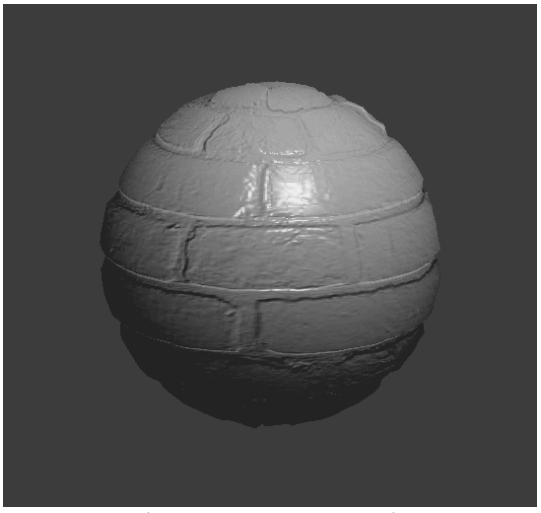
Texture mapped

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`.



Bump mapping on sphere

Bump mapping on cloth



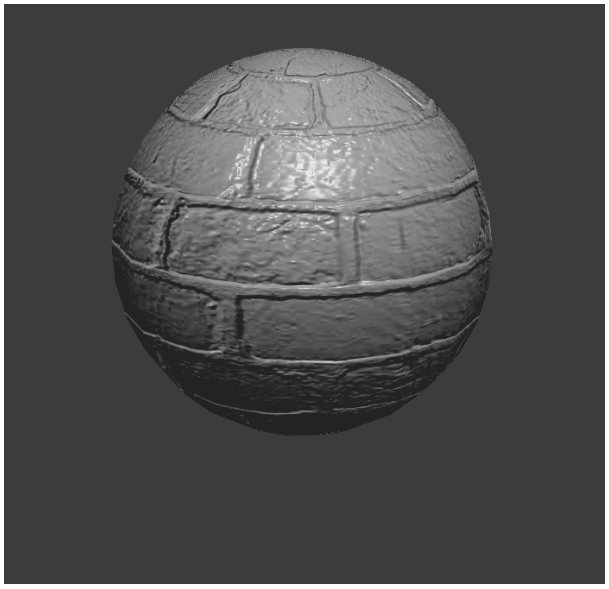
Displacement mapping on sphere

Displacement mapping on cloth

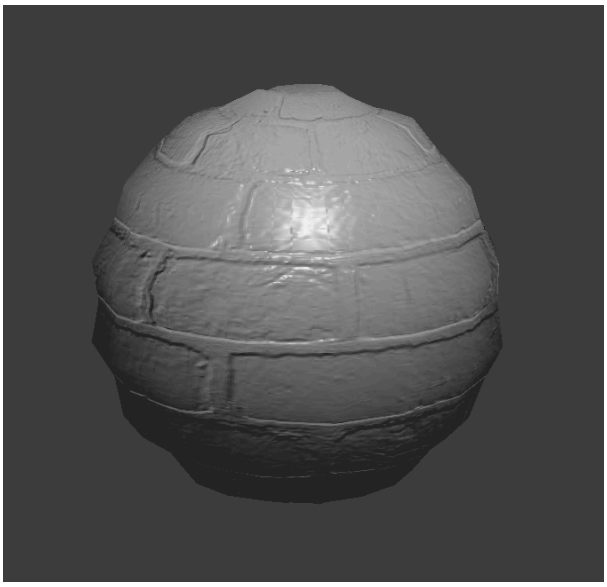
Compare the two approaches and resulting renders in your own words.

Displacement mapping, unlike bump mapping, not only alters the normal vectors of an object but also adjusts the position of vertices. The displacement map modifies the geometry and appearance of the object. It is particularly evident in the cloth, which exhibits a more pronounced texture. On the other hand, bump mapping modifies only the normal vectors to create illusionary details like bumps on the cloth, without affecting the geometry or appearance of the object.

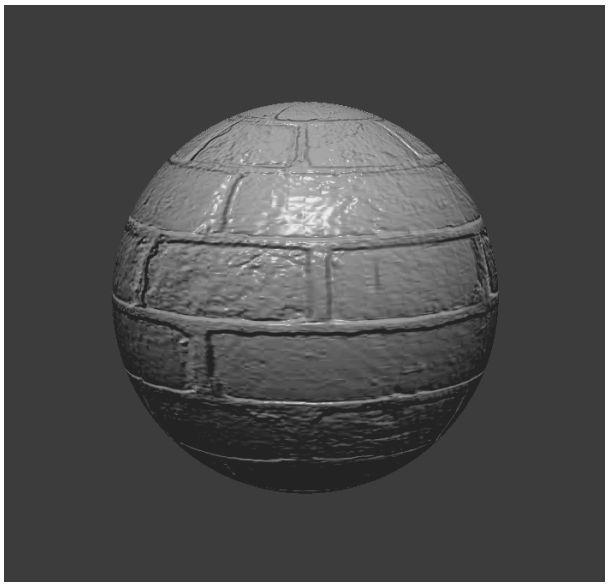
Compare how your the two shaders react to the sphere by changing the sphere mesh's coarseness by using `–o 16 –a 16` and then `–o 128 –a 128`.



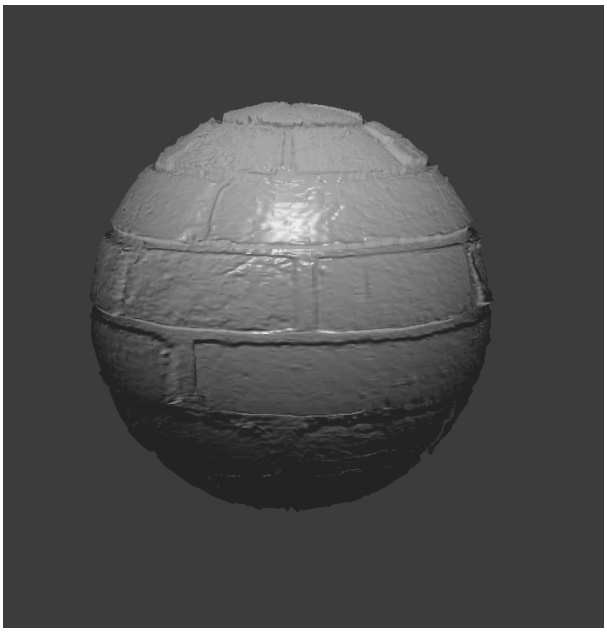
Bump mapping, `–o 16 –a 16`



Displacement mapping, `–o 16 –a 16`



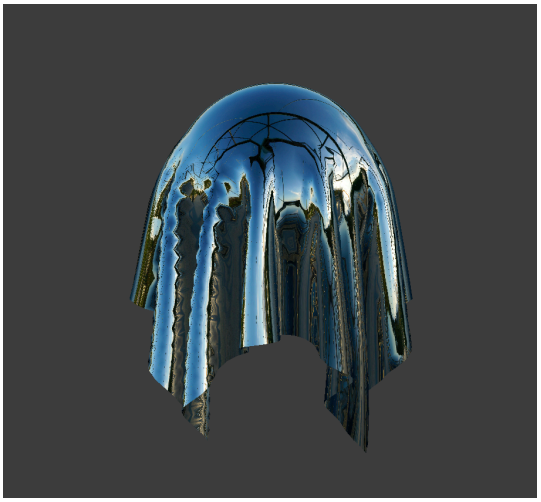
Bump mapping, `–o 128 –a 128`



Displacement mapping, `–o 128 –a 128`

At a resolution of 16x16, there's barely any noticeable contrast between the two spheres under both the bump shader and displacement shader. Yet, at a heightened resolution of 128x128, the sphere's edges manifest a more discernible sharpness enhancement with the application of displacement mapping.

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



Mirror shader on cloth

Mirror shader on sphere