Homework 4: Cloth Sim

URL link:
https://cal-cs184-student.github.io/hw-webpages-sp24-Zzz212zzZ/hw4/hw4%20-%20CS%20184.pdf

Overview:

In this project, we built a dynamic cloth simulation and rendered it with various shading techniques. Starting from modeling the cloth with a mass-spring system, we simulated realistic movements and interactions. We observed changes in the cloth's behavior by tweaking physical properties like spring constants and density. We then added visual complexity through shaders, employing Blinn-Phong shading for light interaction, and texture, bump, and displacement mapping for surface detail. We also explored environment-mapped reflections, giving objects a mirror-like appearance. Each step, from physics to rendering, contributed to a lifelike cloth animation.

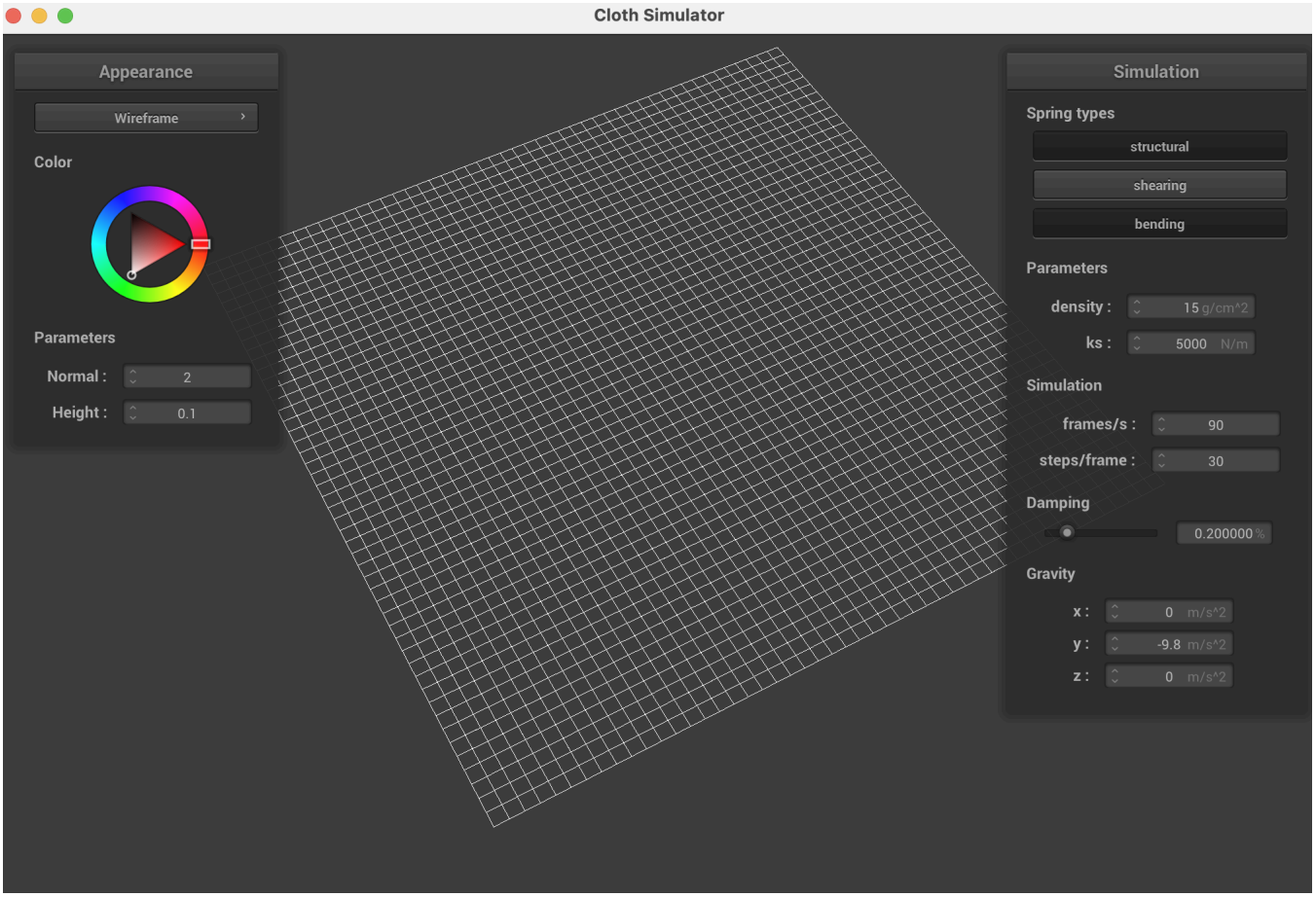
Part 1: Masses and springs

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.

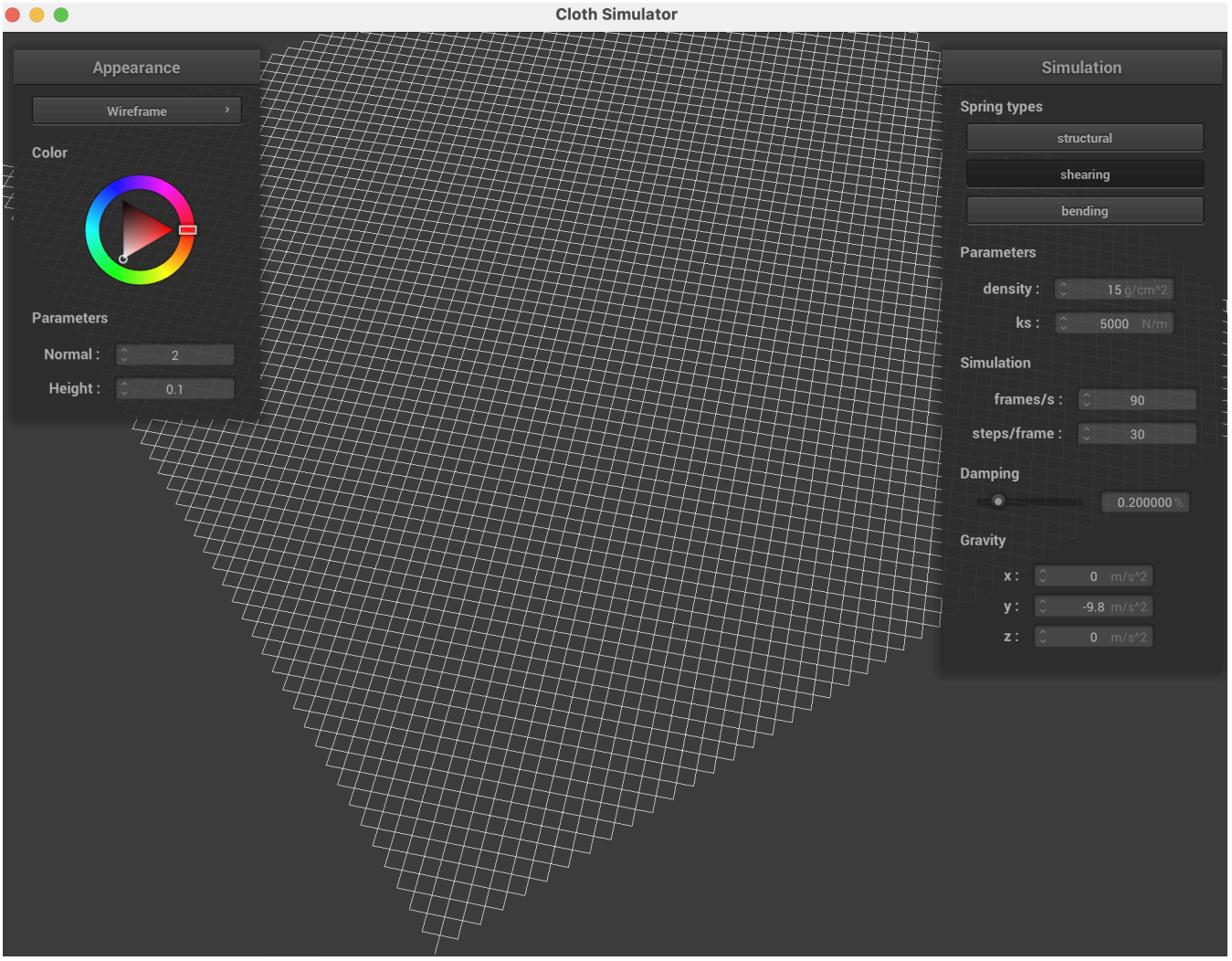
Show us what the wireframe looks like (1) without any shearing constraints, (2) with only shearing constraints, and (3) with all constraints.

Overview: In the Cloth::buildGrid function, we systematically model cloth behavior by creating a grid of point masses, which are anchored in space based on their orientation and designated as "pinned" if required. This setup emulates the structure and constraints of a fabric. We then interconnect these point masses with springs of three types: structural to maintain the cloth's form, shearing to counteract shear deformations, and bending to allow for the cloth's natural flex. This strategic arrangement of masses and springs captures the intricate dynamics of cloth, enabling it to react realistically to forces and interactions within a simulated environment.

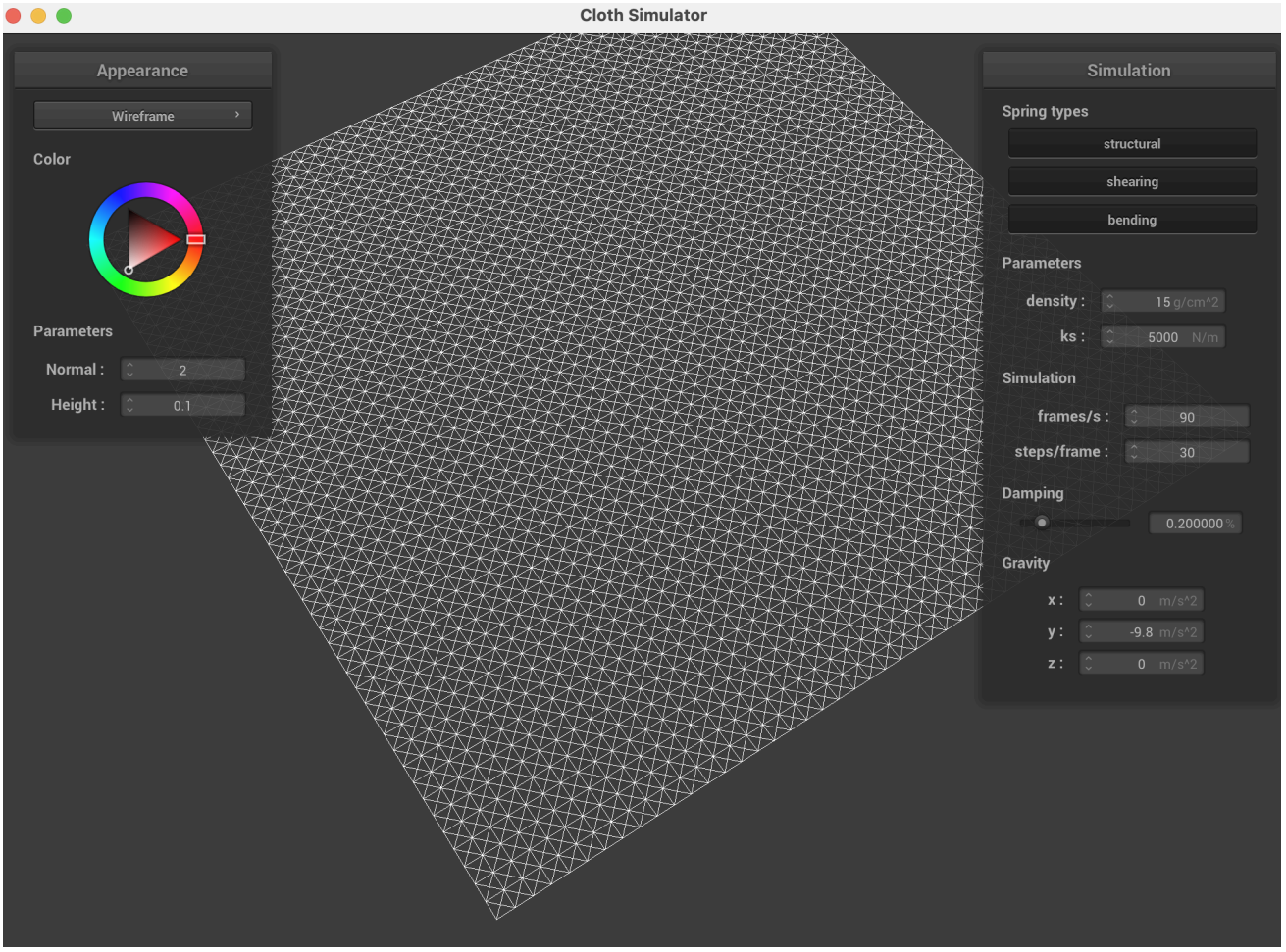
(1) without any shearing constraints



(2) with only shearing constraints



(3) with all constraints

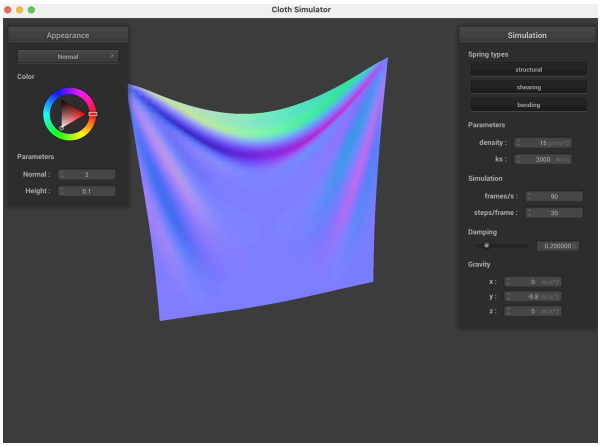


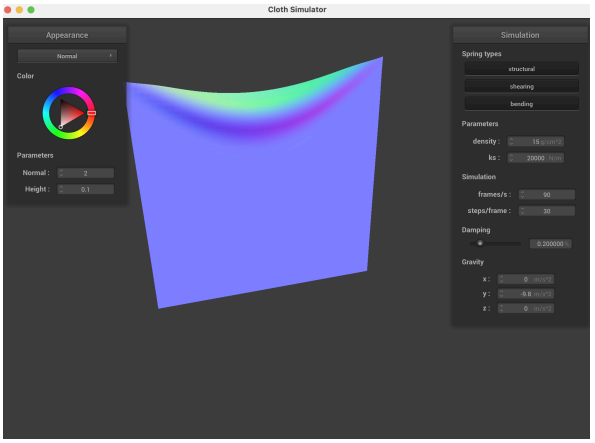
Part 2: Simulation via numerical integration

- 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 the start 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`?

- Low `ks`(2000 N/m)[Left Figure Below]:, the springs are not very stiff, meaning they can stretch more under the same force. The upper middle part drops drastically because the spring is mainly forced by gravity instead of spring correction forces.

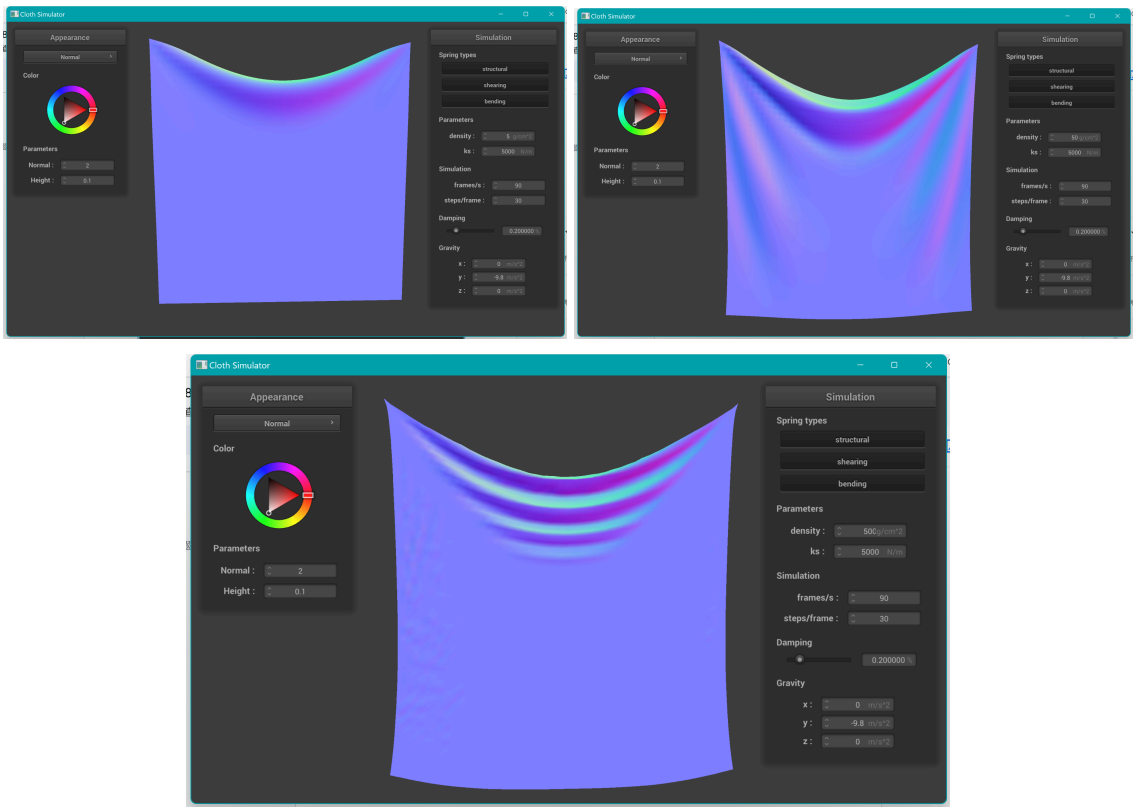
- High `ks`(20000 N/m)[Right Figure Below]: the springs are very stiff, so they resist stretching. This causes the cloth to maintain its shape better against gravity, leading to less stretching. It doesn't droop or sag like the figure above.





    - What about for `density`?
- Higher Density: The cloth will be heavier, and gravity will have a more pronounced effect, making it sag more. It will also have more inertia, so it will be less responsive to forces and take longer to move and stop. Furthermore, with a higher density, we have observed that the mid part has more obvious creases.
- Lower Density: The cloth will be lighter, less affected by gravity, and will have less inertia. This makes it more susceptible to forces, namely gravity, and can lead to more fluttering and rapid movements. Correspondingly, with a lower density, we have observed that the mid part is smoother and doesn't have such obvious creases.

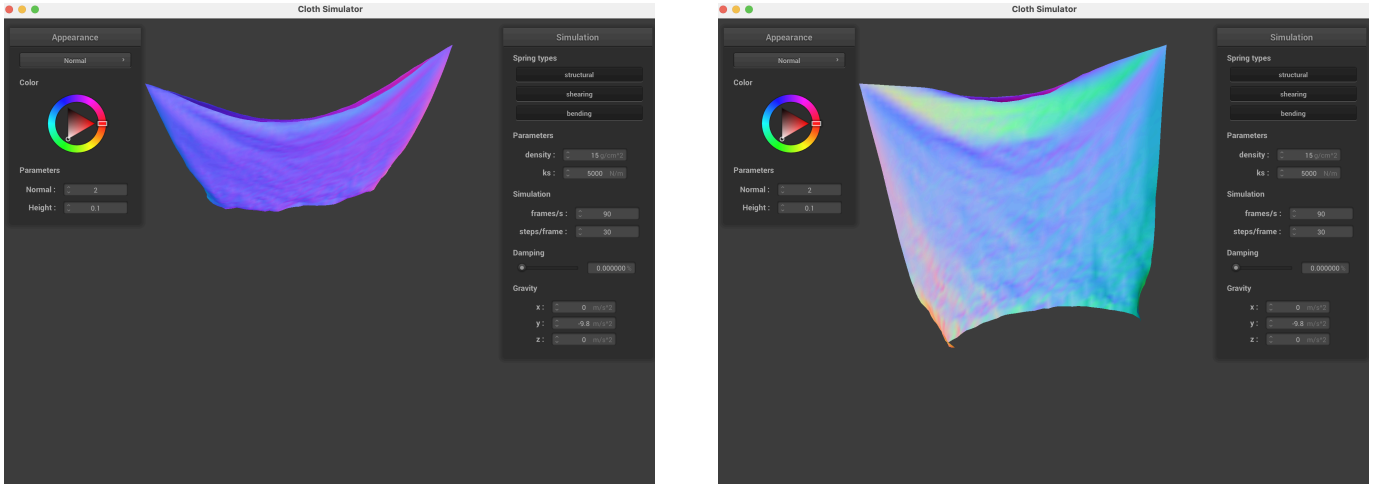
The below figures are configured with densities of 5, 50, and 500, respectively.



- What about for `damping`?

- Low damping

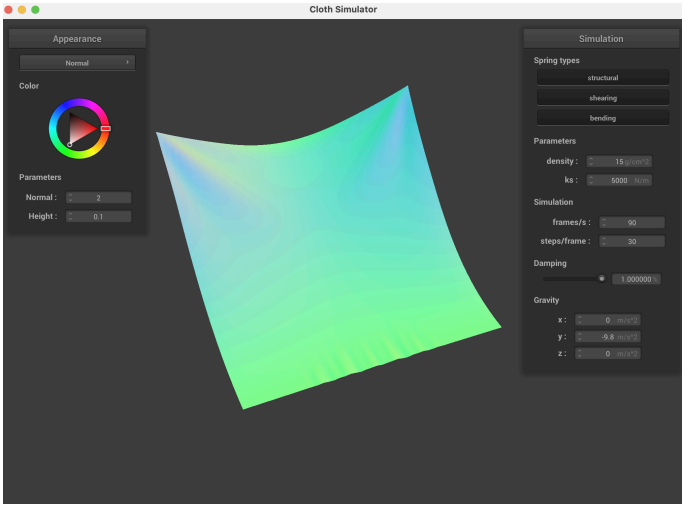
With low damping, the cloth will have very little resistance to motion, so once it starts moving, it will tend to oscillate or sway for a longer time before coming to rest. The lower the damping value, the period from starting to resting takes longer.

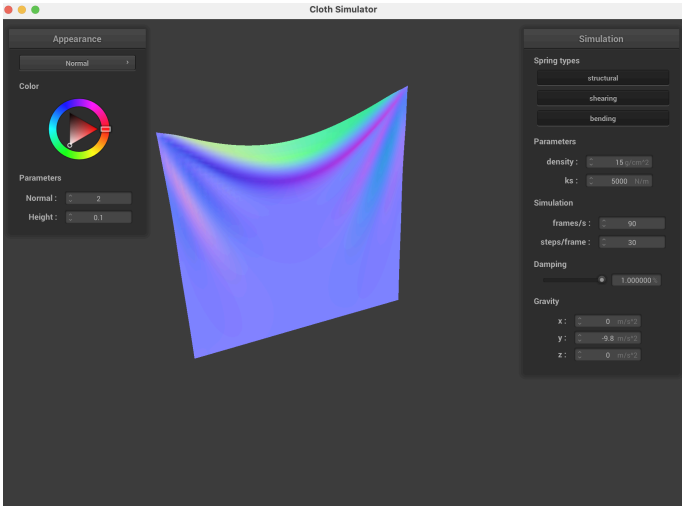


- High damping

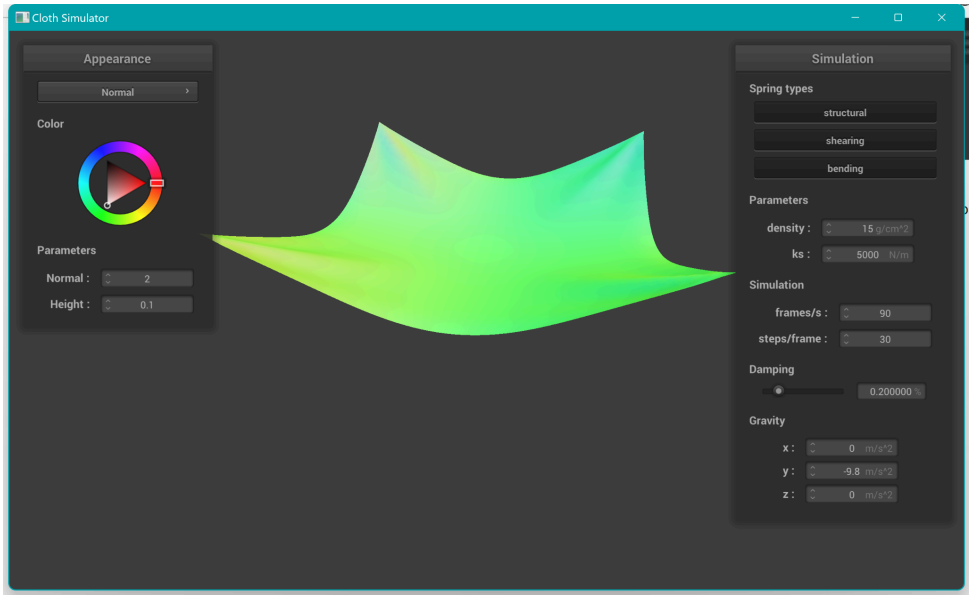
With high damping, the motion of the cloth is quickly dissipated. This means that when the cloth moves, it will settle down to a rest state much faster. High damping

reduces the oscillations and the swaying of the cloth, leading to a steadier and more controlled movement.





- 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.

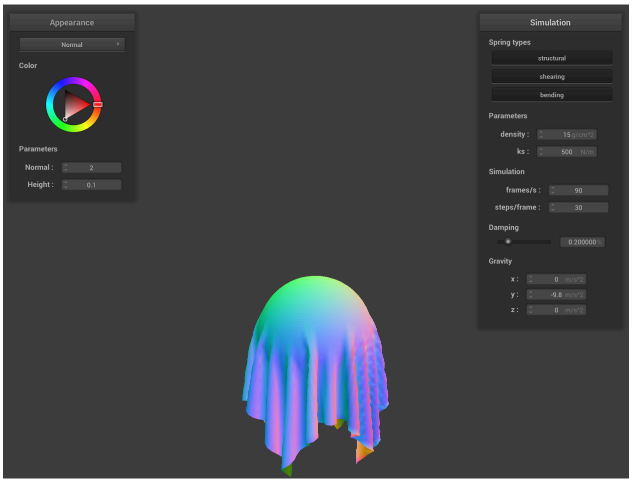


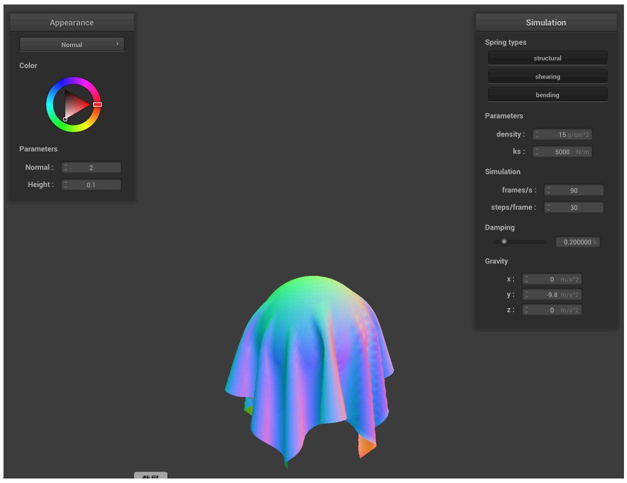
Part 3: Handling collisions with other objects

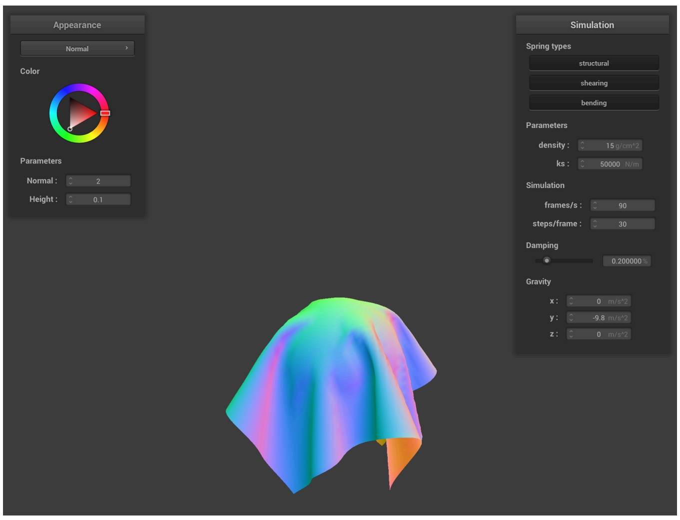
Overview: In this part, we made the cloth in our simulation react to touching other things like spheres and planes. When a part of the cloth moves into one of these objects, we slightly push it back out so it looks like it's resting on or bouncing off the object, just like real cloth would. For the sphere, we make sure the cloth only touches its outer surface, and for the planes, we prevent the cloth from going through. This way, the cloth acts more lifelike in our computer model.

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 [Row 1, Left]: The cloth appears to drape very close to the sphere with obvious folds and creases.
- ks = 5000 [Row 1, Right]: The cloth still shows folds and a draped behavior, but with less obvious sagging than with the lower `ks` value.
- ks = 50000 [Row 2]: At this high spring constant, the cloth barely sags and maintains a shape that implies a much stiffer material.

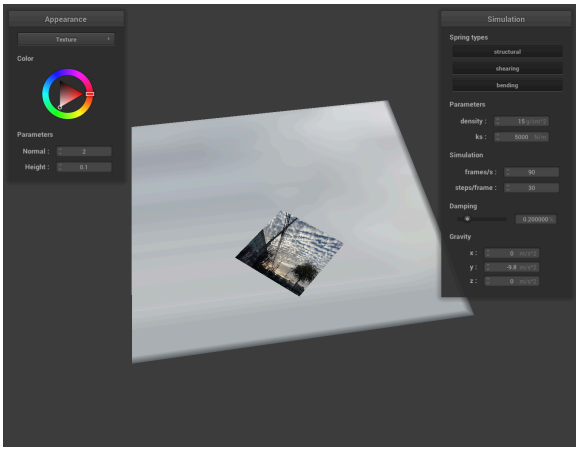


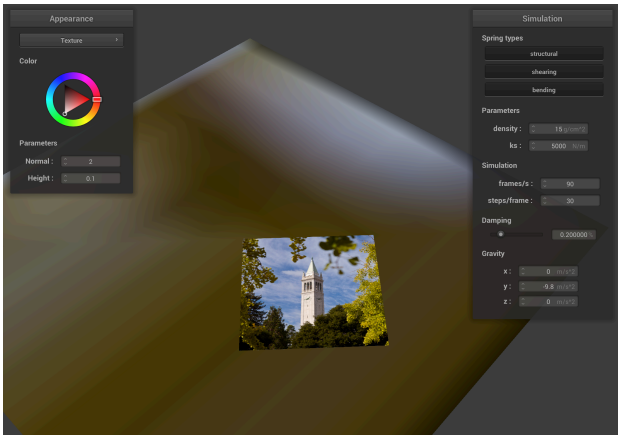




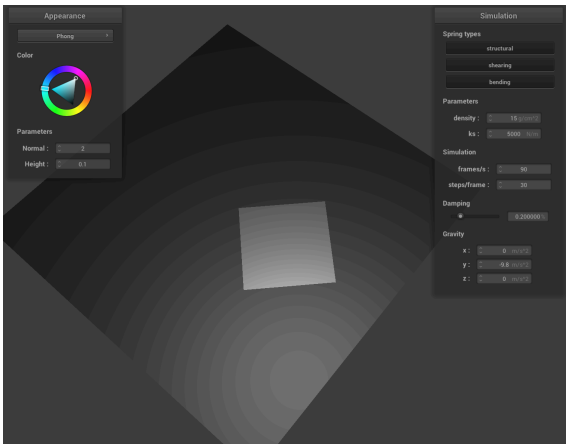
- Show us a screenshot of your shaded cloth lying peacefully at rest on the plane. If you haven't by now, feel free to express your colorful creativity with the cloth! (You will need to complete the shaders portion first to show custom colors.)

Texture mapping





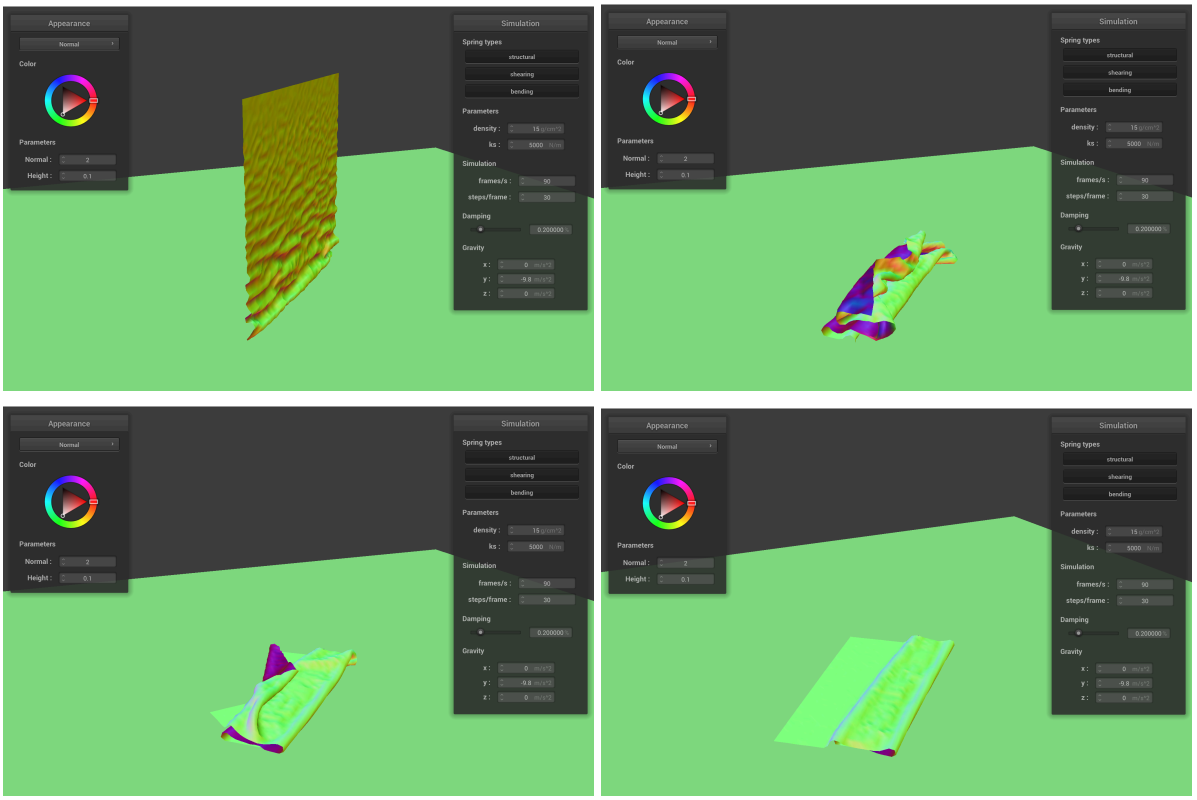
Phong mapping



Part 4: 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).

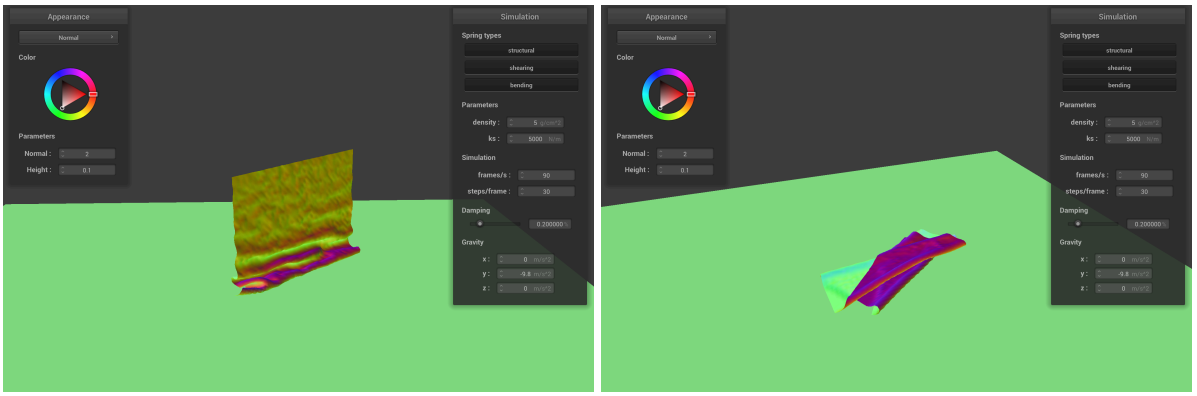
The four pictures below show the falling process, from the point it falls down to its rest state.



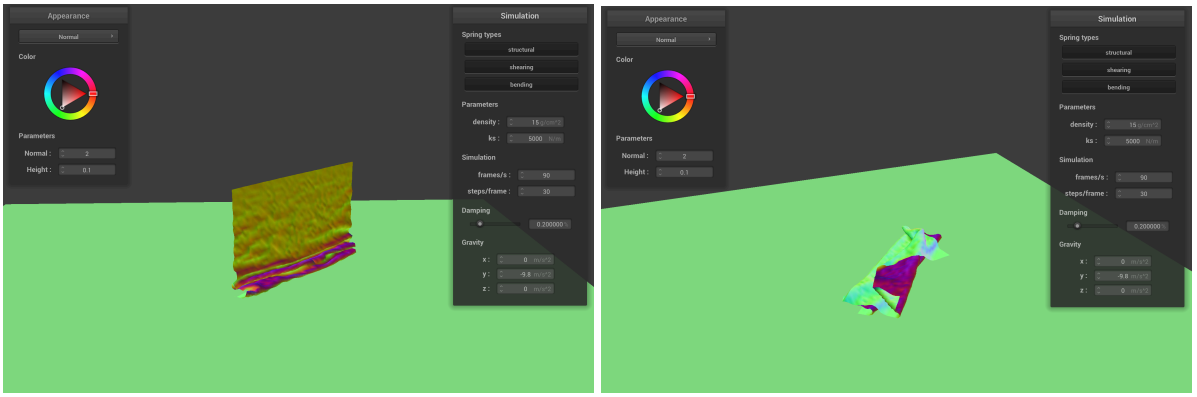
- Vary the `density` as well as `ks` and describe with words and screenshots how they affect the behavior of the cloth as it falls on itself.

Below are three conditions corresponding to low, middle, and high **density**. As the density increases, the cloth becomes heavier, leading to more obvious and sharper folds due to the increased gravitational pull. With higher density, the cloth will also fall faster and exhibit more significant interactions with itself. Conversely, a lower density cloth will fall more slowly and gently, resulting in softer and less defined folds.

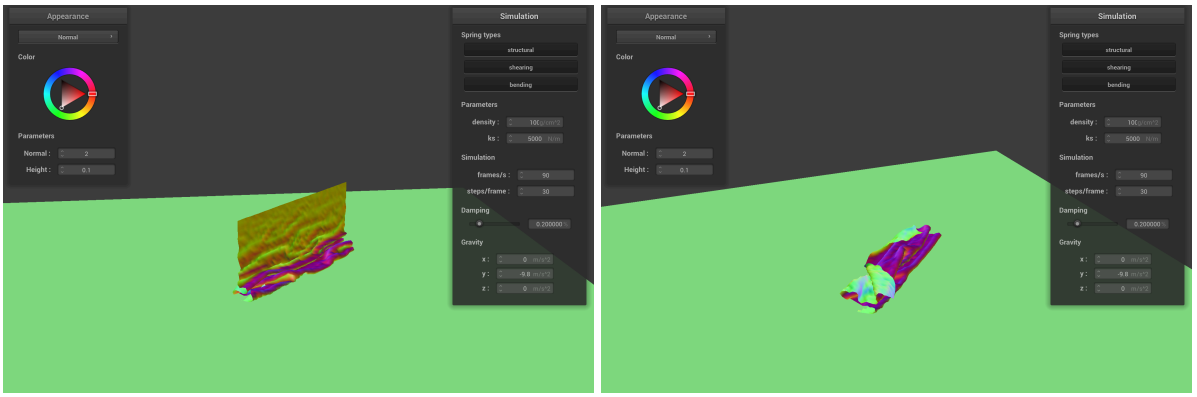
- density = 5



- density = 15

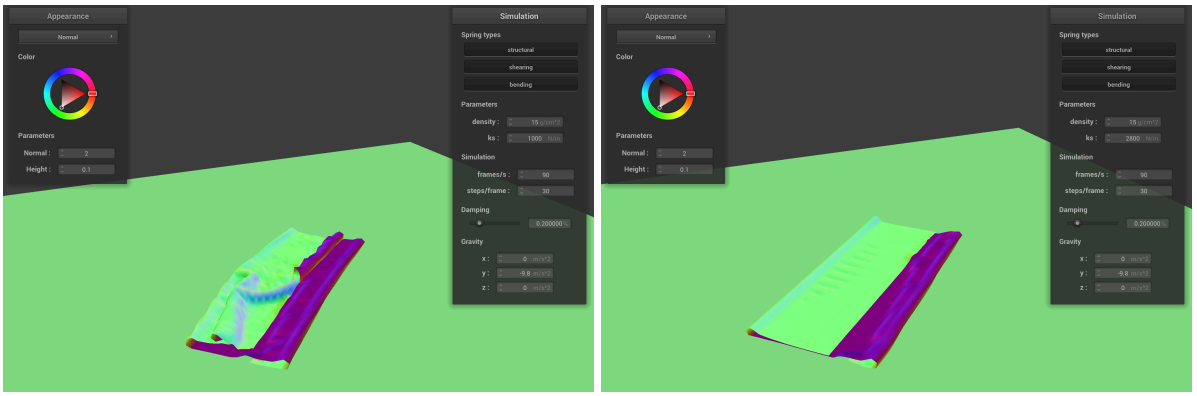


- density = 100

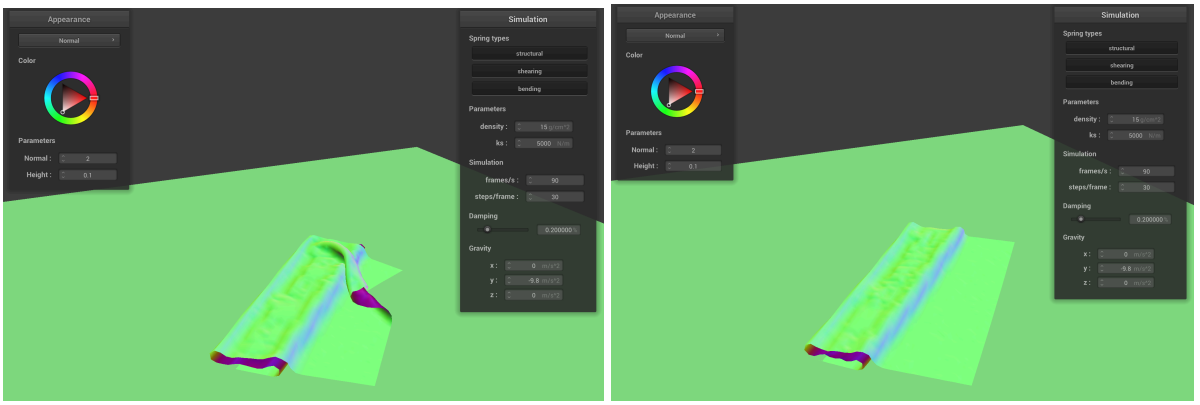


Below are four conditions showing the ks ranging from a low to a high value.The spring constant ks determines the stiffness of the cloth. A low ks makes the cloth more flexible, so it drapes and folds more naturally. A high ks results in a stiffer cloth that resists bending and folding, maintaining a more rigid and obvious shape even when it collides with itself. Another observation in terms of the crease is that while folding, the lower ks has a denser but smaller crease; it is more like cloth made of softer materials. However, these dense creases diminish drastically when the material goes to rest state. On the contrary, a higher ks has a less dense but large and obvious crease, even after the material goes to its rest state.

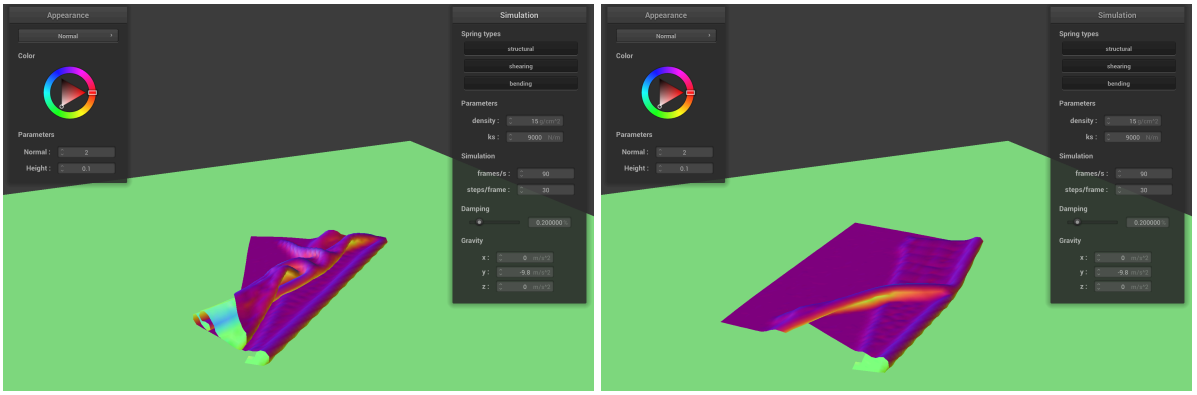
- ks = 1000



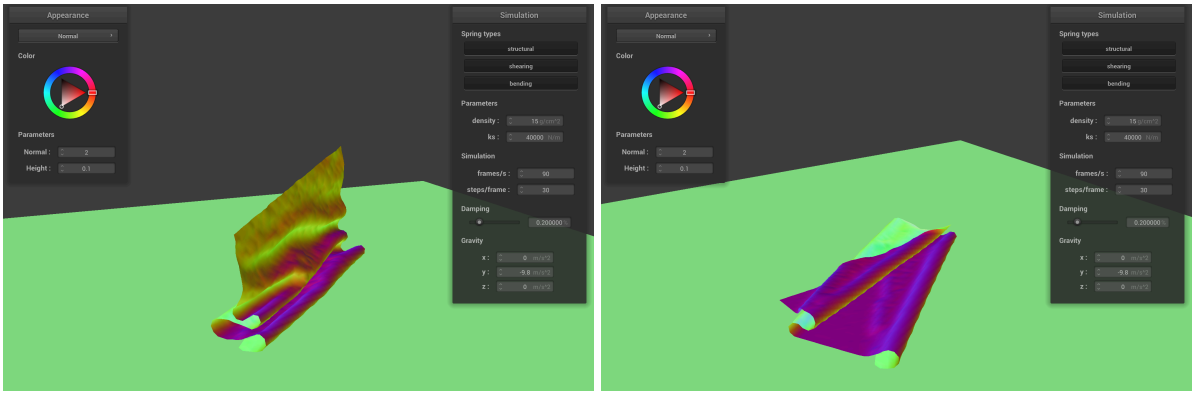
- ks = 5000



- ks = 9000



ks = 40000



Part 5: Shaders (longest, start early!)

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 set of instructions that tells GPU how to render each pixel of a 3D object on the screen. It consists of two main components: a vertex shader (.vert file) and a fragment shader (.frag file).

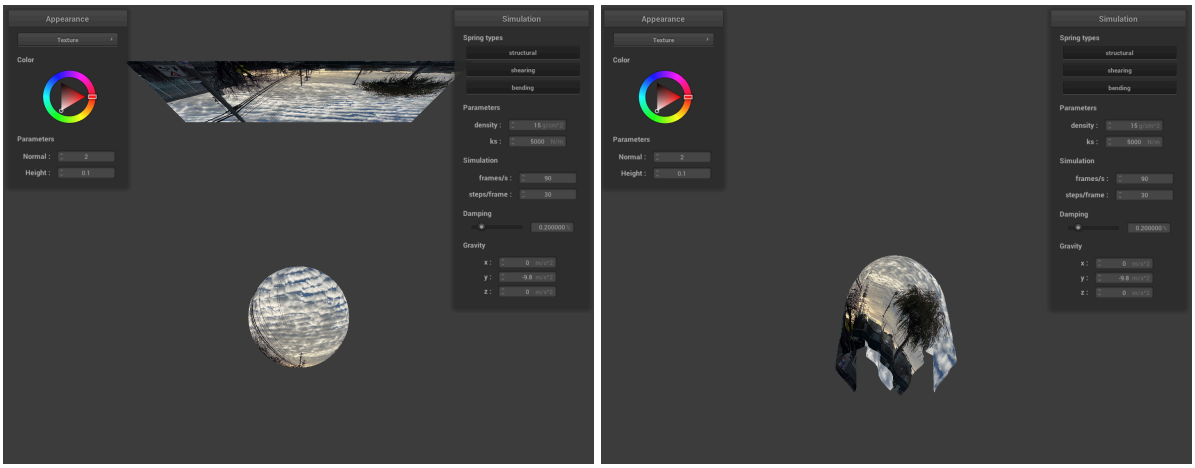
The vertex shader is the first stage in the shading process. It operates on each vertex of a 3D model, performing per-vertex operations including light calculation. The inputs to the vertex shader include attributes of each vertex such as its position, normal vector, texture coordinates, and any other custom attributes needed. The vertex shader can also access uniform variables.

After the vertex shader has run and the graphics pipeline has rasterized the vertices into fragments, the fragment shader takes over. The fragment shader receives interpolated data for each fragment generated from the vertex shader outputs. It uses this data to perform detailed calculations like diffuse and specular lighting, texturing, and applying material properties to determine the final color of each pixel.

- 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 mainly considers three kind of light components: ambient component, diffuse component and specular component. It adds an **ambient light component** that models indirect light scattered in the environment, providing a base color (Left, Row 1). The **diffuse component** (Right, Row 1) accounts for direct light, varying with the angle of incidence to simulate matte surfaces. And this is the same as what we implemented in the `Diffuse.frag` file. The **specular component** (Left, Row 2) simulates the shiny highlights caused by the direct reflection of light, with a shininess factor controlling the spread and intensity of the highlight. When **combined** (Right, Row 2), these components produce a realistic lighting effect on surfaces, capturing both the broad lighting effects and sharp highlights that characterize real-world objects.

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

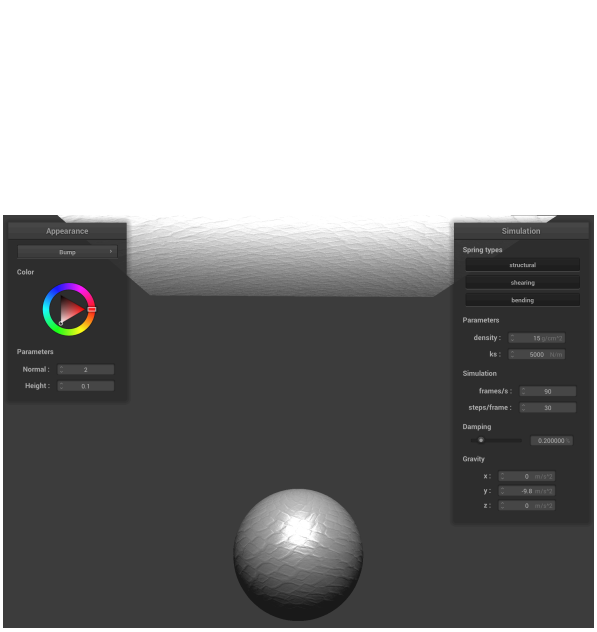


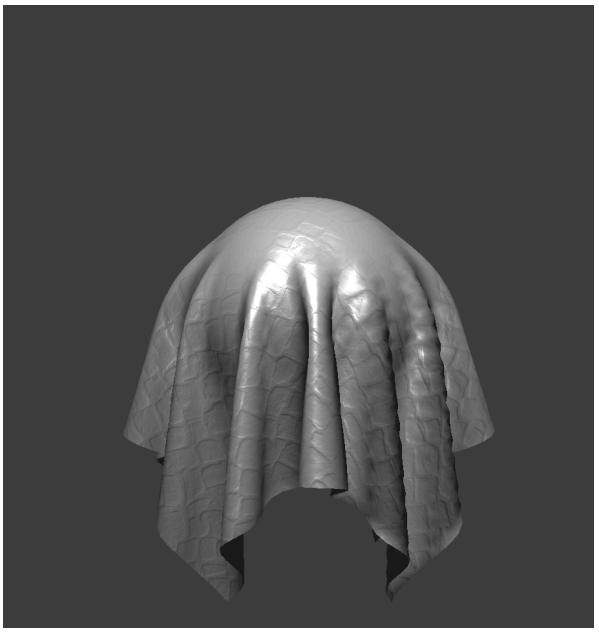
- Show a screenshot of bump mapping on the cloth and on the sphere. Show a screenshot of displacement mapping on the sphere. Use the same texture for both renders. You can either provide your own texture or use one of the ones in the textures directory, BUT choose one that's not the default `texture_2.png`.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

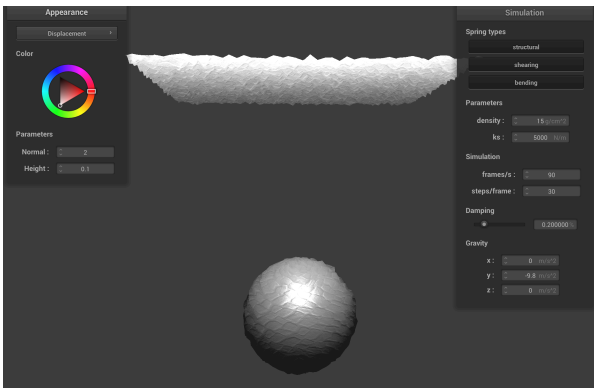
- **Compare** the two approaches and resulting renders in your own words

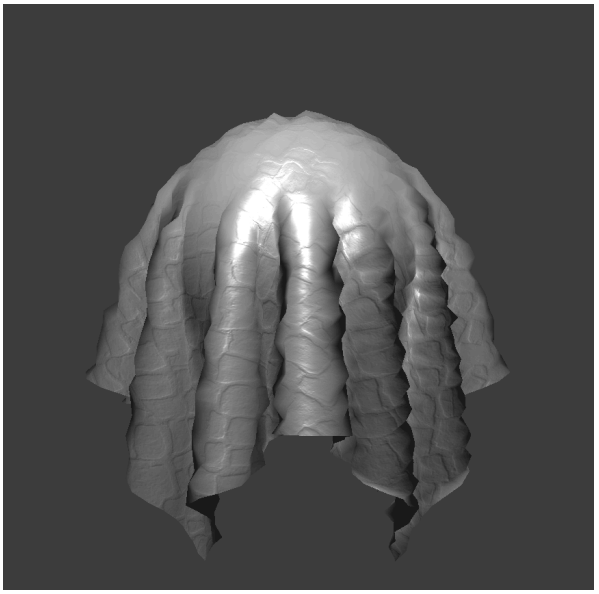
Bump: Bump mapping alters the surface normal during the rendering process based on a texture map. From the results: The cloth's silhouette remains smooth and unchanged, indicating that the geometry has not been displaced.





Displacement: The displacement mapping image shows that the actual geometry of the cloth has been altered according to a height map, which affects both the texture and the profile (silhouette) of the object.

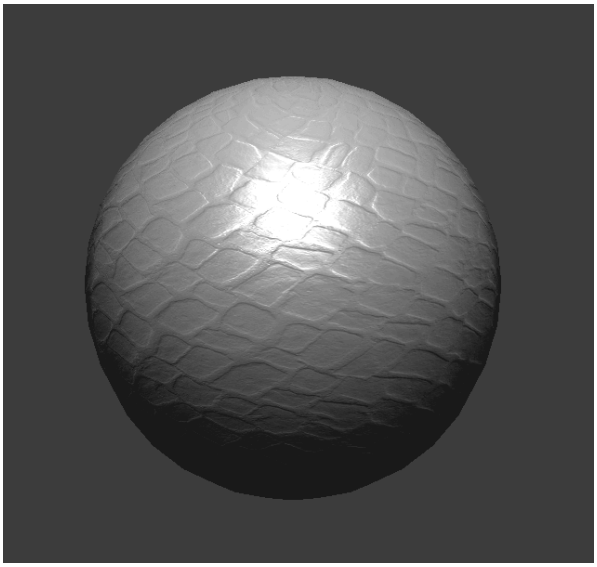




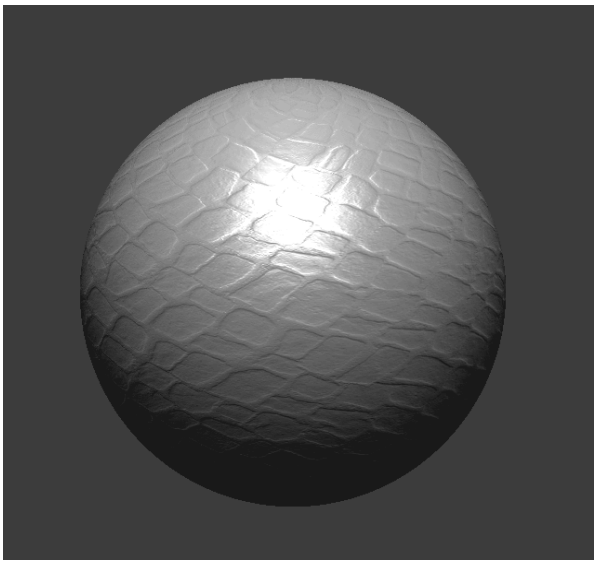
- **Compare how your the two shaders react to the sphere by changing the sphere mesh's coarseness by using -o 16 -a 16 and then -o 128 -a 128.**

For the bump, using lower Accumulate bounces of light (-o) and Samples per batch and tolerance for adaptive sampling (-a) results in a more coarse surface, with a relatively obvious outline (when zooming in, we can notice that the curve is made of some tiny flat edges).

Bump -o 16 -a 16

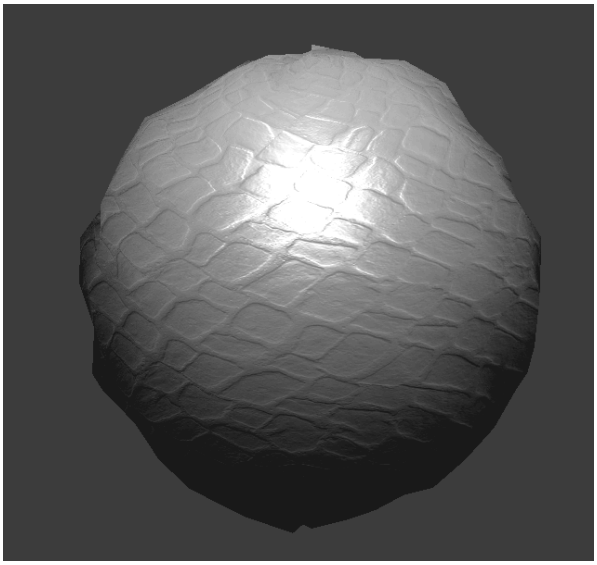


Bump -o 128 -a 128

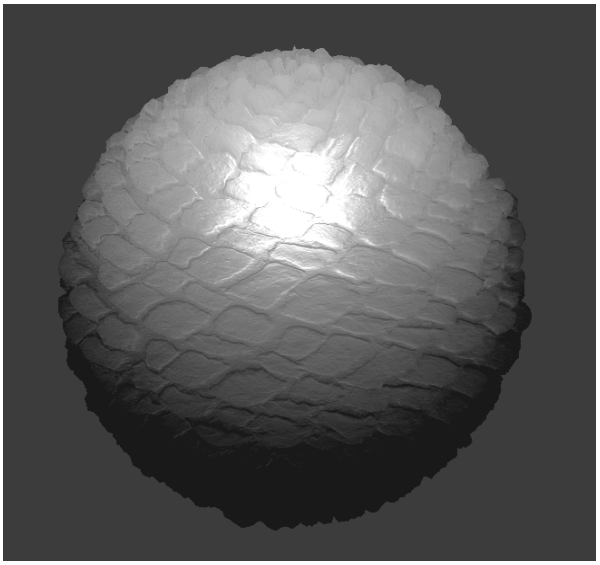


For the displacement, the coarse difference becomes bigger and more obvious. This may be due to a higher sampling rate, resulting in more affected vertices and revealing more ruggedness. As the two pictures below show:

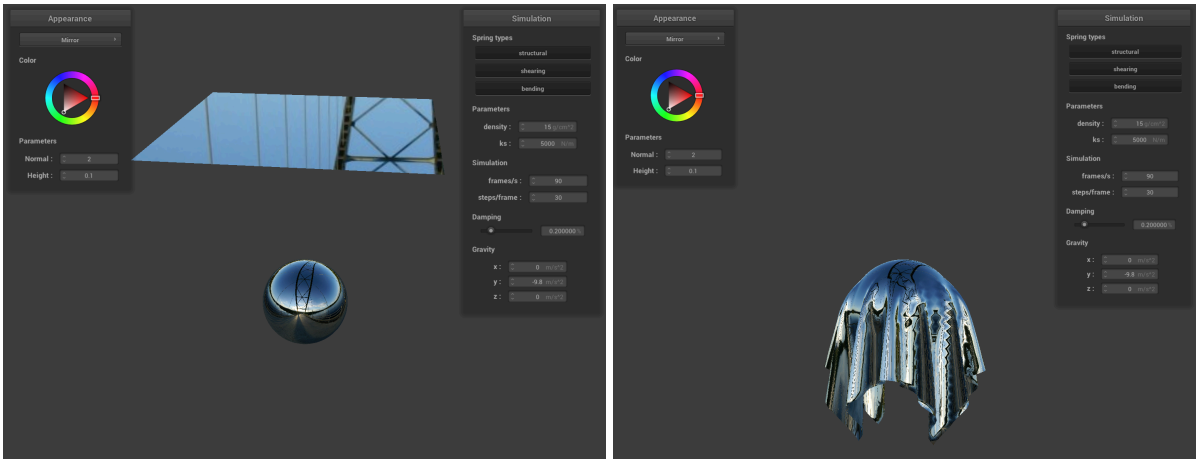
Displacement -o 16 -a 16



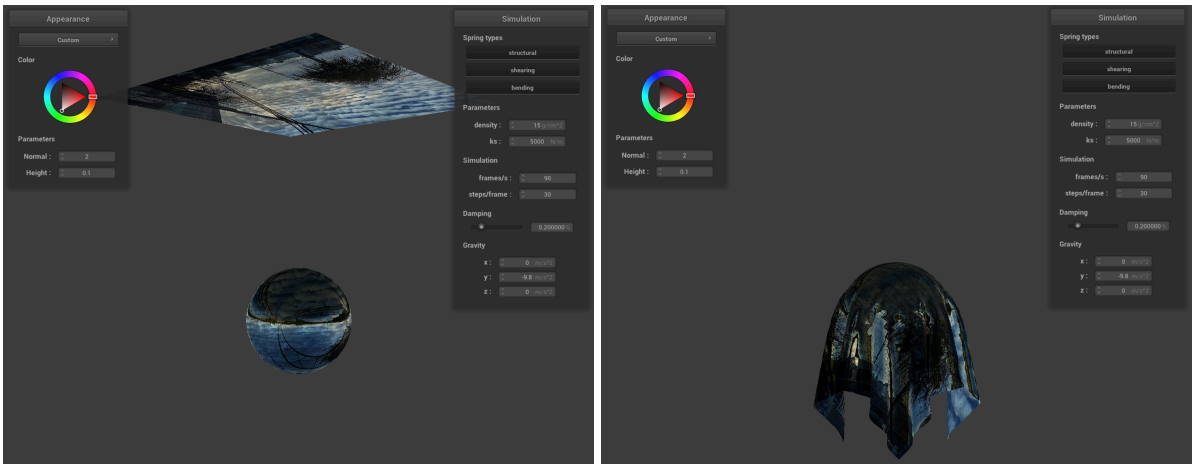
Displacement -o 128 -a 128



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



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



The environment-mapped reflections shader works like a mirror. It takes the camera's view, bounces it off the object's surface, and uses that bounced direction to grab a color from a **360-degree** photo. This makes the object look like it's reflecting the world around it. Unlike bump mapping, which makes a surface look bumpy without changing its shape, this reflection shader makes the object shine like a mirror, showing other parts of the scene on its surface. The environment-mapped reflections provide a mirrored surface by reflecting the environment on the object, whereas mirror mapping involves reflecting other objects and scene elements directly, often with dynamic updates for movement within the scene.