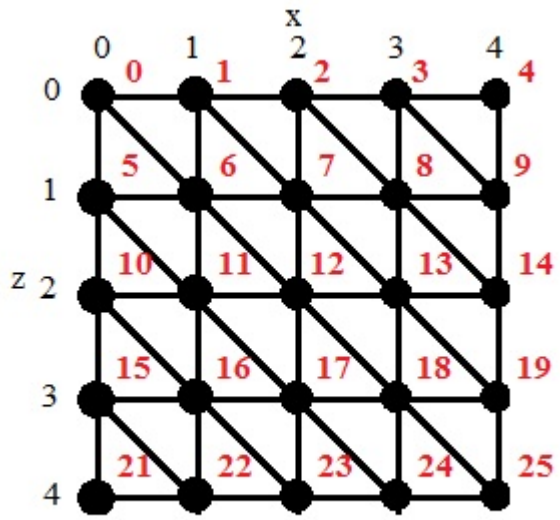
Com S 336
Fall 2020
Homework 5

Please submit an archive on Canvas including the files indicated at the beginning of each problem.

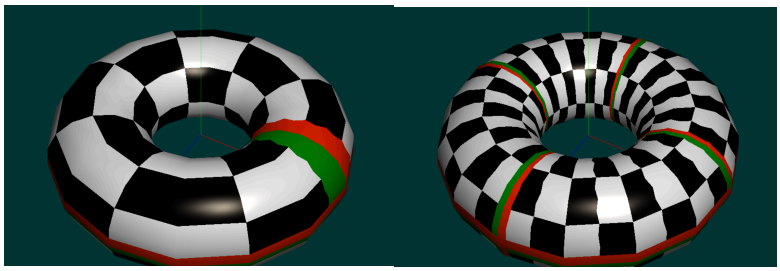
1. (*Please turn in your modified version of Torus.js*.)

The file **`examples/homework5/Torus.js`** is similar to HeightMap.js in that it mathematically creates a simple model with vertices, normals, and indices for mesh or wireframe rendering. You can see what it looks like out of the box by opening **`TorusTest.html`**. This part is very similar to (and reuses much of the code from) the HeightMap example. To see what's going on, take the same picture:



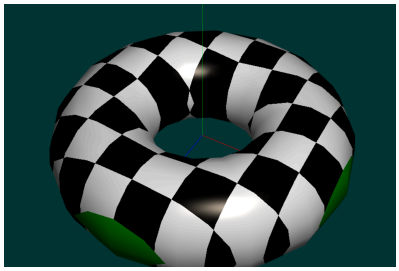
Then imagine that the vertices on the left are geometrically the same points as the ones on the right (0 and 4, 5 and 9, 10 and 14, etc.). This makes it "tubular". Similarly, the vertices on the top are the same points as the ones on the bottom, which allows it to connect to itself and make the doughnut shape. It is generated like a solid of revolution from calculus: the initial points (e.g. vertices 0 through 4 in the diagram) are actually just a circle in the x-y plane, translated in the x-direction by some radius. Then the whole thing is revolved about the y axis. At each segment of the angle of revolution, another row of vertices is generated. Finally the bottom row is just the initial points (top row) again. The code for generating vertices and normals and indices is all done.

Your task is to complete the two methods for generating texture coordinates for this model. The first method is the more obvious of the two: it just maps the grid above into (s, t) values. Again referring to the diagram, vertex 0 would have texture coordinates (0, 0), vertex 1 would have texture coordinates (.25, 0), vertex 2 would have texture coordinates (.5, 0), vertex 3 would have texture coordinates (.75, 0), and vertex 4 would have coordinates (1, 0). For the vertical we also allow the texture to possibly repeat, so instead of ranging from 0 to 1, we allow the t coordinate to range from 0 to *reps* (number of repetitions) which is a parameter to the Torus constructor. Shown below is the torus with reps = 1 on the left and reps = 4 on the right:



These images come from running TorusTest2.html after implementing the method **`generateTexCoords`**.

The other method, **`generateTexCoordsParallel`**, is simpler. It's just a parallel projection in the y direction. That is, the x values are in the range [-(radius + innerRadius), radius + innerRadius], so we map x to an s value in [0, 1], ignoring y. Same for the z values. The result would look like this:

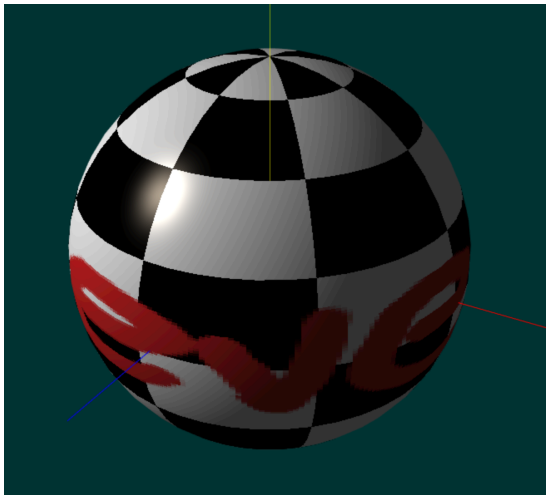


2. (*Please turn in your modified versions of LightingWithTextureForHW5.html* and *LightingWithTextureForHW5.js*)

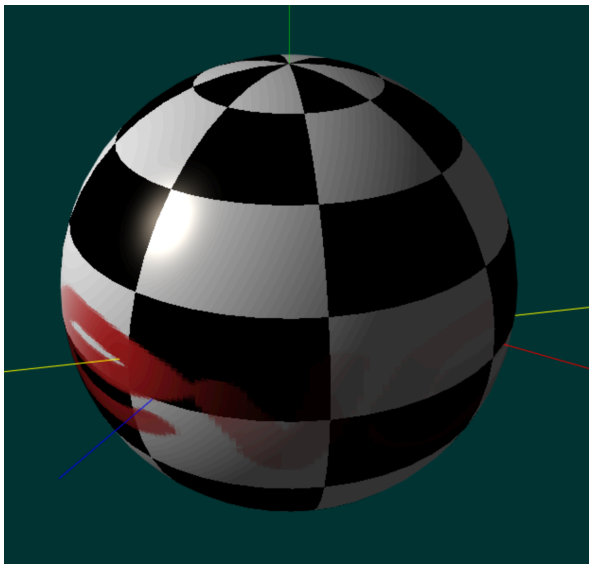
a) **`examples/homework5/LightingWithTextureForHW5.html`** is similar to the previous example LightingWithTextureAndModel but it has the spot light direction and controls

from homework 4 built into it. However, the spot light data is not currently used for anything except to draw the yellow line indicating the spot direction. (The light that is actually used for lighting the scene is in a fixed position.). Out of the box, it is set up to use an image with a transparent background and "decal" it over the existing surface color.

a) Modify the application so that it has two textures. Use one for the surface color, and one for the "decal" effect. Using a checkerboard for the second texture, it would look like this, for example:



b) This part creates a cool effect. Imagine there is secret writing on the surface of an object that can only be seen by shining an ultraviolet light or some special wavelength on it. You have a special flashlight (the spot light) that shines the required wavelength, so within the beam of this flashlight, the writing becomes visible. Otherwise, the flashlight has no effect on the scene, since the "light" it emits is not visible light. It might look like this (note the direction of the yellow line, indicating what would be the spot direction).



You can also check out this video:
http://web.cs.iastate.edu/~smkautz/cs336f20/homework/hw5/animation2.mp4

The effect is surprisingly easy: In the fragment shader, compute two possible fragment colors: one with the writing, one without, and blend them using the spot factor. (Note that the vector pointing to the spotlight position and the vector for the spot light direction are already available in the fragment shader, but are currently unused.)

3. (*Please turn in two files named turbine.html and turbine.js.*)

Create a hierarchical scene with three.js, using the basic techniques of HierarchyWithTree3.js. Your basic task is to create a clunky model of a wind turbine using five simple objects. Parts should include

- a vertical shaft, or tower,
- a generator housing at the top of the shaft,
- a rotor hub, and
- two rotor blades.

The housing should turn on the vertical shaft (the "yaw" control to make it face the wind), the rotor should rotate, and the blades themselves should rotate relative to the rotor to adjust the "pitch" of the blades (angle relative to the rotor). Animate the rotor to rotate continuously, and add some keyboard controls for yaw and the blade pitch.

Instead of using scaled cubes for every part, as we did in HierarchyWithTree3.js, explore some of the other options for three.js geometries.

Below is a basic diagram. (Obviously you should ignore internal details such as the "Generator" and "Rotor brake".)