CS460 Fall 2020
Github Username: yylhyyw
Due Date: 09/30/2020

Assignment 3: Three.js Cubes ... and other geometries

We will use Three.js to create multiple different geometries in an interactive fashion.

In class, we learned how to create a `THREE.Mesh` by combining the `THREE.BoxBufferGeometry` and the `THREE.MeshStandardMaterial`. We also learned how to *unproject* a mouse click from 2D (viewport / screen space) to a 3D position. This way, we were able use the `window.onclick` callback to move a cube to a new position in the 3D scene. Now, we will extend our code.

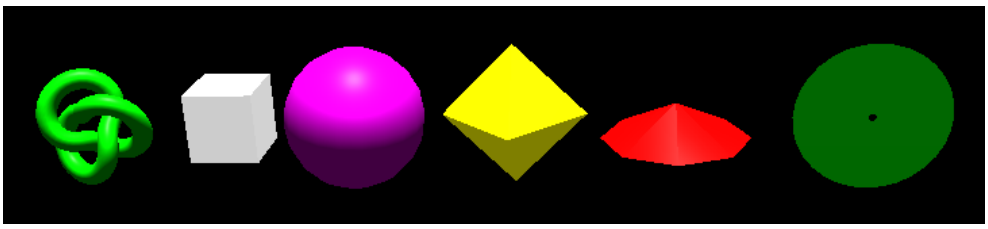
The goal of this assignment is to create multiple different geometries by clicking in the viewport. This means, rather than moving an existing mesh, we will create new ones in the `window.onclick` callback. On each click, our code will randomly choose a different geometry and a random color to place the object at the current mouse position.

We will be using six different geometries. Before we start coding, we want to understand their parameters. Please complete the table below. You can find this information in the Three.js documentation at `https://threejs.org/docs/` (scroll down to Geometries). In most cases, we only care about the first few parameters (**please replace the Xs**).

Constructor	Parameters
THREE.BoxBufferGeometry	(width, height, depth)
THREE.TorusKnotBufferGeometry	(X, X, X, X)
THREE.SphereBufferGeometry	(X, X, X)
THREE.OctahedronBufferGeometry	(X)
THREE.ConeBufferGeometry	(X, X)
THREE.RingBufferGeometry	(X, X, X)

Please write code to create one of these six geometries with a random color on each click at the current mouse position. We will use the `SHIFT`-key to distinguish between geometry placement and regular camera movement. Copy the starter code from `https://cs460.org/shortcuts/08/` and save it as **03/index.html** in your github fork. This code includes the `window.onclick` callback, the `SHIFT`-key condition, and the `unproject` functionality.

After six clicks, if you are lucky and you don't have duplicate shapes, this could be your result:



Please make sure that your code is accessible through Github Pages. Also, please commit this PDF and your final code to your Github fork, and submit a pull request.

Link to your assignment: `https://yylhyyw.github.io/cs460student/03`

Bonus (33 points):

Part 1 (5 points): Do you observe Z-Fighting? If yes, when?

Z-Fighting will occur when two or more geometries using same space, but in this case it is very easy to find Z-Fighting when there are two or more intersecting of the same geometries because they have same length in Z-axis. In the case that when there are two or more intersecting of the different geometries, it is harder to find Z-Fighting because different geometries have different length in Z-axis, the geometry has the longer length will cover the Z-Fighting phenomenon.

Part 2 (10 points): Please change `window.onclick` to `window.onmousemove`. Now, holding `SHIFT` and moving the mouse draws a ton of shapes. Submit your changed code as part of your `03/index.html` file and **please replace the screenshot below with your drawing**.

Part 3 (18 points): Please keep track of the number of placed objects and print the count in the JavaScript console. Now, with the change to `window.onmousemove`, after how many objects do you see a slower rendering performance?

Because of my computer has tons of GPU power, it hard for me to see a slower rendering performance by eyes. Instead, I used Spector.js `https://github.com/BabylonJS/Spector.js#cdn` that introduced in the class to monitor the rendering performance and assume that from 120fps to 35fps will make me to see a slower rendering performance. at initial, it has 120fps and after I draw 6600 objects on the scene it reduced to average 35fps.

What happens if the console is not open during drawing?

If I do not open the console while drawing it gives me a better performance that it starts at 120fps and drops to 35fps when I draw around 6800 objects. But the most noticeable is that the fps during drawing is more stable that the float in the fps gets smaller. I think the better performance is because the browser does not need to show the console log strings so that it make drawing smoother with the same idea that system.out.println will take a lot system resource in Java.

Can you estimate the total number of triangles drawn as soon as slow-down occurs?

After doing research online I found that three.js has its function to get the numbers of triangles that are used in render called renderer.info.render.triangles I used this function in chrome developer console, and it returns 18233496 triangles in 6711 objects with average 35fps on my machine.
There is the reference: https://stackoverflow.com/questions/40274612/is-there-a-way-to-get-the-number-of-polygons-in-a-scene