CS460 Fall 2020
Github Username: EncryptedCurse
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.

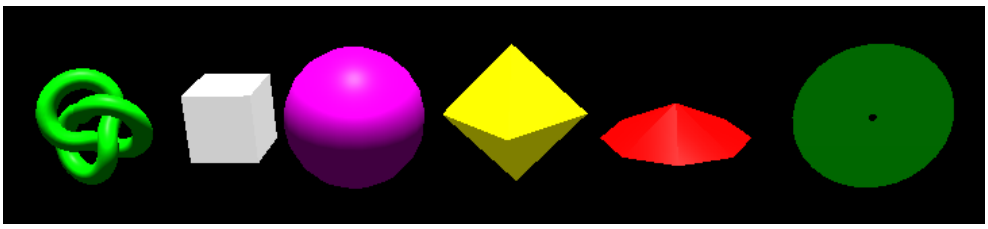
The goal of this assignment is to create multiple different geometries by clicking in the viewport. This means, rather than moving an existing mesh, we will create new ones in the `window.onclick` callback. On each click, our code will randomly choose a different geometry and a random color to place 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	(radius, tube, tubularSegments, radialSegments)
THREE.SphereBufferGeometry	(radius, widthSegments, heightSegments)
THREE.OctahedronBufferGeometry	(radius)
THREE.ConeBufferGeometry	(radius, height)
THREE.RingBufferGeometry	(innerRadius, outerRadius, thetaSegments)

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://encryptedcurse.github.io/cs460student/03/`

Bonus (33 points):

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

Yes, I observed Z-fighting. I first noticed it using `RingBufferGeometry` when rings would overlap, since the renderer wouldn't know which one to render on top. However, I decided to switch to `TorusBufferGeometry` to give the ring more depth (since it was initially a flat disc). I also saw it on overlapping `BoxBufferGeometry` objects, but I alleviated the issue by generating slight differences in the Z-component for each new instance.

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?

On my computer, I started noticing a slowdown after about 2,600 shapes. I also closely monitored GPU, CPU, and memory usage using Chrome's built-in task manager. When I reached over 6,000, that single tab had eaten up over 1 GB of memory and the GPU process was consistently pegged at 80-100% even after I stopped generating new shapes. Moreover, at its highest point, the developer tools alone were taking up 20% of the CPU.

What happens if the console is not open during drawing?

Even without developer tools open, I didn't notice any significant difference. By the time I noticed a slowdown and checked the console for the count, it was still at roughly 2,600 shapes as before.

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

Looking through `three.js` documentation, I found that you can use `renderer.info.render.triangles` to obtain the exact number of triangles drawn. At 2,600 shapes, my scene had 1,413,548 triangles.

The second time, I calculated the number of triangles manually. I first checked how many triangles each individual geometry using my particular arguments contains. I found that it takes: 12 for a cube; 420 for a sphere; 1920 for a torus knot; 8 for an octahedron; 60 for a cone; and 800 for a torus ring. I then created an array that would keep track of how many of each individual shape there is. Combining this data, I found that this particular scene of 2,600 shapes had 1,369,736 triangles, which is fairly close to the previous run.