CS460 Fall 2019
Name: Jared Barresi
Student ID: 00974358
Due Date: 09/23/2019

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.

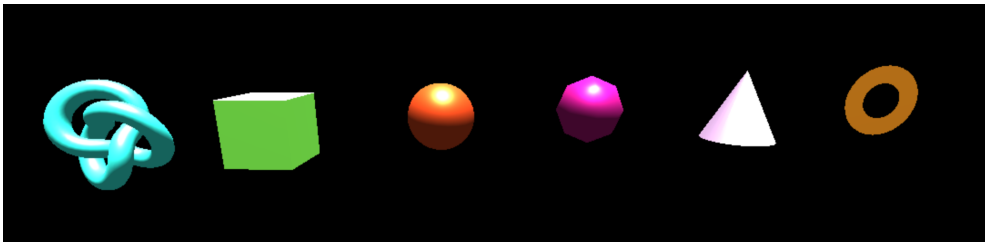
The goal of this assignment is to create multiple different geometries by clicking in the viewport. This means, rather than moving an existing mesh, we will create new ones in the `window.onclick` callback. On each click, our code will randomly choose a different geometry and a random color to place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



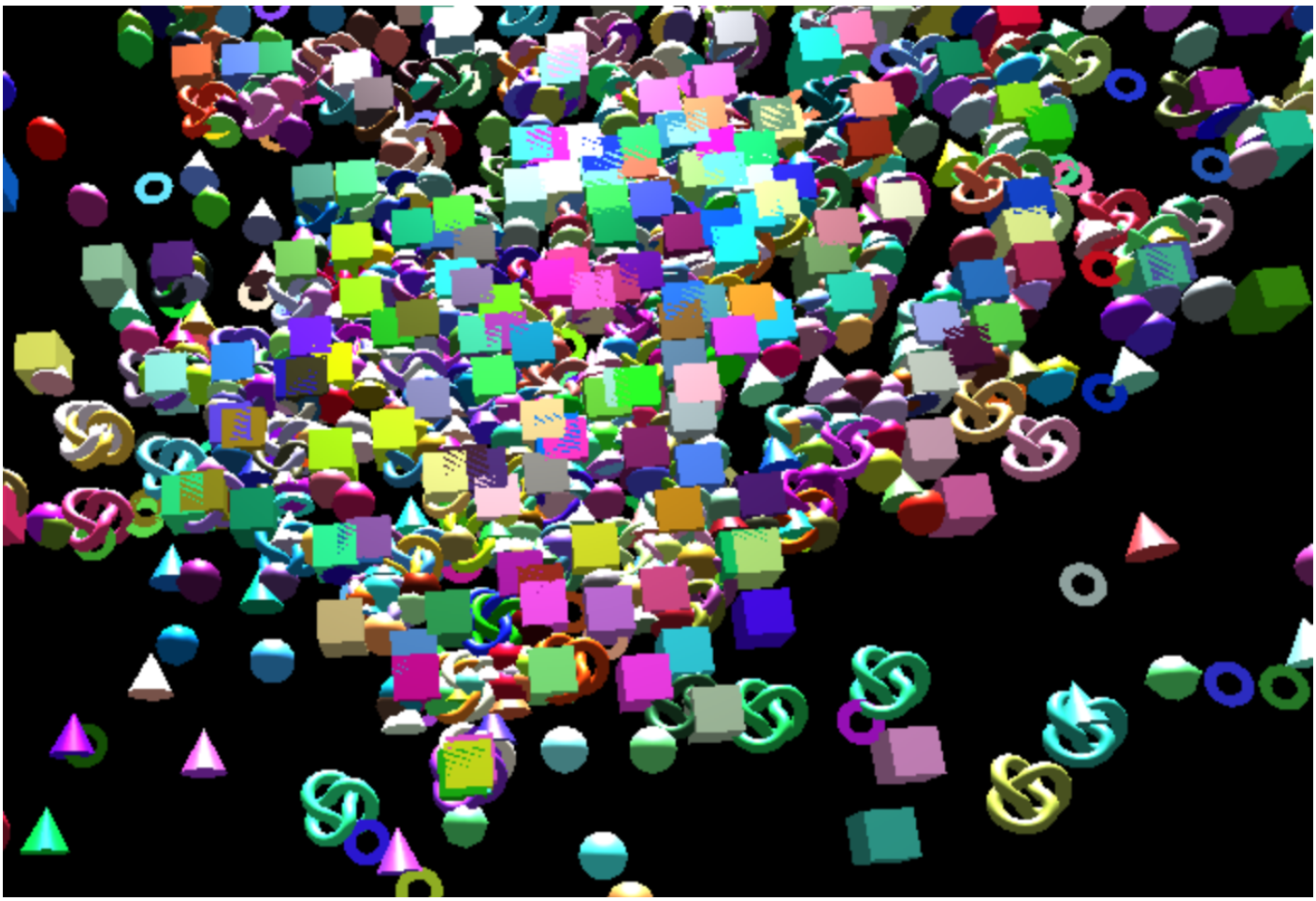
Please make sure that your code is accessible through Github Pages. Also, please commit this PDF and your final code to your Github fork, and submit a pull request.

Link to your assignment: `https://hltdev8642.github.io/cs460student/03/index.html`

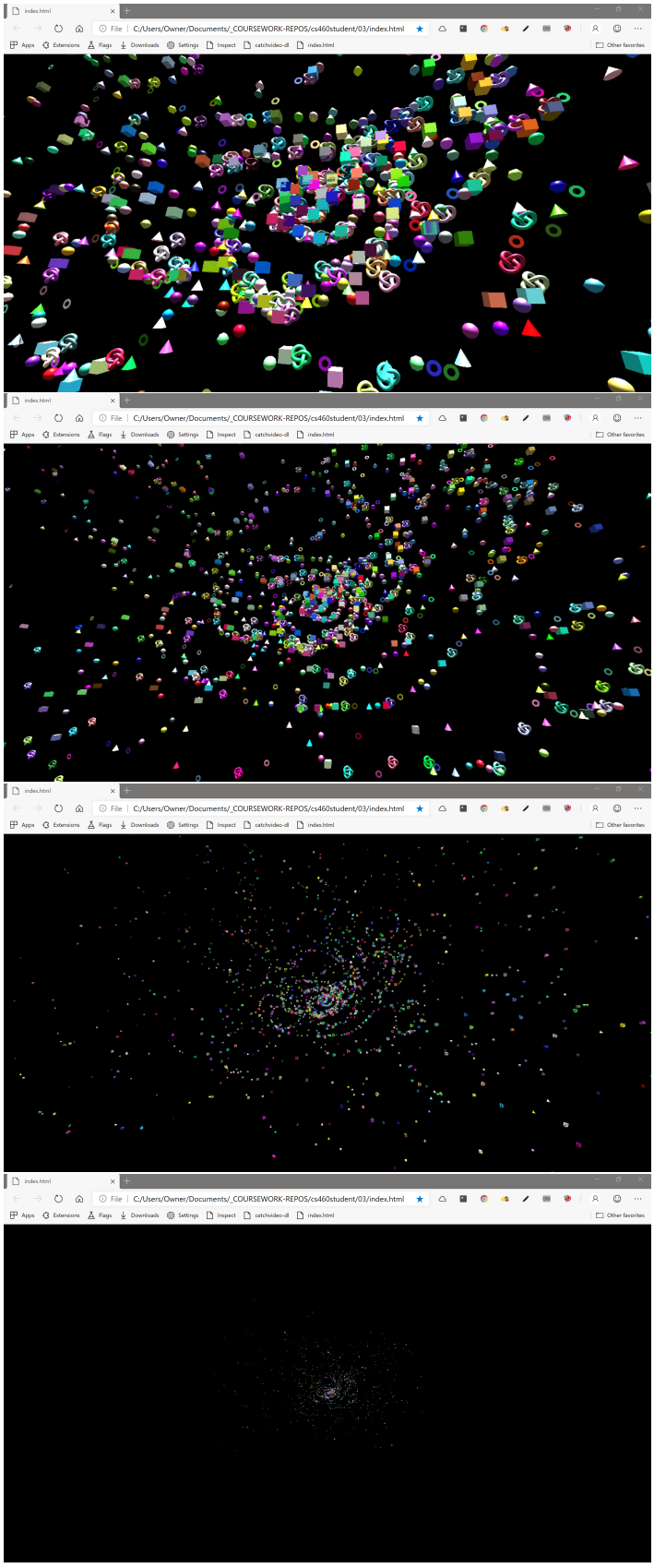
Bonus (33 points):

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

Yes. It is seen most at the points in the scene that have the highest density of geometries, or regions in which there are many overlapping objects/polygons.



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**.

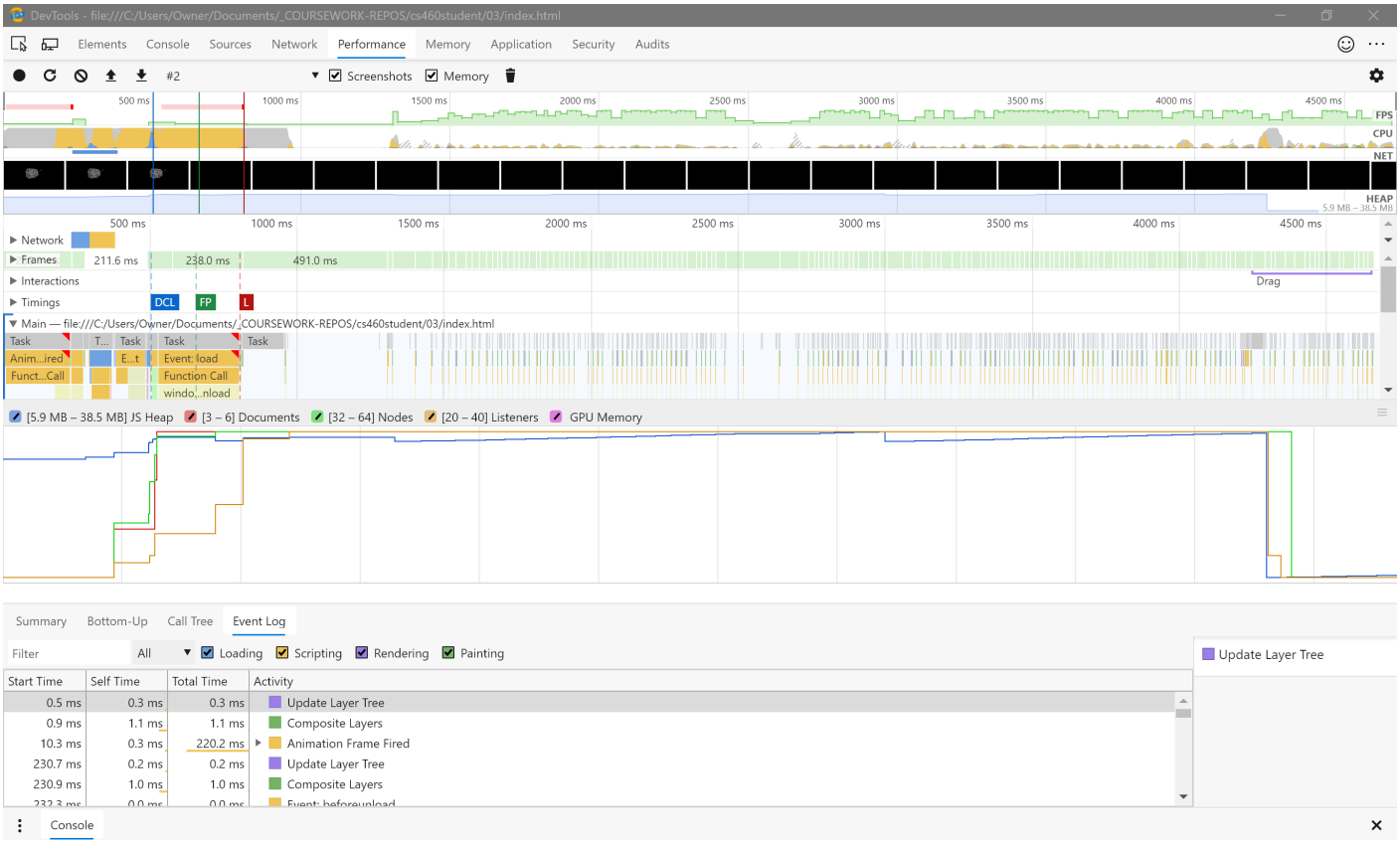


Part 3 (18 points): Please keep track of the number of placed objects and print the count in the JavaScript console. Now, with the change to `window.onmousemove`, after how many objects do you see a slower rendering performance?

I honestly couldn't reach a number that it started to decline. The count I was at however was at 3441 objects and there still was very little noticeable drop in performance.

However I realized through use of the Application/Memory inspector in chrome that my visual assessment of performance was probably not too accurate, as the trace revealed a very well-defined point in the runtime when the performance decreased.

(***See screenshot below***)



What happens if the console is not open during drawing?

The performance *may* have increased slightly, however it was nothing too significant if any. Also, it would be difficult to use the performance monitor (to check the performance), as this is ironically part of the console/inspector panel...

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

A lot....

Using the console command ”renderer.info.render.triangles”, I was able to determine that for one of each type of geometry the total number of triangles is **4164**.

If we assume the slowdown happened at 3241 objects (as mentioned prior), that would be: **4164 triangles per iteration X 3241 / 6 objects per iteration = (3241/6) * 4164**

= A GRAND TOTAL OF (APPROX): **1750140 triangles**