

CS460 Fall 2020

Github Username: ayahea

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
<code>THREE.BoxBufferGeometry</code>	( width, height, depth )
<code>THREE.TorusKnotBufferGeometry</code>	( radius, tube, tubularSegments, radialSegments )
<code>THREE.SphereBufferGeometry</code>	( radius, widthSegments, heightSegments )
<code>THREE.OctahedronBufferGeometry</code>	( radius )
<code>THREE.ConeBufferGeometry</code>	( radius, height )
<code>THREE.RingBufferGeometry</code>	( 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.**

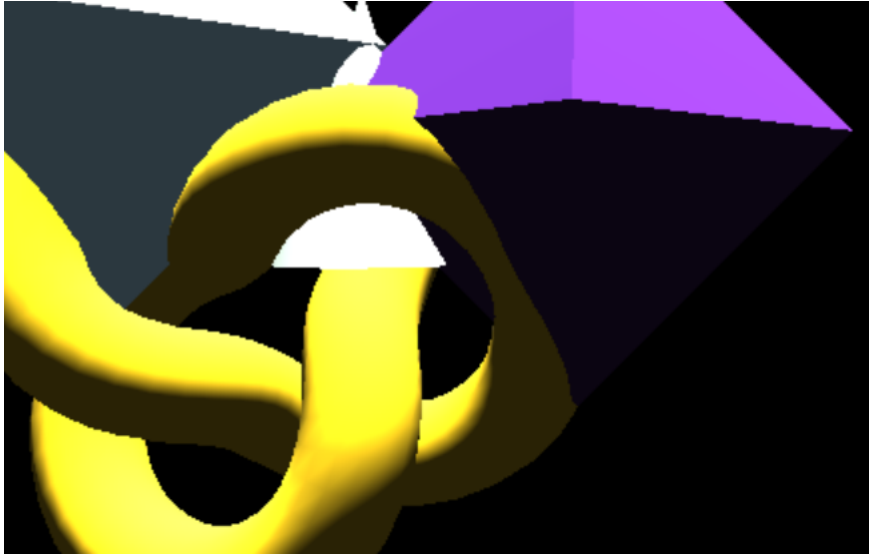
Link to your assignment: <https://ayahea.github.io/cs460student/03/>

**Bonus (33 points):**

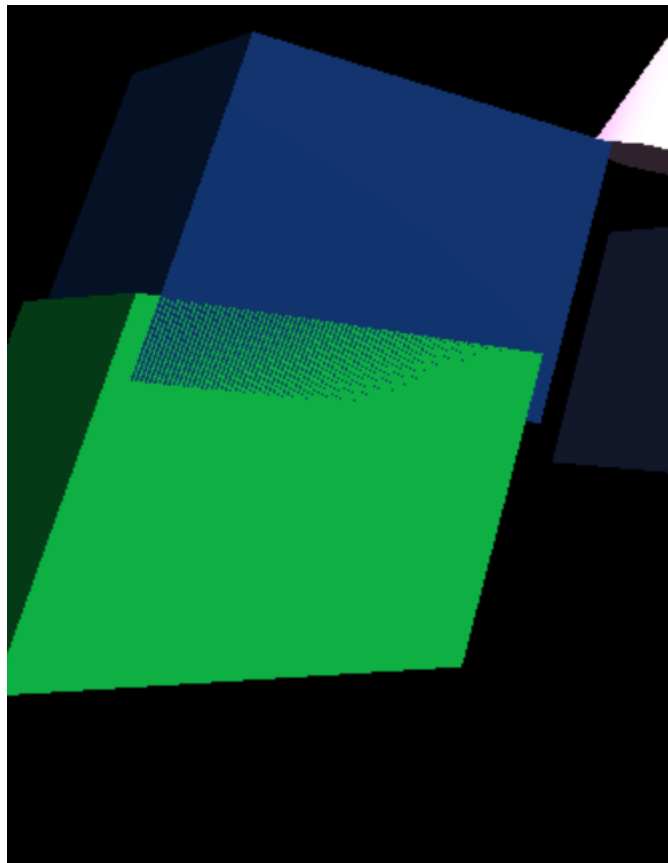
Part 1 (5 points): Do you observe Z-Fighting? If yes, when?

Many times, I did not observe Z-Fighting, but in some cases I did.

The first photo is an example of no Z-Fighting happening despite the overlap:



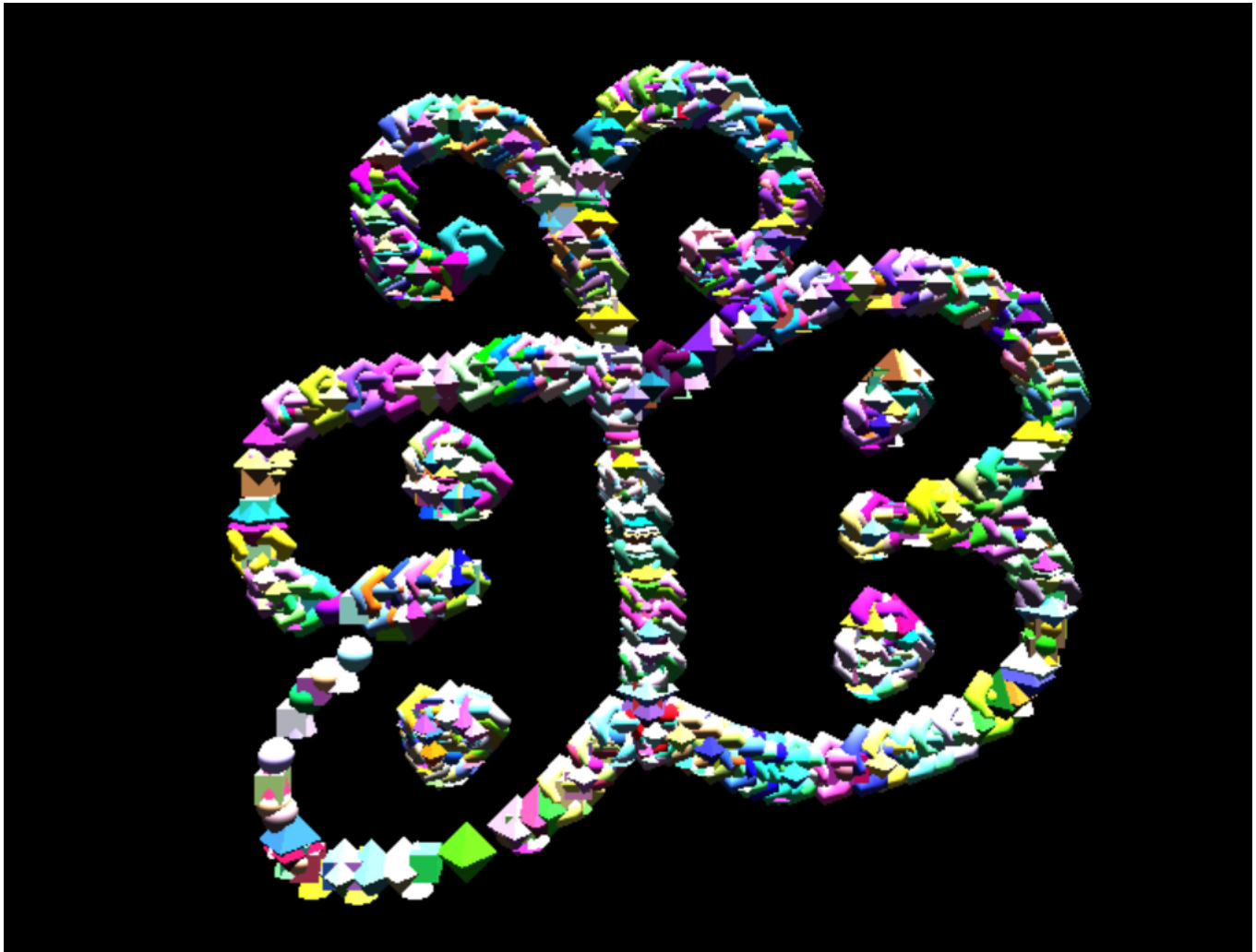
The second photo is an example of Z-Fighting:



I'm not sure why this is so, perhaps it is because of the colors or because in the latter photo, the two z-fighting shapes are both cubes.

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**.

Here is a butterfly:



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?

It might have been before this number but I didn't notice it right away, but I started to clearly notice that around 2000 shapes the performance decreased, and the shapes were not being drawn right where my mouse was clicking.

What happens if the console is not open during drawing?

It will still count the number of shapes, and once the console opens again it will report the number of shapes.

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

If by triangles you mean cones, I can estimate that there have been  $2000/6 = 333$  cones drawn because the shapes are selected at random from a set of 6 shapes. If, however, by triangles you mean the individual parts of a 3D shape, then certainly some multiple of 2000, possibly more than 10,000.