CS460 Fall 2019
Name: JACOB YOON ZENG YEW
Student ID: 01596523
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.

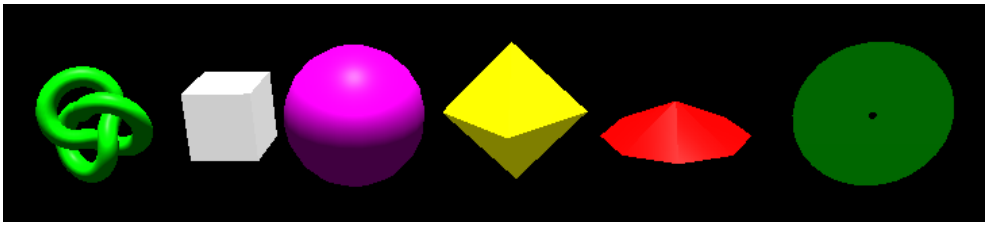
The goal of this assignment is to create multiple different geometries by clicking in the viewport. This means, rather than moving an existing mesh, we will create new ones in the `window.onclick` callback. On each click, our code will randomly choose a different geometry and a random color to place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

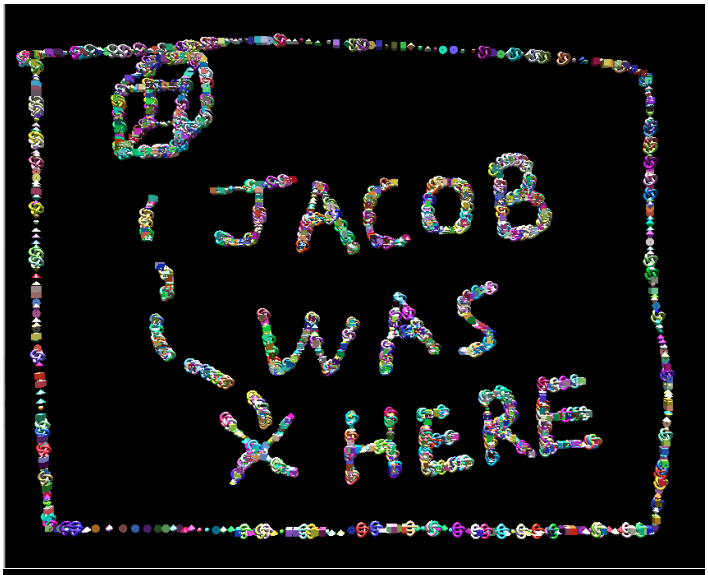
Hosted on GitHub Here!: `https://genlikan.github.io/cs460student/03/`

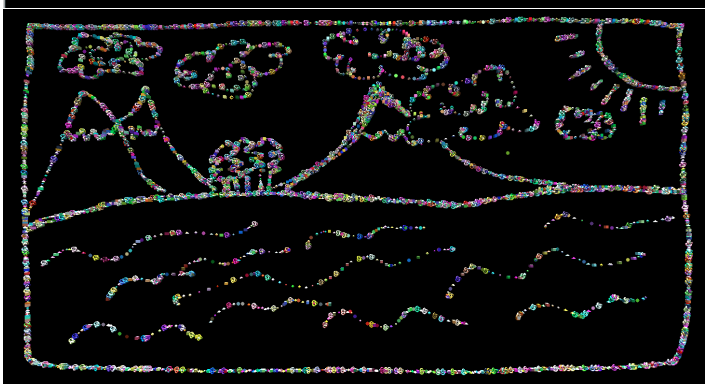
Bonus (33 points):

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

Yes, Z-Fighting occurs when the shapes intersect with one another, especially obvious when a cluster of them are made and when the camera is tilted; when two faces occupy the same space.

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?

With the Console open (with a laptop Nvidia GTX 1070), around 5500 to 6000 shapes, the slow down becomes more visibly obvious. With the Console closed 5900 to 6300, the slow down begins to seep in.

What happens if the console is not open during drawing?

With the console open, rendering performance seems to tank earlier; this is likely because the console takes extra resources to display logs/outputs. When the console is closed, the rendering performance is only slightly smoother for a some 300 shapes before the slow down becomes more obvious.

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

With the help of the lovely three.js documentation, we can calculate approximately the average amount of triangles in the scene based on the amount of segments for some shapes. Below are the calculated totals for each shape with the current values from the hosted assignment.

$$
\begin{aligned}
SimpleBoxes/Cubes &= 12 && Triangles \\
Complex/SmoothTorusKnot &= 1600 && Triangles \\
SmoothSphere &= 1024 && Triangles \\
SimpleOctahedron &= 8 && Triangles \\
SimpleCone &= 16 && Triangles \\
SmoothRing &= 1024 && Triangles
\end{aligned}
$$

Now, say the slow down becomes painfully obvious at 6000 shapes and that (by some RNG miracle) each shape was distributed evenly; that would mean 1000 per geometry type. This brings it to the grand total of 3,684,000 triangles before the slow-down occurs.