CS460 Fall 2020
Github Username: Glisync-00
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.

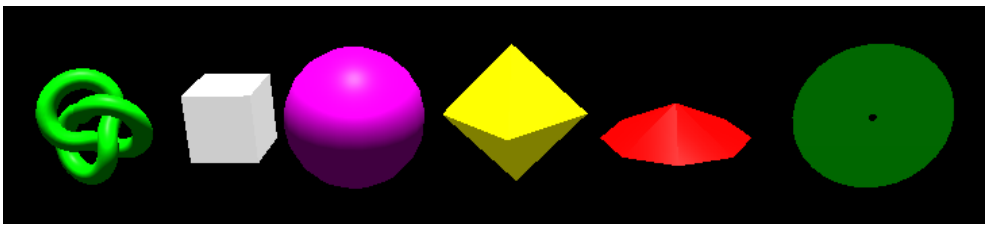
The goal of this assignment is to create multiple different geometries by clicking in the viewport. This means, rather than moving an existing mesh, we will create new ones in the `window.onclick` callback. On each click, our code will randomly choose a different geometry and a random color to place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



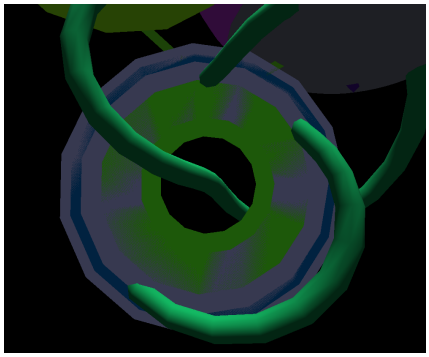
Please make sure that your code is accessible through Github Pages. Also, please commit this PDF and your final code to your Github fork, and submit a pull request.

Link to your assignment: `https://glisync-00.github.io/cs460student/`

Bonus (33 points):

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

I do see z-fighting and it occurs whenever two ring objects intersect one another. You can see this below



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 start to see slower render performance at around 5000 objects spawned (shown above)

What happens if the console is not open during drawing?

This answer depends. If I open the html file without the console, then if I open the console during the session the objects spawn away from my mouse cursor rather than at the mouse cursor. If I don't open the console the objects spawn normally at the mouse cursor.

The same could be said for the opposite case. If I open the html file with the console, everything will spawn at the mouse cursor, but if I closed the console during that session the shapes will spawn away from the mouse cursor instead.

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

For this it's really hard to say. I tried to calculate this anyways based on the triangles I saw for each shape in their documentation using different segment sizes. The Torus knot and sphere in particular are difficult to gauge the total triangles as I had a hard time seeing the relationship of the segments to what was drawn. Below is estimated formulas I found for each shape's triangle total.

Shape	Total Triangles
Box	12
Torus Knot	(2 * tubular segments * 2) + (radial segments * 3)
Sphere	(2 * width segments) + (height segments - 2) * (2 * width segments)
Octahedron	8
Cone	2 * radial segments
Ring	theta segments * (2 * pi segments)

I kept the total triangles in a variable and printed to console. Since the total segments are static in my code, I calculated the total triangles for each shape myself rather than having the computer constantly recalculate the values.

At around 5273 objects (when my computer slows down), there are about 2203628 triangles in the scene! (console shown above)