CS460 Fall 2020
Github Username: nandiniiys
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.

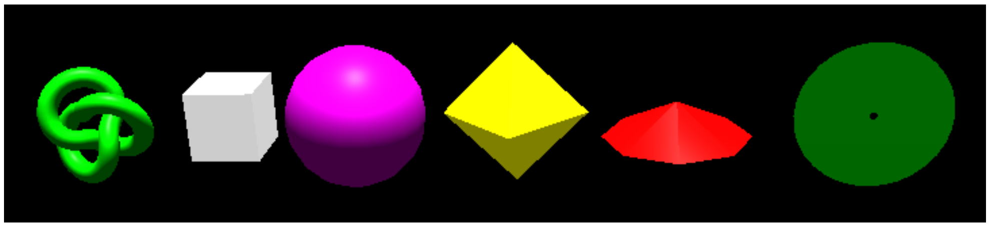
The goal of this assignment is to create multiple different geometries by clicking in the viewport. This means, rather than moving an existing mesh, we will create new ones in the `window.onclick` callback. On each click, our code will randomly choose a different geometry and a random color to place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

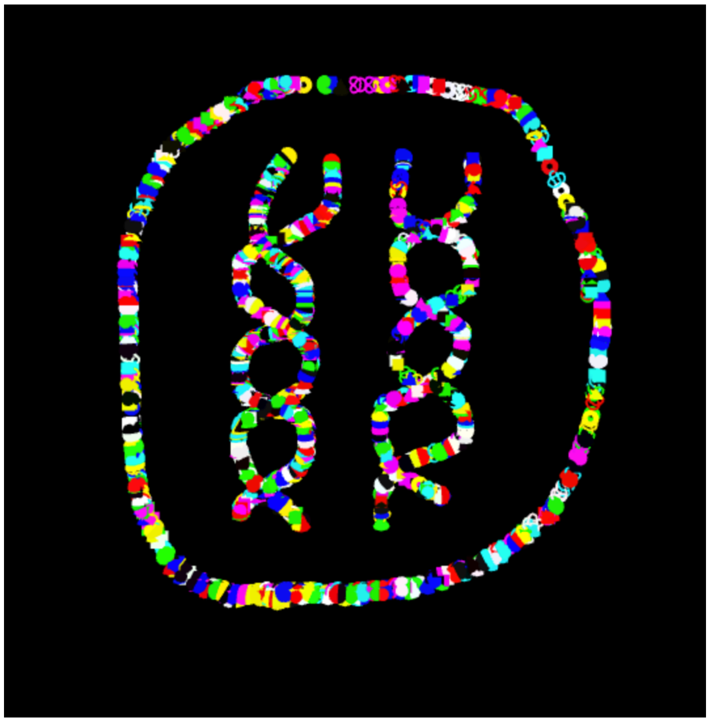
Link to your assignment: `https://nandiniiys.github.io/cs460student/03/index.html`

Bonus (33 points):

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

I see Z fighting when I click in the same position multiple times. This is because by clicking in the same place multiple times places multiple shapes in the same x - y position and the renderer gets confused as to which one to place on top.

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?

The rendering performance seems to start slowing down after placing about 350 objects but the difference isn't very significant. We start seeing a significant decline in performance after placing about 500 objects.

What happens if the console is not open during drawing?

I explored three scenarios:
1) Loading the page with the console already closed and keeping it closed while drawing
- In this case you can draw all over the screen from corner to corner.
2) Loading the page with with the console already closed and then opening it before starting to draw
- In this case, as before, you can draw all over the screen, from corner to corner. If you draw over the console, after you close the console you can still see the drawings at the back.

3) Loading the page while the console is open, then closing it before starting to draw
- In this case, you can draw in any area of the screen apart from the area the console occupies when its open.

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

BoxBufferGeometry | 2 triangles per side | 8 sides | 2 * 8 = 16 triangles
ConeBufferGeometry | 1 triangle per radial segment | default of 8 used | = 8 triangles
OctahedronBufferGeometry | each side = triangle | 8 sides | = 8 triangles
PlaneBufferGeometry | 2 triangles | = 2 triangles
RingBufferGeometry | 30 sides | 8 squares per side | 2 triangles per square | 30 * 8 * 2 = 480 triangles
SphereBufferGeometry | 32 * 32 squares | 2 triangles per square | 32 * 32 * 2 = 2048 triangles
TorusKnotGeometry | 60 * 20 squares | 2 triangles per square | 60 * 20 * 2 = 2400 triangles
Total = 4962 triangles
Slow down began at 350 objects. Assuming the shapes were evenly placed such that each shape appeared 350/6   58 times: = 4962 * (58) = 287, 796 triangles.

–> Approximately 287, 796 triangles are drawn as soon as slow-down occurs.