

## Project 4: Ferris Wheel

The code I wrote is fueled by my hate for WebGL. I'm very much a declarative over imperative solution type of person, and WebGL feels very imperative. To fix this, I decided to build a bunch of abstractions on top of WebGL. These abstractions are all heavily inspired by Three.js, but it's all WebGL code I wrote myself. In other words, here is my crappy remake of Three.js.

The main components of my code:

- Vector3 class
  - This class handles vectors with x, y, and z coordinates
  - It also contains a bunch of functions you can use to manipulate vectors (add, multiply, divide, length squared, length, normalize, etc.)
  - Some of these functions I didn't wind up using (e.g. normalize) but I'm leaving them in because I plan on using this as the foundation for the next project in this class
- Clock class
  - Super small class that uses `window.performance.now()` to get super accurate clock times. I'm using this to calculate the time between each `window.requestAnimationFrame` tick.
- Scene class
  - Inspired by Three.js' scene and other components, this class handles most of the WebGL configuration (loading shaders, calculating viewport size, etc.). Some of this is functionality that lives outside of Three.js' scene, so in the next project I might break this up into multiple classes.
  - Like Three.js, my world class has a `world.add` method that allows you to add objects to the world
- Polygone class
  - Heavily inspired by Three.js' geometry classes except my version combines the geometry and the mesh into one class. Eventually, I might split this out into separate Geometry, Material, and Mesh classes, but for now, this simplified sys



## Challenges

No challenges I wasn't able to overcome. Could probably add some optimizations and I'm sure I'm doing some of the WebGL stuff slightly incorrectly, but overall I'm pretty happy with the results. The hardest part was getting my Vector3.rotate class to handle x, y, z rotations. The rotation function itself isn't too complicated, but applying that to the Polygone class took me a while – I realized I was accidentally applying the z rotation twice so most of the time I spent on this was due to me overlooking this.

## Next steps

If you can't tell, I'm more comfortable working with lines instead of solid objects, but I would like to rewrite my abstractions so I can specify at a high level whether an object's geometry/material should be lines or solid faces.

## Ferris Wheel

- Number of seats = , Seat index to rotate = 
- If you click (focus) the seat index slider, you can use the left/right arrow keys to move it by one seat.
- *Note: the seat index rotation slider is clamped from [0-numSeats) to prevent out of index rotation.*

