Introduction

What is `three.js` and why do we need it?

- A 3D JavaScript library that enables developers to create 3D experiences for the web.
- `three.js` works with WebGL

What is WebGL

- It's a JavaScript API
- Renders at a remarkable speed
- Result can be drawn in a `<canvas>`
- Compatible with most modern browsers
- Uses GPU

Basic Scene

1. Creat a simple `index.html`
2. Link `index.html` to `script.js`
3. Load `three.js`
    - add `three.min.js` to the folder
    - link to the file in HTML **before** `script.js` tag
4. Use `three.js`
    - we now have access to the `THREE` object

- We need **four elements to construct a scene**:
    - a scene that will contain objects

```
const scene = new THREE.Scene();
```

    - objects
    - camera
    - renderer

Scene

- consider it to be a container
- we can put objects, models, lights in it
- we need to ask `three.js` to render the scene

```
const scene = new THREE.Scene();
```

Objects

- different types of objects:
    - primitive geometries
    - imported models
    - particles
    - lights and more
- We need to create a `Mesh` (the object) which is a combination of `geometry` (shape) and `material` (how the object will look)

```
const geometry = new THREE.BoxGeometry(1, 1, 1); // size
const material = new THREE.MeshBasicMaterial({color:"orange"});
const mesh = new THREE.Mesh(geometry, material);
```

- add to scene: `scene.add(mesh);`

Camera

- not visible
- **serves as point of view during rendering**
- many different types

```
const camera = new THREE.PerspectiveCamera()
scene.add(camera);
```

- Parameters:
  1. **field of view**: verticle vision angle, in degrees, also called **fov**
  2. **the aspect ratio**: width of the renderer divided by the height of the renderer

Renderer

- render the scene from the camera's pov
- result drawn into a `<canvas>`
  - which is a HTML element in which stuff can be drawn
- `three.js` will use WebGL to draw the render inside `canvas`

```
<body>
    <canvas class="webgl"></canvas>
</body>
```

```
const canvas = document.querySelector("canvas.webgl");
const renderer = new THREE.WebGLRenderer(
{
    canvas: canvas
});
renderer.setSize(size.width, size.height);
```

- **render** `renderer.render(scene, camera);`
- Move camera because by default, camera is inside of the object (aka cube) we created

Transform an object

- `position` : an object with `x, y, and z` properties
  - left/right: `x`
  - Up/down: `y`
  - Forward/backward: axis `z`
- `rotation`
- `scale`

First Cube

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Basic Scene in three.js</title>
</head>
<body>
    <canvas class="webgl"></canvas>
    <script src="three.min.js"> </script>
    <script src="script.js"></script>
</body>
</html>
```

```js
// create a scene
const scene = new THREE.Scene();
// console.log(scene);

// create an object: orange cube
const geometry = new THREE.BoxGeometry(1, 1, 1); // size
const material = new THREE.MeshBasicMaterial({color:"orange"});
const mesh = new THREE.Mesh(geometry, material);

// add to scene
scene.add(mesh);

// size
const size = {
    width: 800,
    height:600
};

// add a camera
const camera = new THREE.PerspectiveCamera(75, size.width/size.height);
// move the position of the camera
camera.position.z = 2;
scene.add(camera);

const canvas = document.querySelector('canvas.webgl');
```

```
console.log(canvas);

// create a renderer
const renderer = new THREE.WebGLRenderer({
    canvas: canvas,
});

// set renderer size
renderer.setSize(size.width, size.height);

// render
renderer.render(scene, camera);
```

Webpack

Limitations of loading `three.js` with `<script>`

- does not include some of the classes
- we need to run a server to emulate a website for security reasons
- Instead we will **use a bundler**

What is a bundler

- a tool in which you need js, html, css, images, ts, stylus, sass, etc
- the bundler apply potential modifications and output a web-friendsly *bundle*
- can do more like local server, manage dependencies, improve campatibility, add modules support, optimize files, deploy, etc
- we are using **webpack**

How to use the template

```
npm install
npm run dev
build: npm run build
```

- `npm` is a package manager that's been installed with `node.js` which will download what needed to be downloaded for your project
- `npm run build` will create a `dist` directory, and your website is ready to publish online

Structure

- working files are located in `/src`
- `script.js` is the **root file**
- `styles.css` is loaded from `script.js` with `import`
- the page automatically reloads as you save
- some mistakes might break teh auto-reloading and you have to rerun mannually
- You can put *static files* in the `/static` folder: img, xml files, etc
- You can access this local project from any other device on the same network by entering the same url

Add a `<canvas>`

```
<canvas class="webgl"></canvas>
```

Access `THREE` variable

```
import * as THREE from 'three';
```

Transform Objects

- There are 4 properties to transform objects
  - `position`
  - `scale`
  - `rotation`
  - `quarternion`
- All classes that inherit from the `Object3D` possess those properties like `PerspectiveCamera` or `Mesh`
  - these properties will be compiled in matrices

Move Objects

- `position` has 3 properties
  - `x` : left and right
  - `y` : up and down
  - `z` : closer and farther
- `position` inherit from **Vector3** which has many useful methods
- some methods commonly used
  - `position.distanceTo(vector3Object)`
  - `position.normalize()`
  - `position.set(x, y, z)`

Axes Helper

```
const axesHelper = new THREE.AxesHelper(num);
scene.add(axesHelper);
```

Scale Objects

- `scale` has 3 properties:
- default value of each axis is `1`
  - `x`
  - `y`
  - `z`

```
mesh.scale.set(x, y, z);
mesh.scale.x = num;
mesh.scale.y = num;
mesh.scale.z = num;
```

Rotation

- Two ways to rotate objects
  - `rotation` :
    - `x, y, z` properties, it's a **Euler** Object
    - Imagine putting a stick through the object's center in the axis direction and then rotating that object on that stick
    - The rotation goes by default in `x, y, z` order but this may result in a bad result when an axis fails to work
      - this is called a **gimbal lock**
    - To solve this, you can change the order using `reorder(...)`

```
object.rotation.reorder('yxz');
```

  - `quarternion` :
    - also expresses a rotation but in a more mathematical way

`lookAt()`

- this method rotates the object so that it `-z` **faces the target you provided**, target **must be** a `Vector3`

```
camera.lookAt(mesh.position);
```

Scene Group

- You can put objects inside groups and use `position` , `rotation` , `quaternion` , `scale` on the group in its entirety → use the `Group` class

```
const group = new THREE.Group();
scene.add(group);

const cube1 = new THREE.Mesh(
    new THREE.BoxGeometry(1, 1, 1),
    new THREE.MeshBasicMaterial({color:'blue'})
);

group.add(cube1);
```

Animation

- Animating is like doing stop motion
  - move the object
  - take a picture
  - move the object a bit more
  - take a picture
- Most screens run at 60 **frames per second** (FPS) but not always. Animation must look the same regardless of the frame rate.

Request Animation Frame

- The purpose of `requestAnimationFrame` is to call the function provided on the next frame.

```
const tick = () => {
//    console.log('tick');
    //update objects
    mesh.position.x += 0.1;

    // render
    renderer.render(scene, camera);

    window.requestAnimaionFrame(tick);
};
tick();
```

Adapt to the different framerate

Solution 1: `deltaTime`

- Different machine with different frame rate will preceive the animation differently
- Solution:
  - find out how much time it's been since the last frame refresh: get the current **timestamp** with `Date.now()`
  - subtract the previous time to get `deltaTime`

```
let time = Date.now();

const tick = () => {
     const currentTime = Date.now();
     const deltaTime = currentTime - time;
     time = currentTime;

     mesh.rotation.x += 0.001 * deltaTime;
 };
```

Solution 2: `Clock`

```
const clock = new THREE.Clock();

const tick = () => {
    const elapsedTime = clock.getElapsedTime();

    // one full circle per second
    mesh.rotation.x = elapsedTime * Math.PI * 2;

    //makes object go up and down
    mesh.position.y = Math.sin(elapsedTime * 4) * 4;
};
```

- **do not use** `getDelta()`

Using a library

- If you want more control over animation, create tweens,timelines, etc
  - use a library like `GSAP`
- Add `GSAP` to the dependencies with `npm install --save gsap@3.5.1`
- import the library `import gsap from 'gsap';`

`GSAP`

- Make a tween with `gsap.to()`
  - **tween**: nbetweening, also commonly known as tweening, is a process in animation that involves generating intermediate frames, called inbetweens, between two keyframes. The intended result is to create the illusion of movement by smoothly transitioning one image into another.

```
gsap.to(mesh.position, {duration: 1, delay: 1, x: 2});
```

Camera

- Camera is **an abstract class** → you should not use it directly
- **Types** of cameras
  - **ArrayCamera** renders the scene **from multiple cameras on specific areas of the render**
  - **StereoCamera** render the scene through two camera that **mimic the eyes to create a parallax effect**
    - use with VR headset, red and blue glasses or cardboard
  - **CubeCamera** does 6 renders, each of them facing a different direction
    - can render the surrounding for things like environment map, reflection or shadow map
  - **OrthographicCamera** renders the scene without perspectiver
  - **PerspectiveCamera**

Perspective Camera

- Parameters:
  - **field of view**: in **degrees**, vertical visual angle
    - When you have a large field of view, the shape near the edge of the camera's fov will be distorted
  - **aspect ratio**: the width of the render divided by the height of the render
  - **near and far**: the last two parameters correspond to how close and far the camera can see
    - any object that is outside of this scope will not show up
    - do not use extreme values to prevent **z-fighting**

Orthographic Camera

- Compared to perspective camera, orthographic camera **lacks perspective**
  - objects are the same size regardless of their distance to the camera
- Parameters:
  - how far the camera can see in each direction
    - `left` : `num * aspectRatio`
    - `right` : `num * aspectRatio`
    - `top`
    - `bottom`
    - `near`
    - `far`

Custom Controls

- control the camera position with the mouse
  - first we **need the coordinates of the mouse**

```
window.addEventListener('mousemove', (event) => {
   console.log(event.clientX, event.clientY);
});

//output:
// x coordinate of cursor: 119, y coordinate of cursor: 373
```

  - then create a **cursor object** and store these coordinates times `sizes.width` and `sizes.height` respectively

```
// cursor
const cursor = {
  x: 0,
  y: 0,
};

window.addEventListener("mousemove", (event) => {
  cursor.x = event.clientX / sizes.width - 0.5;
  cursor.y = - (event.clientY / sizes.height - 0.5);
  console.log(
    "x coordinate of cursor: " +
      cursor.x +
      ", y coordinate of cursor: " +
      cursor.y
  );
});
```

  - update camera `position`

```
const tick = () => {
    camera.position.x = cursor.x;
    camera.position.y = cursor.y;
    camera.lookAt(new THREE.Vector3());
    //make camera look at the center by default
}
```

  - spin horizontally by `Math.sin(...)`, `Math.cos()` and `Math.PI`

```
const tick = () => {
    camera.position.x = Math.sin(cursor.x * Math.PI * 2) * 10;
    camera.position.z = Math.cos(cursor.x * Math.PI * 2) * 10;
    camera.position.y = cursor.y * 8;
    camera.lookAt(mesh.position);
}
```

- `three.js` has built controls to make this easier

Built-in Controls

- search `Controls` in three.js documentation

Device Orientation Controls

- automatically retrieve the device orientation if your device, OS, and browser allow it and rotate the camera accordingly

Fly Controls

- enable moving the camera like if you were on a spaceship, you can rotate on all three axes

First Person Controls

- similar to fly controls, but cannot change the y-axis
- is not FPS game

Pointer Lock Controls

- go for first person 3D games

Orbit Controls

- similar to the controls we've made but with more features

Trackball Control

- similar to ObitCOntrols without the vertical angle limit

Transformation COntrols

- nothing to do with camera
- allows you to move the object

Drag Controls

- Nothing to do with camera
- drag objects around

`OrbitControls`

- **Instanciate**
  - it **cannot** be access with `THREE.orbitControls`
  - we need to **import** it

```
import { OrbitControls } from 'three/examples/jsm/controls/OrbitControls.js';
```

  - add `controls` after `camera`
  - The class needs the **camera and a DOM element for mouse events**

```
const canvas = document.querySelector('.webgl');
...

// create camera
...

// create controls
const controls = new OrbitControls(camera, canvas);
```

  - Now you have full control of the canvas, **drag and drop to experiment**

Target

- `target` property is a **Vector3**
- change default target by

```
const canvas = document.querySelector('.webgl');
...

// create camera
...

// create controls
const controls = new OrbitControls(camera, canvas);
controls.target.y = 2;
controls.update();
```

- update the change with

```
controls.update();
```

Damping

- the movement so far has been harsh.
  - to fix this, we can add damping
- **Damping** will **smooth the animation** by adding acceleration and friction
  - controls **need to be updated on each frame** for it to work

```
const controls = new OrbitControls(camera, canvas);
controls.enableDamping = true;

...

const tick = () => {
    controls.update();
}
```

---

fullscreen and resizing

Fit in the viewport

- get the viewpoint's width and height

```
const sizes = {
    width: window.innerWidth,
    height: window.innerHeight,
};
```

- get rid of the default margin

```
body {
    margin: 0px;
    padding: 0px;
}
```

- move `<canvas>` to the top left corner

```
canvas.webgl {
    position: fixed;
    left: 0;
    top: 0;
}
```

- some browsers have a blue outline, get rid of it

```
canvas.webgl {
    position: fixed;
    left: 0;
    top: 0;
    outline: none;
}
```

- Finally, make sure the page is impossible to scroll

```
html,body {
    overflow: hidden;
}
```

Handling resizing

- listen to the `resize` event

```
window.addEventListen('resize', () => {
   console.log("resize test") ;
});
```

- update the `sizes` variable

```
window.addEventlisten('resize', ()=> {
    sizes.width = window.innerWidth;
    sizes.height = winddow.innerHeight;
});
```

- update `camera` aspect ratio

```
window.addEventlisten('resize', ()=> {
    sizes.width = window.innerWidth;
    sizes.height = winddow.innerHeight;

    camera.aspect = sizes.width / sizes.height;
    camera.updateProjectionMatrix();
});
```

- Update `renderer`

```
window.addEventlisten('resize', ()=> {
    sizes.width = window.innerWidth;
    sizes.height = winddow.innerHeight;

    camera.aspect = sizes.width / sizes.height;
    camera.updateProjectionMatrix();

    renderer.setSize(sizes.width, sizes.height);
});
```

Handling pixel ratio

- Some browsers may see a blurry render and stairs effect on the edges
    - this is because some machine has a screen with a pixel ratio greater than 1
- Get current pixel ratio

```
window.devicePixelRatio
```

- Update renderer with pixel ratio
    - limit the pixel ratio to avoid over rendering

```
window.addEventlisten('resize', ()⇒ {
    sizes.width = window.innerWidth;
    sizes.height = winddow.innerHeight;

    camera.aspect = sizes.width / sizes.height;
    camera.updateProjectionMatrix();

    renderer.setSize(sizes.width, sizes.height);
    renderer.setPixelRatio(Math.min(window.devicePixelRatio, 2));
});
```

Handling fullscreen

- add support to a full screen mode, listen to double click event

```
window.addEventListener('dblclick', () ⇒ {
    console.log("dblclick test");
});
```

- go to fullscreen mode
    - this does not work with safari (bc safari sucks)

```
window.addEventListener('dblclick', () ⇒ {
    if (!document.fullscreenElement) {
        canvas.requestFullscreen();
    } else {
        document.exitFullscreen();
    }
});
```

- Accommodation for Safari

```
// full screen
window.addEventListener("dblclick", () ⇒ {
  console.log('dblclick test');
  const fullscreenElement =
    document.fullscreenElement || document.webkitFullscreenElement;

  if (!fullscreenElement) {
```

```
    if (canvas.requestFullscreen) {
      canvas.requestFullscreen();
    } else if (canvas.webkitRequestFullscreen) {
      canvas.webkitRequestFullscreen();
    }
  } else {
    if (document.exitFullscreen) {
      document.exitFullscreen();
    } else if (document.webkitExitFullscreen) {
      document.webkitExitFullscreen();
    }
  }
});
```

Updating... (Latest Updated Date: 05/11/2022)