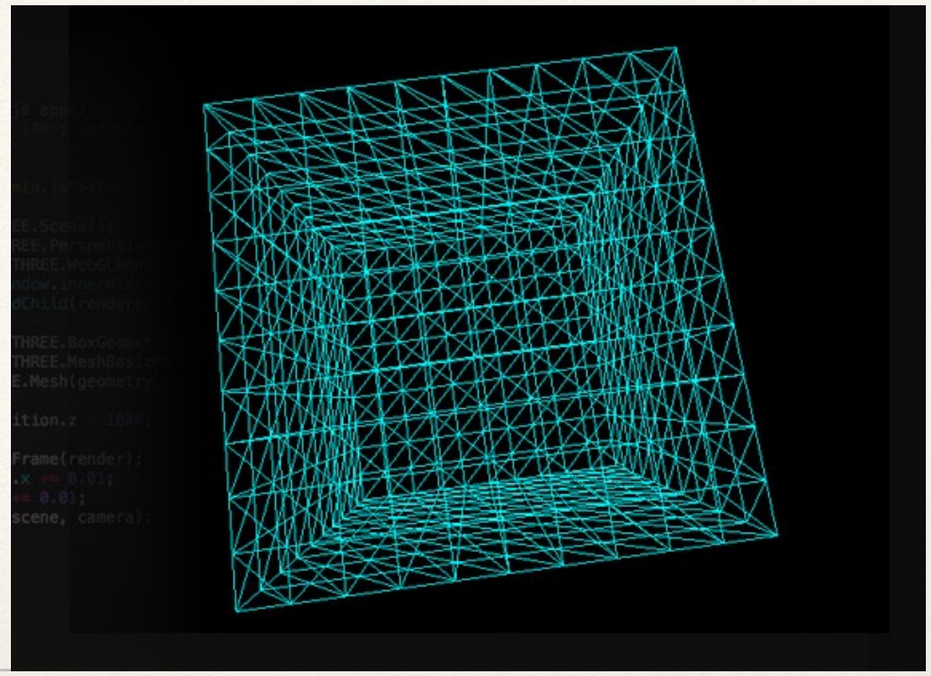
Dr. Sarah Abraham
University of Texas at Austin
Computer Science Department



Three.js

Elements of Graphics
CS324e

What is Three.js?

- A JavaScript library and interface for creating 3D scenes and animations
- Will run in any browser that supports JavaScript/ WebGL
- Built on top of WebGL (Web Graphics Library)
  - We will discuss WebGL in greater detail next time!

JavaScript vs Java

- No relation between the two languages
  - JavaScript named after Java for marketing reasons
- JavaScript:
  - Scripting language
  - Is not strongly typed (note: must declare variables with `var`, `let`, or `const`)
  - Does not support classes (note: does support OOP principles with prototypes)

Integrating Three.js

- Framework available from Three.js website: https://threejs.org/
  - Download entire framework here: https://github.com/mrdoob/three.js/archive/master.zip
- Must include three.js in any project directory
  - Script in “build” folder within master framework
  - Keep master framework clean then copy the script to specific project directory
- Must create a .html file that will run Three.js

Creating the HTML Scaffolding

- Create initial html, head, and body tags to set up webpage
  - Display the local (not deployed) .html file within a web browser to view / debug the results
- WebGL will run inside a Canvas element
  - Three.js will mostly hide working with the Canvas

HTML Example

```
<html>
  <head>
    <title>Hello World in Three.js</title>
  </head>
  <body>
  </body>
</html>
```

Adding Three.js to the HTML

- Connect the Three.js script to the html using the `script` tag within the html's body
- Create a new `script` tag that will contain code for displaying the custom scene

```
<script src="js/three.js"></script>
<script>
  //Draw scene here
</script>
```

Working with Three.js

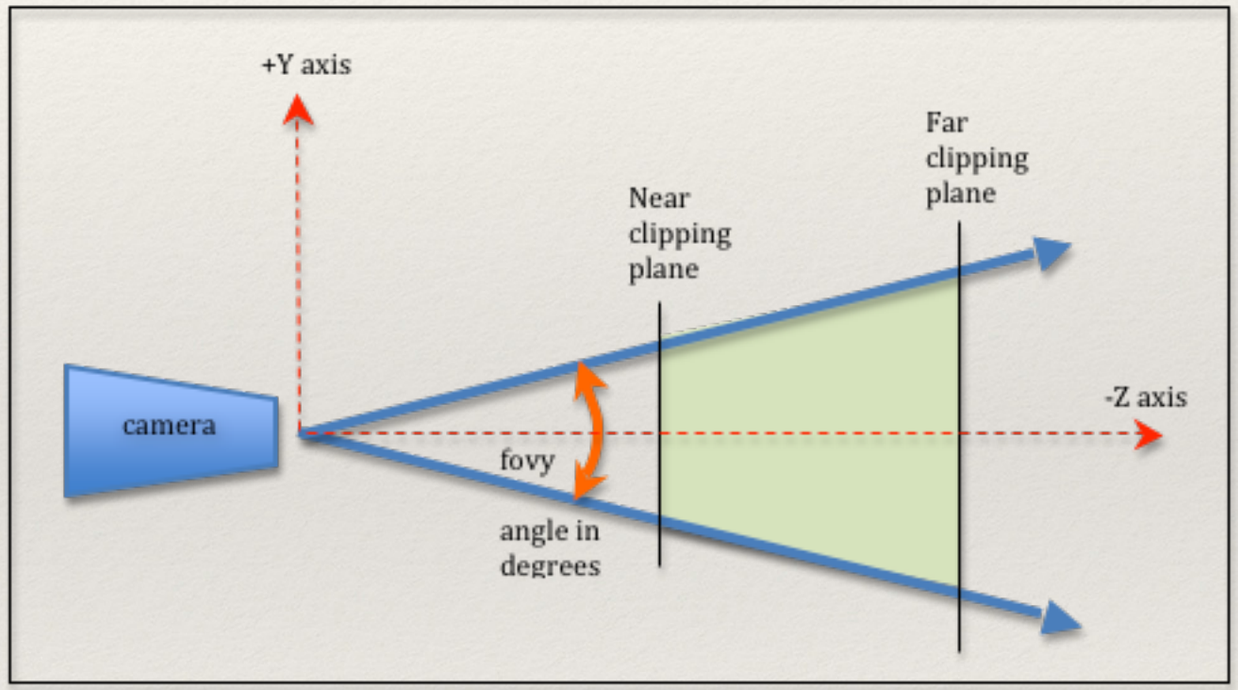
- Three.js requires a scene, a camera, and a renderer to draw things to the canvas
  - Will not display until all these things are in place

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

let camera = new THREE.PerspectiveCamera( 75,
window.innerWidth / window.innerHeight, 0.1,
1000 );

camera.position.z = 2;

let renderer = new THREE.WebGLRenderer();
```

Perspective Camera

- Three.js defines a perspective camera using:

1. field of view (degrees visible)
2. aspect ratio (width/ height of canvas)
3. near clipping plane
4. far clipping plane



(http://learnwebgl.brown37.net)

Other Cameras in Three.js

- `OrthographicCamera`: an orthographic camera for rendering 2D scenes
  - Defined by left, right, top, bottom, near, and far planes
- `StereoCamera`: A stereoscopic camera that renders from two perspective cameras for 3D displays
  - Defined by two perspective cameras

Associating the Renderer to the Canvas

- Set the size of the renderer
  - Should match camera's aspect ratio to prevent render stretching/squashing
- Add the renderer to the html document to display the renderer's canvas element

```
renderer.setSize( window.innerWidth,
window.innerHeight );

document.body.appendChild( renderer.domElement );
```

Adding Geometry to the Scene

- Create new geometry and material using the `Geometry` and `Material` objects
- Create a new mesh using the geometry and material
- Add the mesh to the scene

```
let geometry = new THREE.BoxGeometry();

let material = new
THREE.MeshBasicMaterial( {color: 0xff0000} );

let cube = new THREE.Mesh( geometry, material );

scene.add( cube );
```

Geometry Objects

- `Geometry` is an abstract class
- Many built in `Geometry` classes:
  - `BoxGeometry`
  - `CircleGeometry`
  - `ConeGeometry`
  - `PlaneGeometry`
  - `ShapeGeometry`
  - etc
- Documentation here: https://threejs.org/docs/#api/en/core/Geometry

Material Objects

- `Material` is an abstract class
- Many built-in material classes:
  - `MeshBasicMaterial`
  - `MeshPhongMaterial`
  - `MeshStandardMaterial`
  - `ShaderMaterial`
- Documentation here: https://threejs.org/docs/#api/en/materials/Material

Starting the Rendering Loop

- Must explicitly start calls to render to canvas
- Create a function that requests the web browser to redraw the screen then updates it from the renderer
  - Standard is 60 frames per second

```
draw = function () {

requestAnimationFrame( draw );

renderer.render( scene, camera );

};

draw();
```

Adding Animations

- Can apply transformations within the draw loop to add animations
- Transformations can be applied directly with a matrix transform (e.g. `Matrix4().setTranslation`)
- Transformations can be applied directly to `Object3D` objects (base class of `Mesh`) using `position`, `rotation`, `scale`

```
cube.rotation.x += 0.01;
```

Orbit Rotation

- Can apply orbital rotations by creating a pivot object that mesh object orbits

```
let pivot = new THREE.Object3D();

cube.position.x = 1;

pivot.add( cube );

pivot.rotation.x += 0.01;
```

References

- <https://threejs.org/docs/#manual/en/introduction/Creating-a-scene>