WEBGL WITH THREE JS

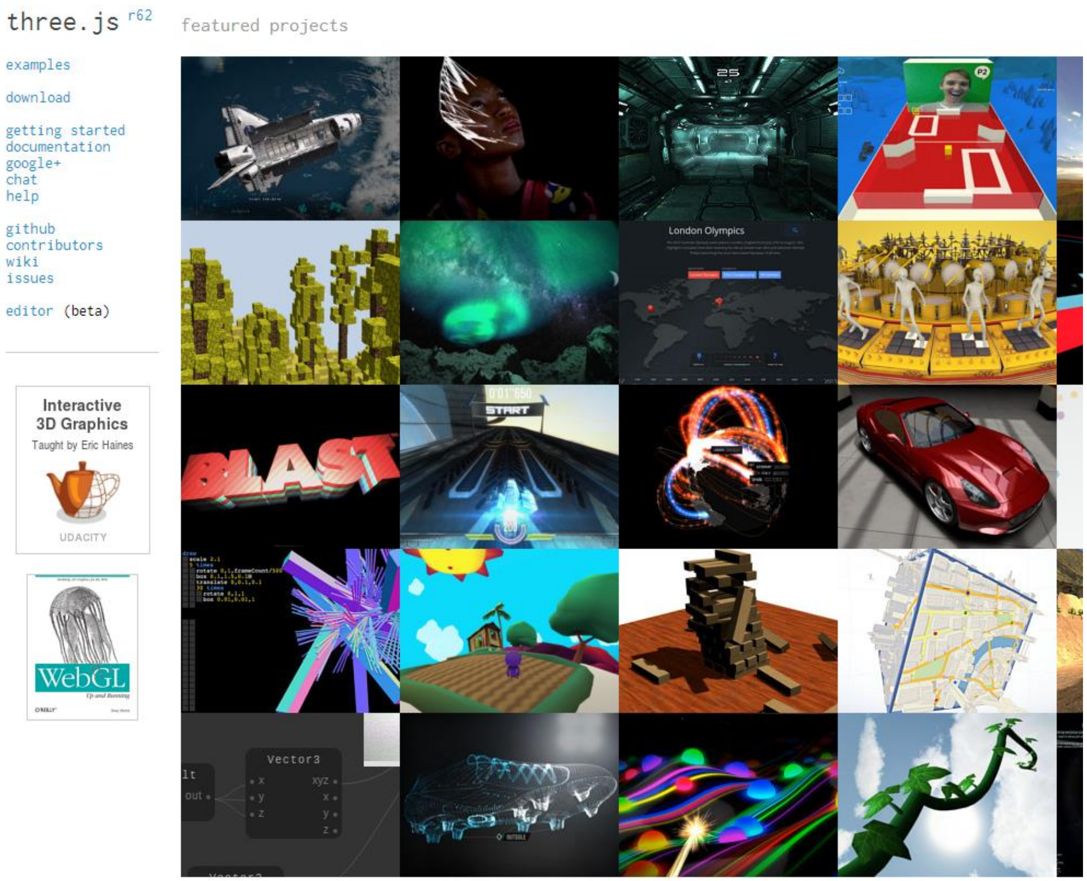
By Raymond Pang

WebGL Programming

- Programming with purely WebGL API 1.0 is painful
  - Some common functions in OpenGL are missing
- Three js
  - Wrapper of WebGL
  - Include useful classes and objects for Graphics programming
  - Large number of developer and resources online

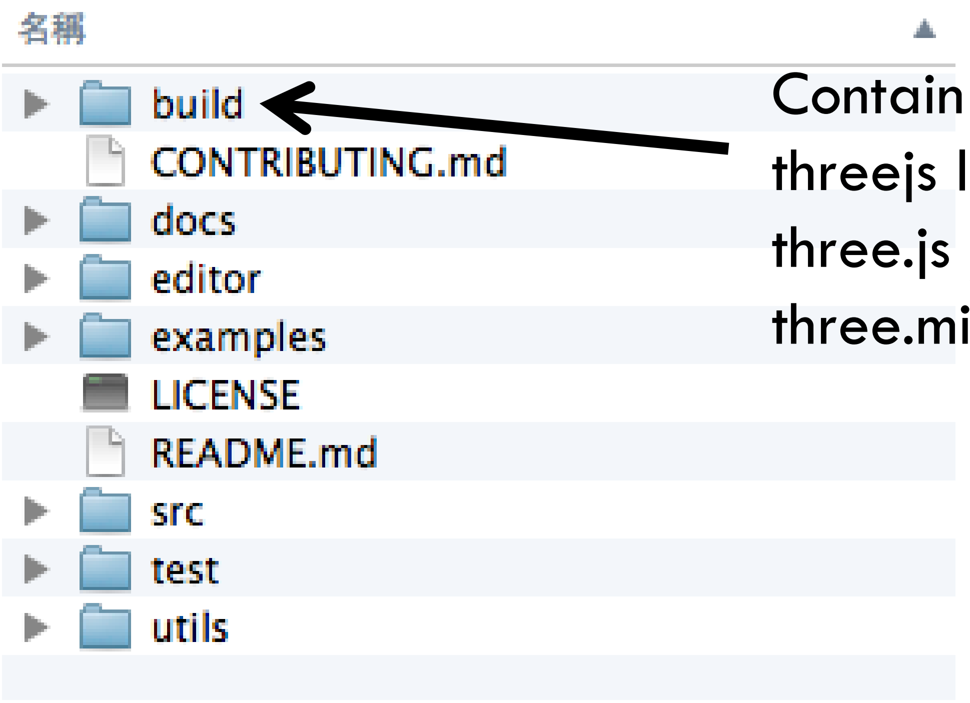
Three.js

- The API includes features :
  - Mesh loader
  - Basic geometry
  - Scene graph
  - Lighting
  - Material
  - Texture mapping
  - Hardware accelerated Shaders



Download the Library

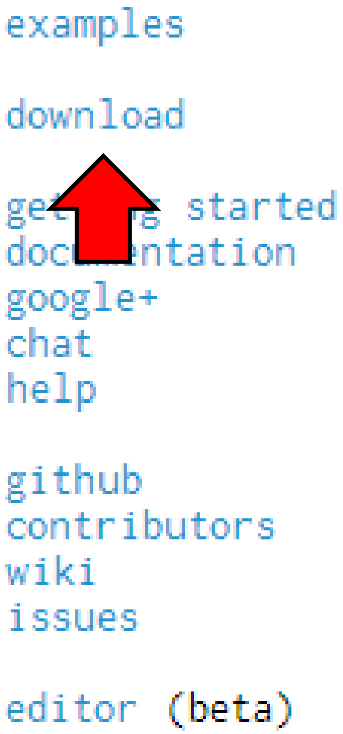
- □ Download the lastes ThreeJS library
- □ Unzip the package will see a folder structure like:



名稱

```
▶ build
  CONTRIBUTING.md
▶ docs
▶ editor
▶ examples
  LICENSE
  README.md
▶ src
▶ test
▶ utils
```

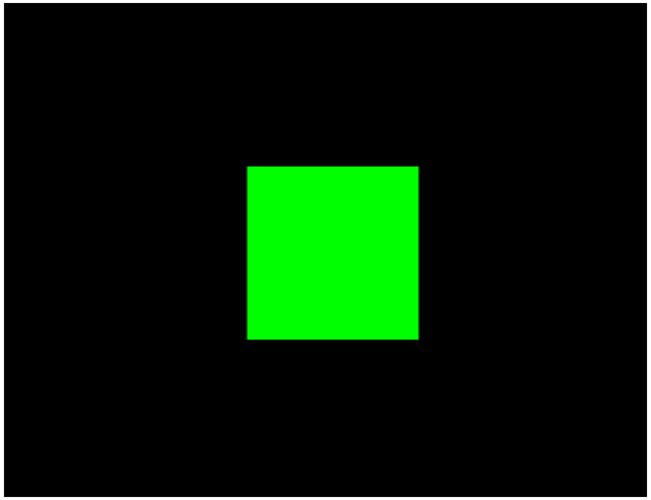
Contain the threejs library: three.js / three.min.js



```
examples

download

getting started
documentation
google+
chat
help

github
contributors
wiki
issues

editor (beta)
```

Basic Framework

- Several step are involved to setup ThreeJS
- We will base on the sample code “HelloGL.htm”
  - Draw a simple green cube
- But first, we need to create :
  - A renderer,
  - A scene, and
  - A camera



HelloGL.htm

- Only a very short Script

```
<script>
var renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
renderer.setClearColor(0x000000, 1);
document.body.appendChild(renderer.domElement);

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

var geometry = new THREE.CubeGeometry(1,1,1);
var material = new THREE.MeshBasicMaterial({color: 0x00ff00});
var cube = new THREE.Mesh(geometry, material);
scene.add(cube);
camera.position.z = 5;
renderer.render(scene, camera);
 </script>
```

Include the ThreeJS library

- First to include the library of ThreeJS which is defined in three.js or three.min.js

```
<script src="three.min.js"></script>
```

- three.min.js is a minified verison of three.js

Renderer

- Create WebGL Renderer from ThreeJS (there are other options with ThreeJS for browsers without supporting)

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

- We can set the rendering size with “setSize” method

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

  - Set as the same size of the window

Renderer

□ Can set the background color of scene

```
renderer.setClearColor(0x000000, 1);
```

□ Add the renderer element to the HTML, i.e. the <canvas> element the renderer uses to display

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

Scene and Camera

- Create the scene graph in ThreeJS

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

  - We will show how to add object into the scene later

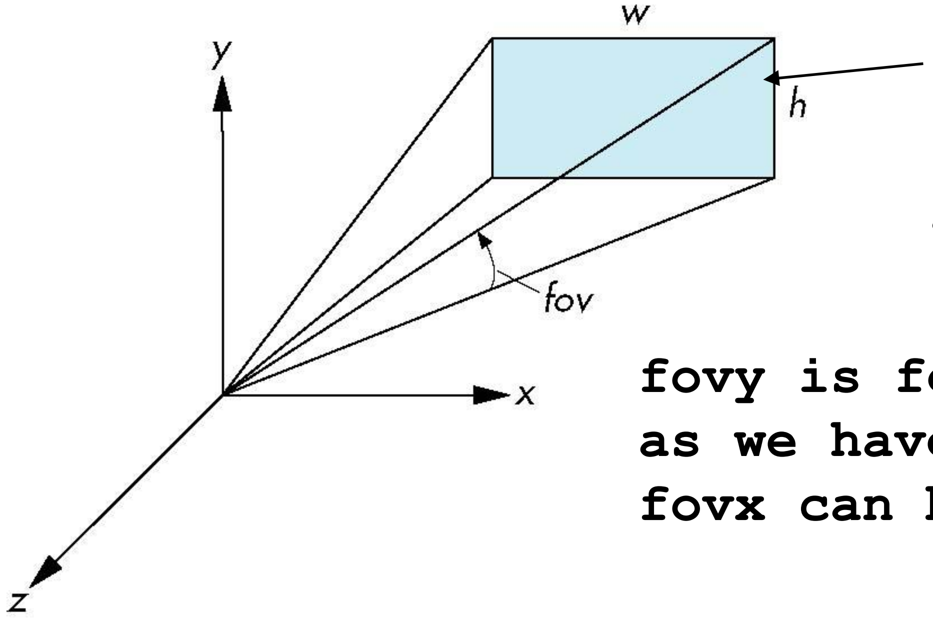
- Create a camera

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

PerspectiveCamera

- We can define a camera with perspective projection by invoking“THREE.PerspectiveCamera”:

PerspectiveCamera(fovy, aspect, zNear, zFar);



```
aspect = w/h
```

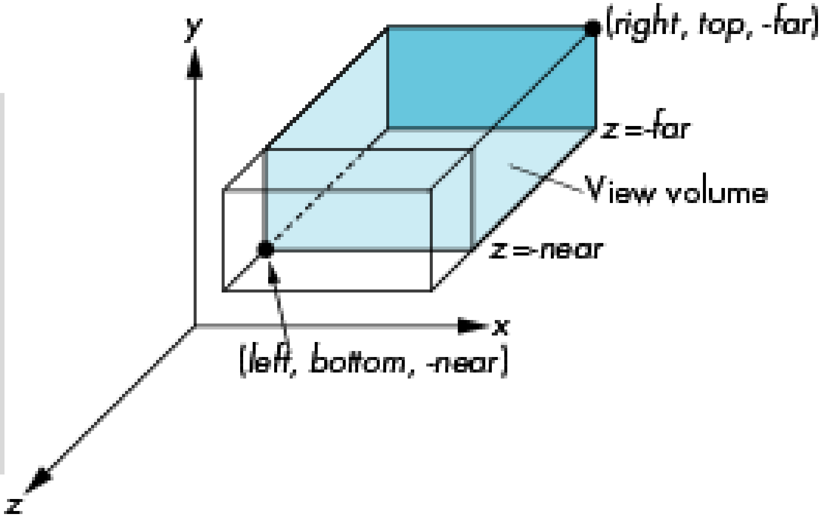
```
fovy is fov in y direction,
as we have aspect ratio,
fovx can be determined
```

Orthographic Projection

□ There is alternative way of projection which is orthographic projection

□ In ThreeJS, we can invoke

```
OrthographicCamera(
left, right,
top, bottom,
 near, far )
```



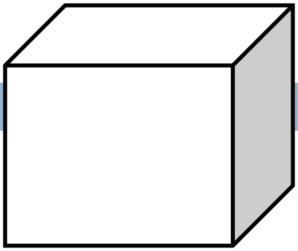
for orthographic projection matrix.

Camera

- By default, when a camera (or other 3D object) is added to the scene, it is placed at
  - (0,0,0) : origin
  - Facing -Z
- If our cube is also placed at (0,0,0), then..
  - We can not see it !!!
- Move our camera out of origin

```
camera.position.z = 5;
```

Create Object in ThreeJS

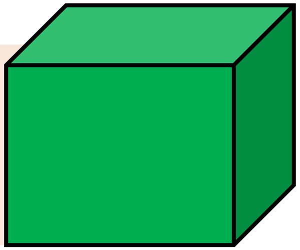


- Create Cube Geometry
  - an object that contains all the points (vertices) and fill (faces) of the cube

```
var geometry = new THREE.CubeGeometry(1,1,1);
```

  - The cube is 1x1x1 unit large
- Create the material (simple material with color in green)

```
var material = new
THREE.MeshBasicMaterial({color: 0x00ff00});
```



Create Object in ThreeJS

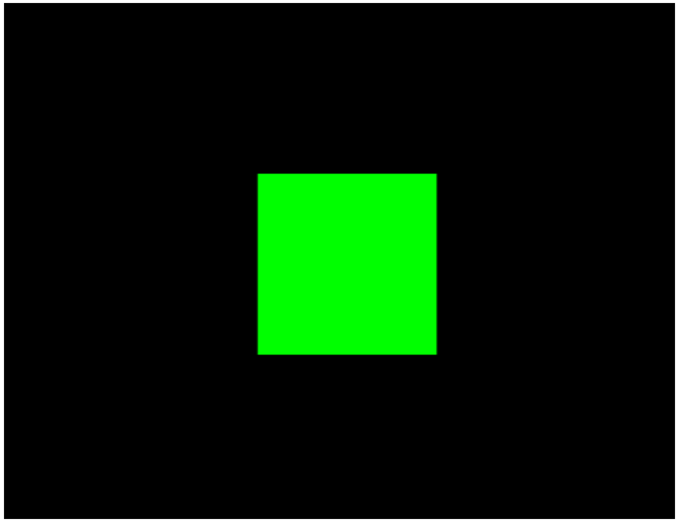
- □ Create a Mesh object that takes a geometry, and applies a material to it

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

- □ Finally, add it to our scene, by default, it add to the origin in the scene (i.e. 0,0,0)

```
scene.add(cube);
```

Then, we are done!
Much simpler than pure WebGL

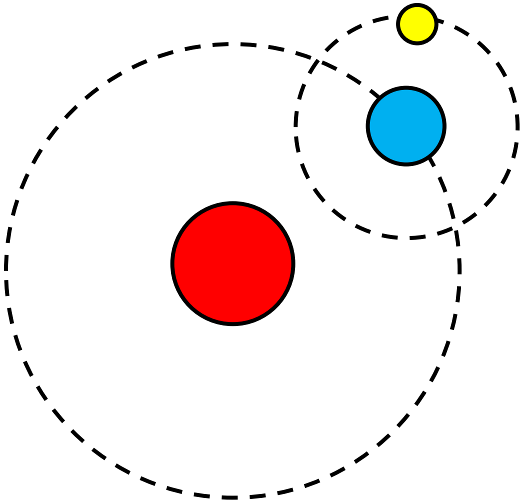


The Planet Orbit

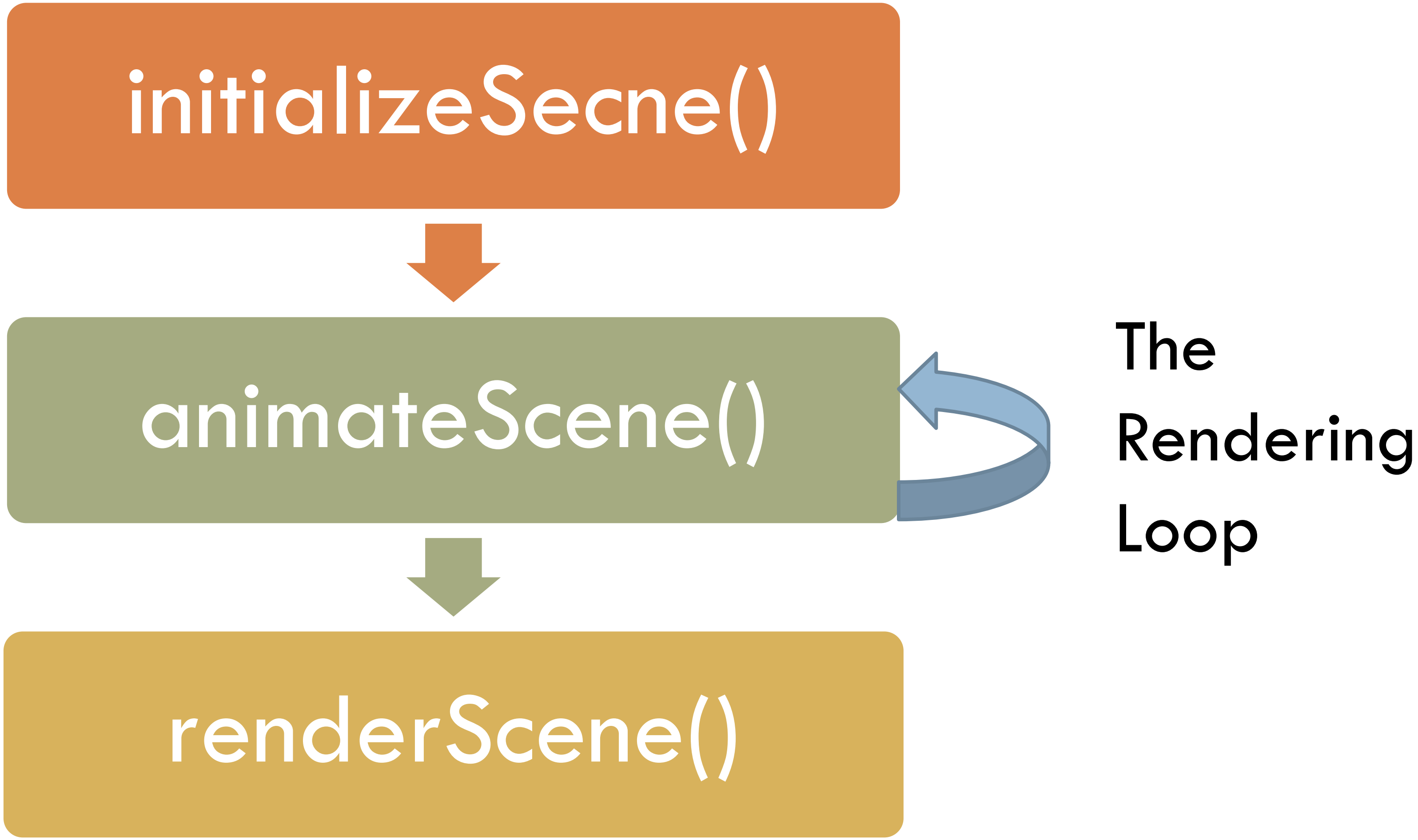
An Animated Example

Planet Orbits Example

- Last example draws only a static cube
- In PlanetOrbit, we will animate object drawn
- This example is trying to draw 3 planets:
  - Sun
  - Earth
  - Moon
- Earth rotated about Sun
- Moon rotated about the Earth



Structure of the Script



initializeSecne()

animateScene()

renderScene()

The Rendering Loop

Structure of the Script

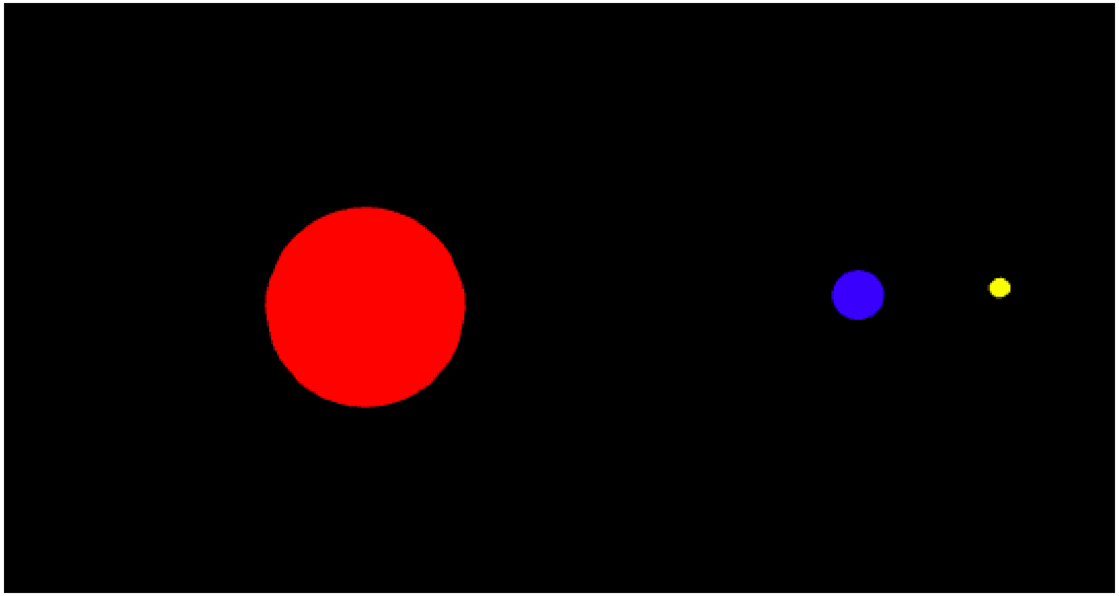
- initializeScene
  - Setup the camera
  - Create all the objects (the planets)
  - Setup the scene
- animateScene
  - Perform movements / rotations on 3D objects
  - Schedule to invoke itself again (form a loop)
    - requestAnimationFrame
- renderScene

Creating the Planets

- □ In our example, we create the planets using SphereGeomety, the code fragments are as follow:

```
new THREE.SphereGeometry(0.2, 20, 20) // the Moon
new THREE.SphereGeometry(0.5, 20, 20) // the Earth
new THREE.SphereGeometry(2.0, 20, 20) // the Sun
```

- □ Spheres with different radius



SphereGeometry

- The Sphere Geometry is defined as:

```
SphereGeometry(radius, widthSegments, heightSegments,
 phiStart, phiLength, thetaStart,thetaLength)
```

radius — sphere radius. Default is 50.
widthSegments — number of horizontal segments. Minimum value is 3, and the default is 8.
heightSegments — number of vertical segments. Minimum value is 2, and the default is 6.
phiStart — specify horizontal starting angle. Default is 0.
phiLength — specify horizontal sweep angle size. Default is Math.PI * 2.
thetaStart — specify vertical starting angle. Default is 0.
thetaLength — specify vertical sweep angle size. Default is Math.PI.

Creating the Planets

- Also define different colors for the planets

```
new THREE.MeshBasicMaterial( { color: 0xffff00 }) // Yellow
new THREE.MeshBasicMaterial( { color: 0x0000ff }) // Blue
new THREE.MeshBasicMaterial( { color: 0xff0000 }) // Red
```

- Finally, create a mesh with the geometry and material, e.g. for the moon:

-
```
moonMesh = new THREE.Mesh(
new THREE.SphereGeometry(0.2, 20, 20),
new THREE.MeshBasicMaterial( { color: 0xffff00 }));
```

Creating the Planets

- We have divided the code into 3 different methods
  - drawSun(…)
  - drawEarth(…)
  - drawMoon(…)
- The code are more or less similar, just to call gluSphere with different radius and colors
- And they are being invoked in the same order as above in the display method

3D Transformation in ThreeJS

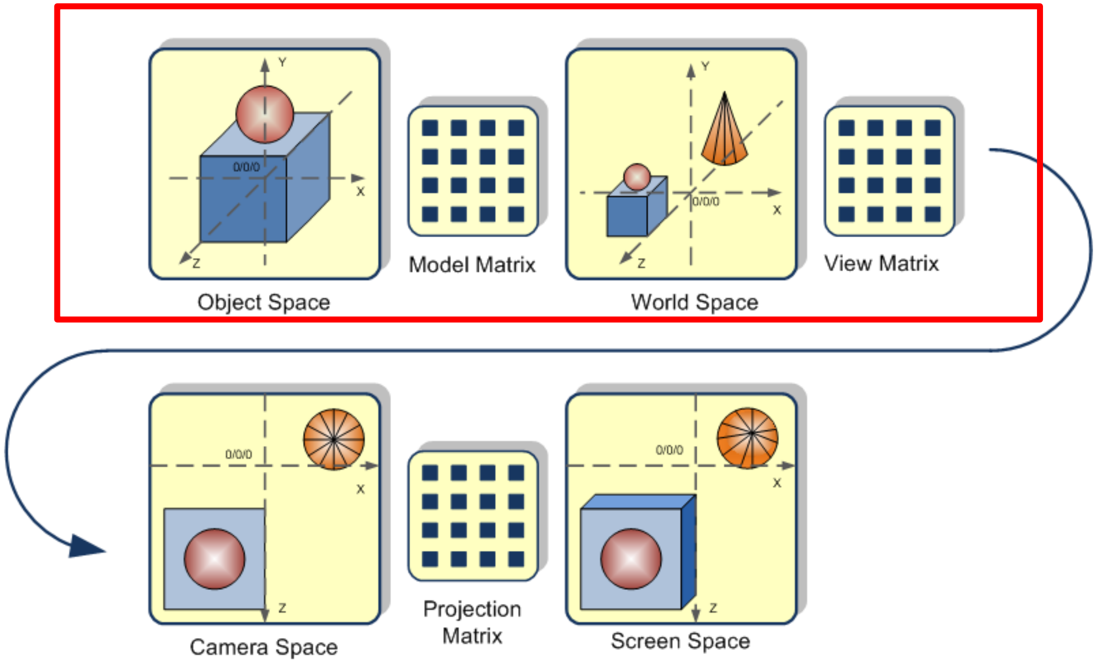
We can apply the 3 common rigid transformation to Object3D objects in ThreeJS

- Rotation
  - rotateOnAxis(axis, angle)
- Scale
  - scale.set(sx, sy, sz)
- Translation
  - translateX(dx)
  - translateY(dy)
  - translateZ(dz)

All these methods help you to create a transformationmatrix

3D Transformation in ThreeJS

- You can also modify the matrix directly
  - applyMatrix(matrix)
- This is corresponding to the **ModelView** Matrix in OpenGL, which combined both model and view matrices



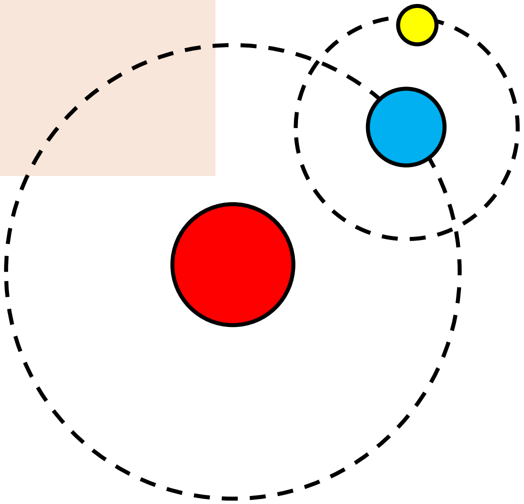
Rotating the Planets

- Remember we would like to rotate the Earth and Moon
- Therefore, to rotate the Earth, we can try:

```
earthMesh.rotateOnAxis(
new THREE.Vector3(0, 1, 0),
0.075);
```

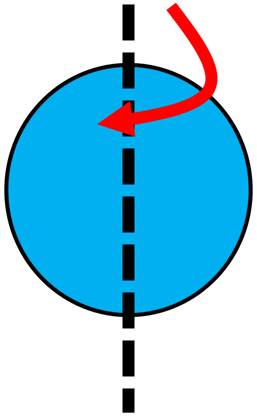
Rotate about Y-axis
(i.e. rotate horizontally

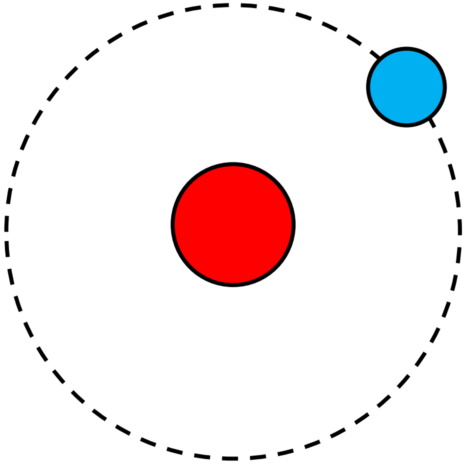
Rotate in 0.075 radian,
around 4 degree



Rotating the Planets

- However, you will find the Earth is only self rotating (i.e. rotate in its own axis)
  - Because RotateOnAxis only works on object space
- It is not what we want, we would like rotation about the Sun





Rotating about the Sun

- As a result, we need to make the rotation of the Earth in the Sun’s space (or world space)
- First, we need to create a dummy object called “sunSpace”

```
sunSpace = new THREE.Object3D();
```

- Then, add the Earth into this space

```
sunSpace.add(earthMesh);
```

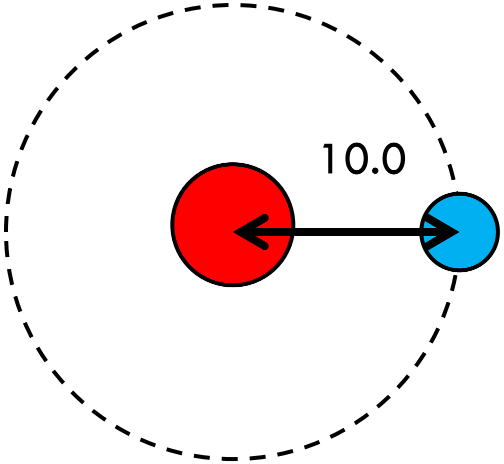
Rotating about the Sun

- The most important is that we have to translate the Earth 10.0 units away from the Sun (which is at origin (0,0,0))

```
earthSpace.position.set(10.0, 0.0, 0.0);
```

- Finally, in the animateScene, rotate the sunSpace instead of earthMesh:

```
sunSpace.rotateOnAxis(new
THREE.Vector3(0, 1, 0), 0.075);
```



Rotating about the Earth

- Moon is rotating about Earth
- The solution is similar :
  - Create the earthSpace
  - Create moonMesh
  - Add moonMesh to earthSpace
  - Move the moonMesh out of the origin

```
earthSpace = new THREE.Object3D();
moonMesh = new THREE.Mesh(…);
earthSpace.add(moonMesh);
moonMesh.position.set(3.0, 0.0, 0.0);
```

Rotating about the Earth

- In animateScene, rotate the earthSpace instead

```
earthSpace.rotateOnAxis(new
THREE.Vector3(0, 1, 0), 0.075);
```

- We need further changes:
  - we have to add earthSpace to sunSpace too.
  - Also it move as what the earthMesh did

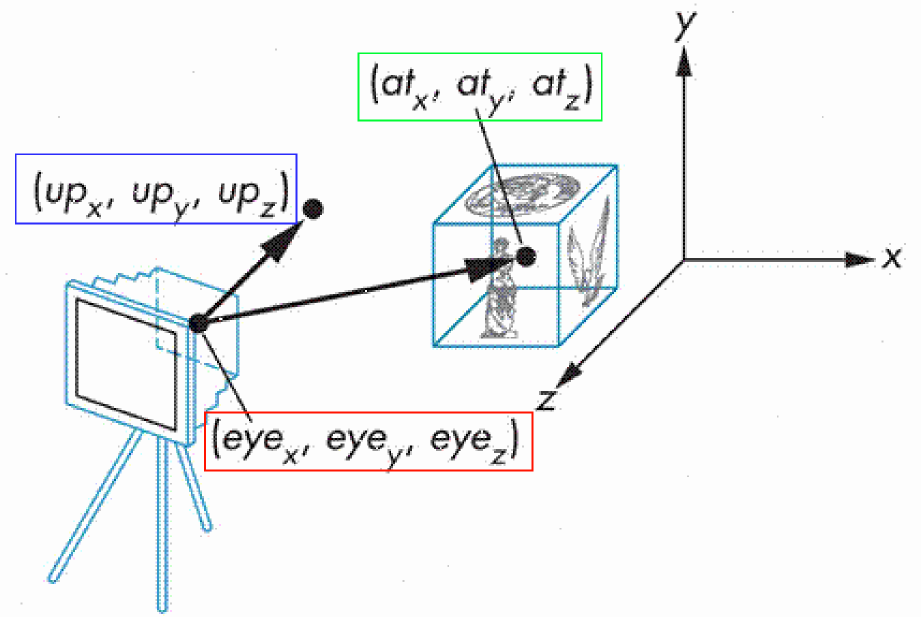
```
sunSpace.add(earthSpace);
…
earthSpace.position.set(10.0, 0.0, 0.0);
```

Camera Looking Direction

- We can move our camera similar to what you done for normal objects
- While, we have not fix the viewing direction of your camera

```
camera.lookAt( vector )
```

- Vector is the target viewing position



Example

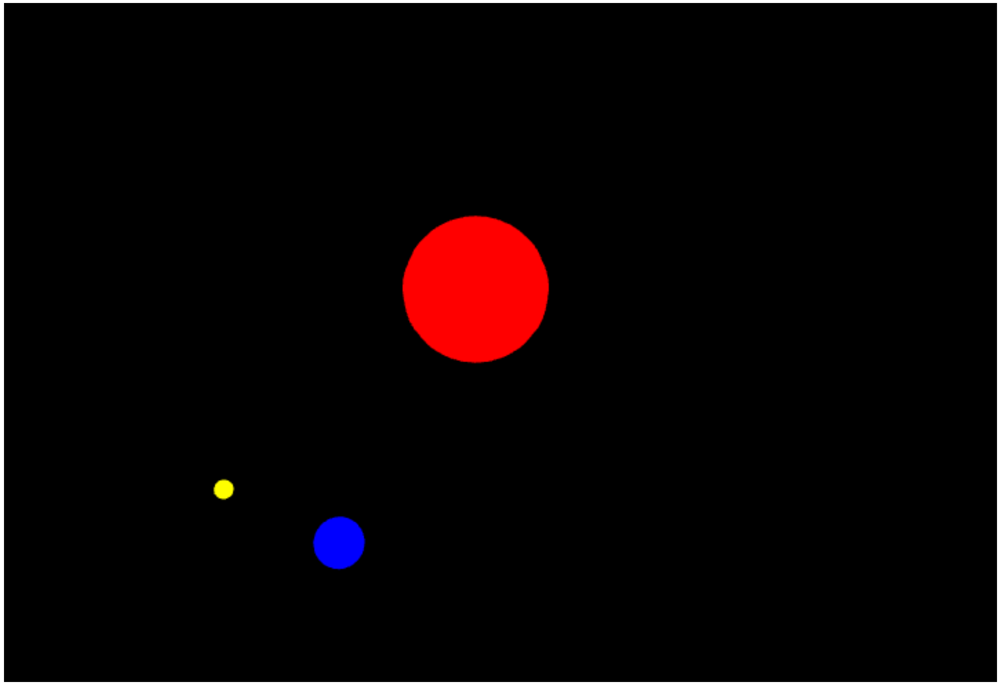
- In our sample program, perform the following code to place the camera and make it look at a target positon :

```
camera.position.set(0, 15, 25);

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

- Note that scene.position is also the position of the Sun in our example

Planet Example

- Try the sample code "Planet.html" and see the effect
- See if you can understand all the code as a whole



Summary

- Introduction of a popular WebGL wrapper: ThreeJS
- Go through a simple static example, and an animated scene
- A renderer, scene and camera are the first items to create
- We can place object, e.g. cube or sphere, inside the scene
- Simple animation can be done by applying rigid transformations continuously in the rendering loop