Introduction to WebGL and Three.js

WebGL (low level)

- <canvas> has 3D option—WebGL—for low-level 3D graphics
- WebGL ≈ OpenGL ES 2.0 (embedded systems)
- Supported by all major browsers
- Working group: Apple, Google, Mozilla, Opera (not MS)
- Low-level API, not for faint of heart
  (Most users will use higher-level libraries)
- Good book: *WebGL: Up and Running*

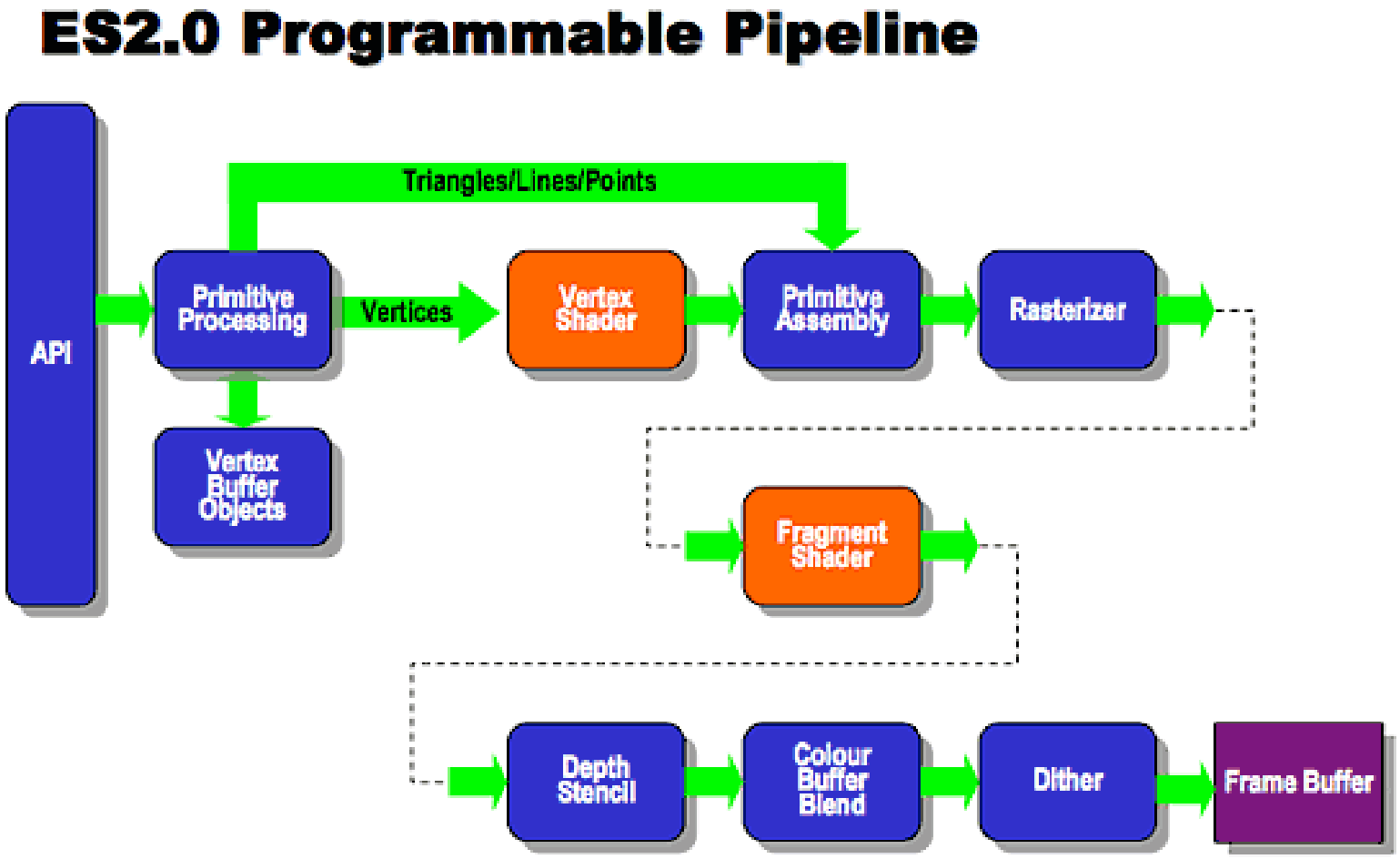
WebGL → Three.js

- WebGL is low-level; 3D is hard work
- Need libraries for higher-level capabilities
  - Object models
  - Scene graphs
  - Display lists
- We'll rather start with raw WebGL examples, then move quickly to Three.js

WebGL overview

- Steps to 3D graphics:
  - Create a canvas element
  - Obtain drawing context
  - Initialize the viewport
  - Create buffers of data (vertices) to be rendered
  - Create model and view matrices
  - Create shaders
  - Draw

Graphics Pipeline



Three.js Features

- Renderers: <canvas>, <svg> and WebGL; effects: anaglyph, crosseyed, stereo and more
- Scenes: add and remove objects at run-time; fog
- Cameras: perspective and orthographic; controllers: trackball, FPS, path and more
- Animation: morph and keyframe
- Lights: ambient, direction, point and spot lights; shadows: cast and receive
- Materials: Lambert, Phong and more - all with textures, smooth-shading and more
- Shaders: access to full WebGL capabilities; lens flare, depth pass and extensive post-processing library
- Objects: meshes, particles, sprites, lines, ribbons, bones and more - all with level of detail
- Geometry: plane, cube, sphere, torus, 3D text and more; modifiers: lathe, extrude and tube
- Loaders: binary, image, JSON and scene
- Utilities: full set of time and 3D math functions including frustum, Quaternion, matrix, UVs and more
- Export/Import: utilities to create Three.js-compatible JSON files from within: Blender, CTM, FBX, 3D Max, and OBJ
- Support: API documentation is under construction, public forum and wiki in full operation
- Examples: More than 150 files of coding examples plus fonts, models, textures, sounds and other support files

Three.js

- Written by Mr.doob aka Cabello Miguel of Spain
- Perceived leader of WebGL frameworks
- Documentation is thin, but 150 examples

Some books

- Brian Danchilla, Beginning WebGL for HTML5, Apress, 2012. Pure WebGL, shaders, Three.js, a little bit of physics.
- Diego Cantor, Brandon Jones, WebGL Beginner's Guide, Packt, 2012. Pure WebGL with shaders.
- Andreas Anyuru, Professional WebGL Programming, wrox, ??. Pure WebGL, math, matrices, etc.
- Tony Parisi, WebGL Up and Running, O'Reilly, 2012. Pure WebGL + Three.js. Classic.
- Tony Parisi, Programming 3D Applications with HTML5 and WebGL, O'Reilly 2014. Similar to previous position, but enhanced.
- Sumeet Arora, WebGL Game Development, Packt, 2014. Pure WebGL with shaders + Three.js
- Jos Dirksen, Learning Three.js: The JavaScript 3D Library for WebGL, Packt, 2013. Based on examples.
- Jos Dirksen, Three.js Essentials, Packt, 2014. Similar to previous.
- Isaak Sukin, Game Development with Three.js, Packt, 2013. Short (118 pages) but nice.

First Three.js Program

- A document to draw on:

```
<html>
<head>
<title> First Three.js applicaton </title>
<style> canvas {width: 100%; height: 100%} </style>
</head>
<body>
<script
src = "three.js" >
</script>
<script>
// Our Javascript code will go here
</script>
</body>
</html>
```

Three.js basics

- To display something with Three.js we need:
  - A scene
  - A camera
  - A renderer

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

var renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement);
```

Adding geometry

- Now we need to add an object to the scene:

```
var geometry = new THREE.BoxGeometry(1,1,1);
var material = new THREE.MeshBasicMaterial({color:
0x00ff00});
var cube = new THREE.Mesh(geometry, material);
scene.add(cube);

camera.position.z = 5;
cube.rotation.x = .5;
cube.scale.x = 3.


camera.positon.set(0,4,5);
```

Render the scene

```
function render() {
requestAnimationFrame(render);

cube.rotation.x += 0.1;
cube rotaton.y += 0.1;

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

Three.JS overview

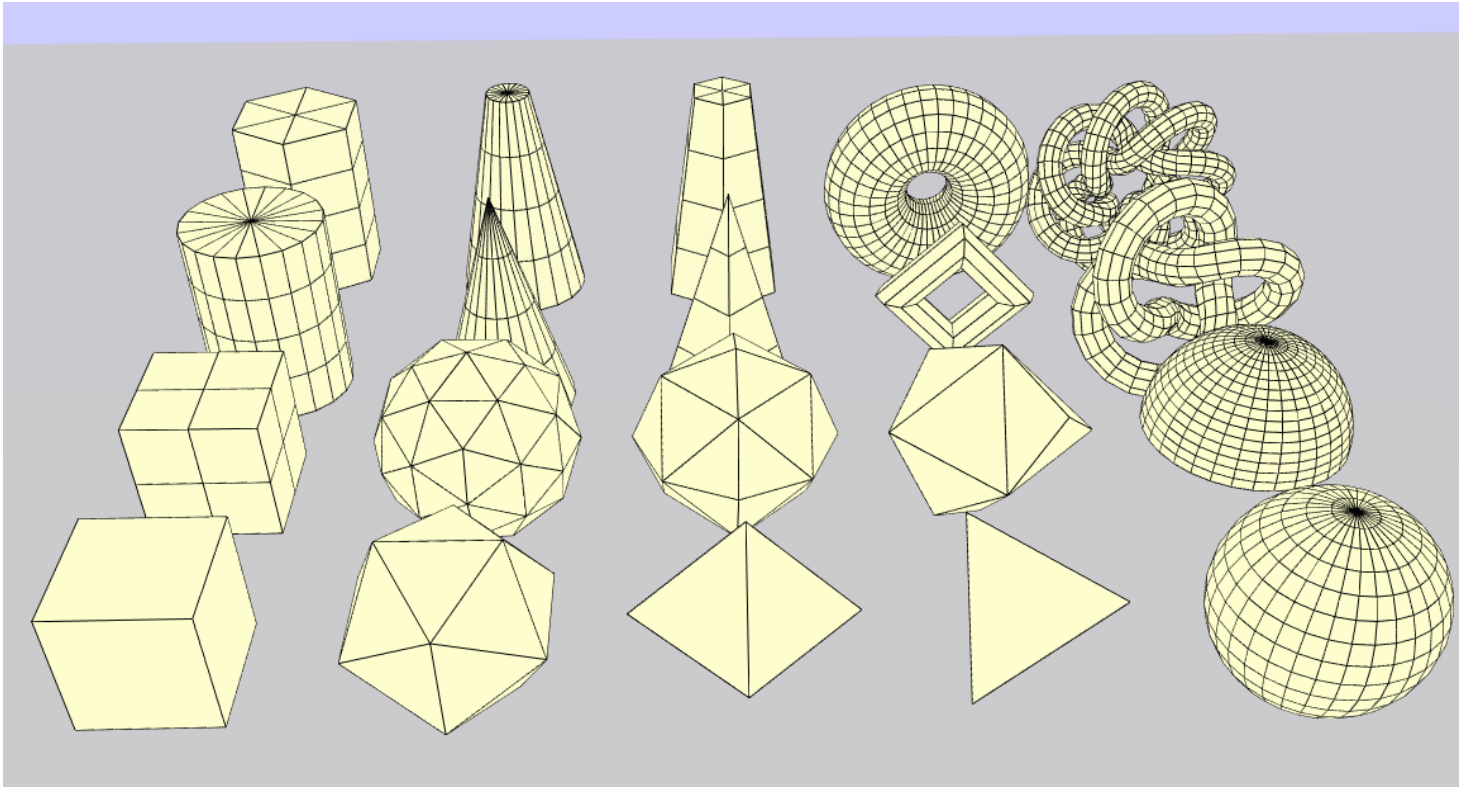
- Documentation thin, incomplete. [More examples]
- Types of objects:
  - Cameras (orthographic, perspective)
  - Controllers (firstperson, fly, path, roll, trackball)
  - Scenes
  - Renderers (WebGL, Canvas, SVG)
  - Objects (mesh, line, particle, bone, sprite, etc)
  - Geometries (cube, cylinder, sphere, lathe, text, etc)
  - Lights,
  - Materials
  - Loaders
  - Animation (animationHandler, morphTarget)
  - Collision detection

Project: animated flower

- Make a 3D flower
- Simple version:
  - Doesn't have to be realistic
  - Use a function for petals, etc.
  - Make it rotate or move
  - Trackball controller
- Fancier version:
  - More realistic
  - Animated, e.g. bends in the wind, slider to open/close flower, etc.

Geometry

- How would you create geometry?



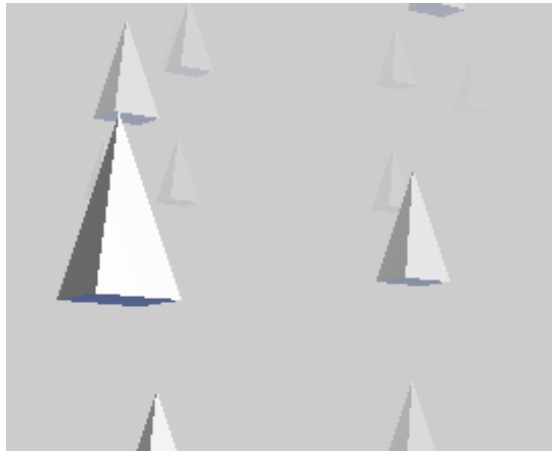
Creating Geometry

- Use an object like CubeGeometry, CylinderGeometry, PolyhedronGeometry, etc to create an object
- Add it to your scene

- Documentation:
- Check out example (or look at source code)

Creating Geometry

```
scene = new THREE.Scene();
scene.fog = new THREE.FogExp2( 0xcccccc, 0.002 );

var geometry = new THREE.CylinderGeometry( 0, 10, 30, 4, 1 );
var material =  new THREE.MeshLambertMaterial( { color:0xffffff, shading: THREE.FlatShading } );

for ( var i = 0; i < 500; i ++ ) {

    var mesh = new THREE.Mesh( geometry, material );
    mesh.position.x = ( Math.random() - 0.5 ) * 1000;
    mesh.position.y = ( Math.random() - 0.5 ) * 1000;
    mesh.position.z = ( Math.random() - 0.5 ) * 1000;
    mesh.updateMatrix();
    mesh.matrixAutoUpdate = false;
    scene.add( mesh );

}
```



Virtual Trackball?

- How would you figure out how to set up a virtual trackball?

Trackball controller

- Use the TrackballControls camera controller
- Documentation
- Check out example (or look at source code)

Trackball controller

```
camera = new THREE.PerspectiveCamera( 60, window.innerW
camera.position.z = 500;

controls = new THREE.TrackballControls( camera );

controls.rotateSpeed = 1.0;
controls.zoomSpeed = 1.2;
controls.panSpeed = 0.8;

controls.noZoom = false;
controls.noPan = false;

controls.staticMoving = true;
controls.dynamicDampingFactor = 0.3;

controls.keys = [ 65, 83, 68 ];

controls.addEventListener( 'change', render );
```

Lighting?

- Lights: AmbientLight, DirectionalLight, PointLight, SpotLight
- Documentation: there is some!
- Check out an example anyway

Lighting in Three.js

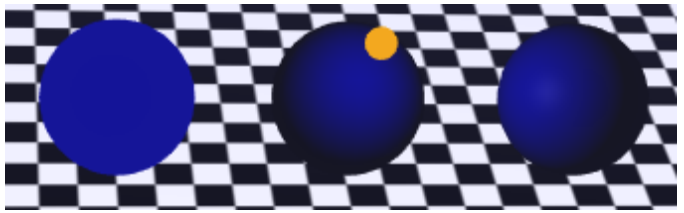
```
var light = new THREE.PointLight( 0xff2200 );
light.position.set( 100, 100, 100 );
scene.add( light );

var light = new THREE.AmbientLight( 0x111111 );
scene.add( light );


var geometry = new THREE.CubeGeometry( 100, 100, 100 );
var material = new THREE.MeshLambertMaterial( { color: 0xff
```

Shading and material types

- Material types:
  - MeshBasicMaterial
  - MeshLambertMaterial
  - MeshPhongMaterial

- Parameters/properties:
  - Color, wireframe, shading, vertexColors, fog, lightMap, specularMap, envMap, skinning, morphTargets

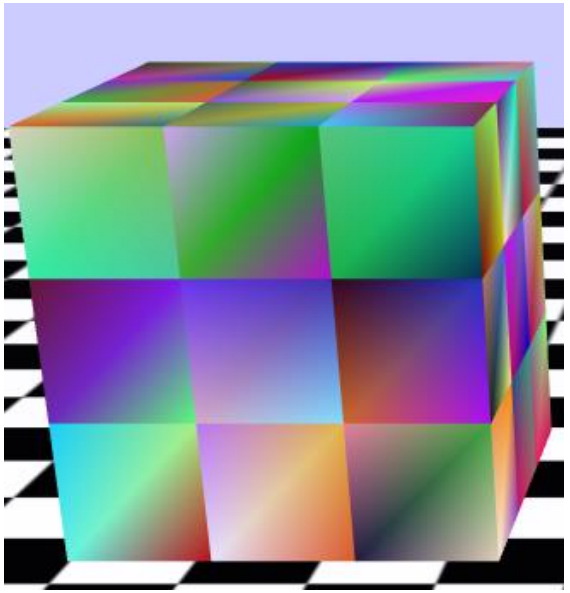
Shading and material types



```
// Sphere parameters: radius, segments along width, segments along height
var sphereGeom =  new THREE.SphereGeometry( 50, 32, 16 );

// Three types of materials, each reacts differently to light.
var darkMaterial = new THREE.MeshBasicMaterial( { color: 0x000088 } );
var darkMaterialL = new THREE.MeshLambertMaterial( { color: 0x000088 } );
var darkMaterialP = new THREE.MeshPhongMaterial( { color: 0x000088 } );

// Creating three spheres to illustrate the different materials.
// Note the clone() method used to create additional instances
//    of the geometry from above.
var sphere = new THREE.Mesh( THREE.GeometryUtils.clone(sphereGeom), darkMaterial );
sphere.position.set(-150, 50, 0);
scene.add( sphere );

var sphere = new THREE.Mesh( THREE.GeometryUtils.clone(sphereGeom), darkMaterialL );
sphere.position.set(0, 50, 0);
scene.add( sphere );

var sphere = new THREE.Mesh( THREE.GeometryUtils.clone(sphereGeom), darkMaterialP );
sphere.position.set(150, 50, 0);
scene.add( sphere );
```

Gradients

- Use vertex colors



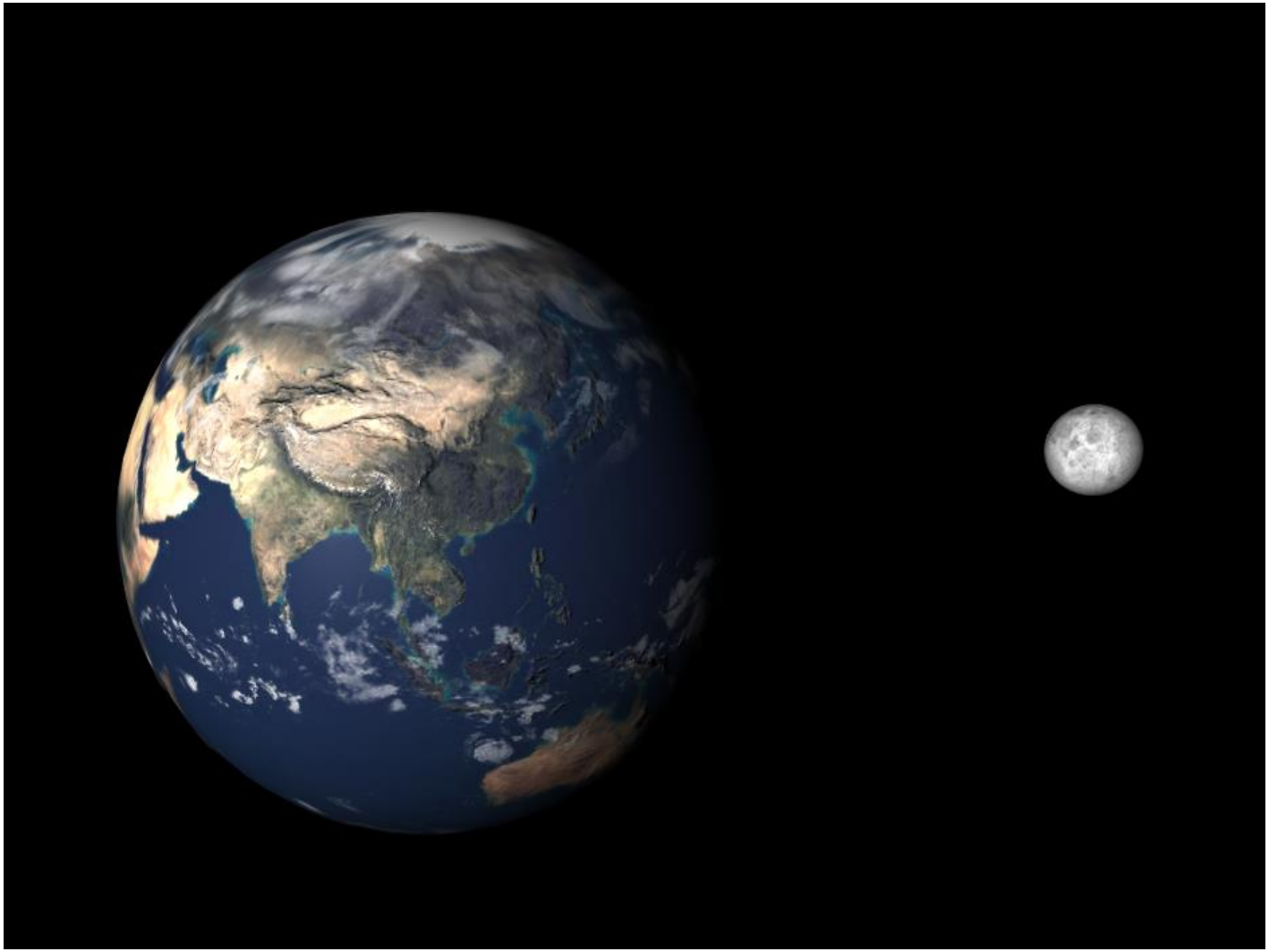
```
face   = cubeGeometry.faces[ i ];
// determine if current face is a tri or a quad
numberOfSides = ( face instanceof THREE.Face3 ) ? 3 : 4;
// assign color to each vertex of current face
for( var j = 0; j < numberOfSides; j++ )
{
    vertexIndex = face[ faceIndices[ j ] ];
    // initialize color variable
    color = new THREE.Color( 0xffffff );
    color.setHex( Math.random() * 0xffffff );
    face.vertexColors[ j ] = color;
}
```

Moving your objects around

- object.positon.set(x, y, z)
- object.rotation.x = 90 * Math.PI / 180
  - Rotations occur in the order x, y, z
  - With respect to object's internal coord system
  - If there is an x-rotation, y and z rotations may not be lined up with world axes
- Object properties (parent-relative):
  - Position
  - Rotation
  - Scale

Object Hierarchy

- What if you want to create an object with parts?
- Object transform hierarchy
  - Scene: top-level object in hierarchy
  - Can add objects to other objects
  - Move or rotate one part: its children move as well



```
    var geometry = new THREE.SphereGeometry(Moon.SIZE_IN_EARTHS,
        32, 32);
    var texture = THREE.ImageUtils.loadTexture(MOONMAP);
    var material = new THREE.MeshPhongMaterial( { map: texture,
        ambient:0x888888 } );
    var mesh = new THREE.Mesh( geometry, material );

    // Let's get this into earth-sized units (earth is a unit sphere)
    var distance = Moon.DISTANCE_FROM_EARTH / Earth.RADIUS;
    mesh.position.set(Math.sqrt(distance / 2), 0,
        -Math.sqrt(distance / 2));

    // Rotate the moon so it shows its moon-face toward earth
    mesh.rotation.y = Math.PI;

    // Create a group to contain Earth and Satellites
    var moonGroup = new THREE.Object3D();
    moonGroup.add(mesh);

    // Tilt to the ecliptic
    moonGroup.rotation.x = Moon.INCLINATION;

    // Tell the framework about our object
    this.setObject3D(moonGroup);

    // Save away our moon mesh so we can rotate it
    this.moonMesh = mesh;
}
```

Morphing

- Image/video morphing: smoothly shifting from one image to another
- First popularized in a Michael Jackson video
- Method for video: a combination of
  - Identifying corresponding points in images over time
  - Warping both images, gradually moving control points from location in first image to location in the second
  - Cross-fading from first image sequence to second

3D Morphing

- Define 3D before and after shapes
- Linear interpolation of point locations from first setting to second

Morphing in Three.js

- Create geometry
- Move vertices to create “morph targets”
  - geometry.morphTargets.push(
    { name: “target” + i, vertices: vertices });
- Set influence
  - mesh.morphTargetInfluences[0]=0.3;
  - mesh.morphTargetInfluences[1]=0.7;
- Can also set up animations that can be played (people walking, etc)

Morphing in Three.js

- MorphAnimMesh documentation:“todo”
- See morph target example

Summary

- WebGL is OpenGL ES in the browser
- Distributed and SIMD-like programming
- Vertex and fragment shaders
- WebGL graphics pipeline
- Depth buffer algorithm for hidden surface removal
- Three.js is nice!