Chapter 5

Three.js: A 3D Scene Graph API

Dr. Terence van Zyl

University of the Witwatersrand

6th April 2016

Outline

1. Three.js Basics
    - Scene, Renderer, Camera
    - THREE.Object3D
    - Object, Geometry, Material
    - Lights
    - A Modelling Example

Three.js Basics

Introduction

Three.js is an object-oriented JavaScript library for 3D graphics. It is an open-source project created by Ricardo Cabello (who goes by the handle "mr.doob", http://mrdoob.com/), with contributions from other programmers.

Scene, Renderer, Camera

Introduction

Three.js works with the HTML <canvas> element, the same thing that we used for 2D graphics. Three.js is an object-oriented scene graph API. The basic procedure is to build a scene graph out of three.js objects, and then to render an image of the scene it represents. Animation can be implemented by modifying properties of the scene graph between frames.

Getting Started

Including three.min.js

- You need to download the file three.js-master.zip from `http://threejs.org/`
- Put the contents of the build folder file three.min.js in the same directory as your html files.
- Add the below line to your html.

THREE js code including three.min.js

```
<script src="three.min.js"/>
<script>
        var scene, renderer, camera;
</script>
```

Getting Started

Definitions (THREE functions)

THREE.Scene Holdes all the objects in your world including Cameras

THREE.Camera Provides you viewing & projection transform

THREE.WebGLRenderer Provides your viewport transform

Warning

Almost all of the three.js classes and constants that we will use are properties of an object named THREE, and their names begin with "THREE.". The THREE may be left out in the notes for conciseness.

THREE.Scene

THREE js code

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

Notes

Scene is in fact a scene graph with the top level THREE.Scene() representing your world node.

Definitions (THREE.Scene functions)

.add(item) adds cameras, lights, and objects to the scene

.remove(item) removes an item from the scene

THREE.Camera

THREE js code setting up a camera

```
camera = new THREE.PerspectiveCamera(45,canvas.width/canvas.height,1,100);
```

Notes

There are two types of cameras orthographic and perspective projections.

Definitions (THREE functions)

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

.PerspectiveCamera(fieldOfViewAngle,aspect,near,far) see gluPerspective()

THREE.WebGLRender

Example (THREE js code: Setting up a renderer)

```
renderer = new THREE.WebGLRenderer({canvas:theCanvas, antialias:true});
renderer.render(scene,camera);
```

Notes

There are in fact a number of different renderers.

Definitions (THREE methods)

WebGLRenderer(params) params is an optional dictionary

Definitions (THREE.WebGLRenderer methods)

.render(scene,camera) render scene from point of view of camera

Lets See It In Action

Demo

threejs/full-window.html

THREE.Object3D

Introduction

A three.js scene graph is made up of objects of type THREE.Object3D (including objects that belong to subclasses of that class). Cameras, lights, and visible objects are all represented by subclasses of Object3D. In fact, THREE.Scene itself is also a subclass of Object3D.

Object3D

Definitions (THREE methods)

.Object3D()	Cameras, lights, and visible objects are all represented by subclasses of Object3D.

Definitions (THREE.Object3D methods)

.add(obj)	adds object to the list of children of node.
.remove(obj)	remove an object from the list.
.parent	points to the parent of **this** object in the scene graph.
.clone()	copies **this** node, including a clone of the children.

Object3D cont.

Warning

Since Object3D is in fact a tree. If an object already has a parent when it is added as a child of node, then the object is first removed from the child list of its current parent before it is added to the child list of node.

Object3D cont.

Examples (THREE js code: setting up a scene graph)

```
var node = THREE.Object3D();
      .
      .  // Add children to node.
      .
scene.add(node);
var nodeCopy1 = node.clone();
      .
      .  // Modify nodeCopy1, maybe apply a transformation.
      .
scene.add(nodeCopy1)
var nodeCopy2 = node.clone();
      .
      .  // Modify nodeCopy2, maybe apply a transformation.
      .
scene.add(nodeCopy2);
```

Vectors

Definitions (THREE methods)

.Vector2(x,y) represents a point in 2D

.Vector3(x,y,z) represents a point in 3D

Example (THREE js code: creating a vector)

```
var v = new THREE.Vector3(17, -3.14159, 42);
```

Vectors cont.

Definitions (THREE.Vector3 methods)

.set(x,y,z) set the x, y and z component of the vector

.x, .y, .z set the x,y,z properties

Examples (THREE js code: modifying a Vector3)

```
v.set(2, 2, 2);
v.x = 10;
```

Object3D cont

Definitions (THREE.Object3D methods)

- **.position** the position of the object given as a THREE.VectorX
- **.scale** the scale that that is applied as a THREE.VectorX
- **.rotation** the rotation that is applied is a THREE.Euler
- **.translate** the translation that that is applied as a THREE.VectorX

Examples (THREE js code: changing object properties)

```
obj.scale.set(2,2,2);
obj.scale.y = 0.5;
camera.position.z = 20;
```

Euler Angles

Note

The object is first scaled, then rotated, then translated according to the values of these properties.

Note

The object is rotated first about the x-axis, then about the y-axis, then about the z-axis. (It is possible to change this order.) The value of obj.rotation is not a vector. Instead, it belongs to a similar type, THREE.Euler, and the angles of rotation are called Euler angles.

Object, Geometry, Material

Introduction

A visible object in three.js is made up of either points, lines, or triangles as well as a some geometry plus a material that determines the appearance of that geometry.

Point Clouds, Lines and Meshes

Definitions (THREE constructors)

.PointCloud(geometry,material,...) used to represents an object of GL_POINTS

.Line(geometry,material,...) used to represent an object of GL_LINES

.Mesh(geometry,material,...) used to represent an object of GL_TRIANGLES

- We pass a Geometry() and Material() object to the above constructors

Geometry and Material

Definitions (THREE constructors)

.Geometry() stores the set of vertices for the visible object

.Material() determines the appearance of that geometry

Examples (THREE js code: putting points into a Geometry)

```
var points = new THREE.Geometry();
while ( points.vertices.length < 1000 ) {
    var x = 2*Math.random() - 1;  // (between -1 and 1)
    var y = 2*Math.random() - 1;
    var z = 2*Math.random() - 1;
    if ( x*x + y*y + z*z < 1 ) { // use vector only if length is less than 1
        var pt = new THREE.Vector( x, y, z );
        points.vertices.push(pt);
    }
}
```

Point Cloud Example

Definitions (THREE constructors)

.PointCloudMaterial(...), is a subclass of Material for point clouds

Examples (THREE js code: defining a material)

```
var pointMaterial = new THREE.PointCloudMaterial( {
              color: "yellow",
              size: 2,
              sizeAttenuation: false;
          } );
```

Point Cloud Example cont.

Examples (THREE js code: tweaking the material)

```
var pointMaterial = new THREE.PointCloudMaterial();
pointMaterial.color = "yellow";
pointMaterial.size = 2;
pointMaterial.sizeAttenuation = false;
```

Examples (THREE js code: setting up the cloud)

```
var sphereOfPoints = new THREE.PointCloud( points, pointMaterial );
scene.add( sphereOfPoints );
```

Lets See That In Action

Demo

A Three.js PointCloud

More on Colour

- WebGL is pretty flexible about how colour can be passed to objects.

Examples (THREE js code: changing colour)

```
var c1 = new THREE.Color("skyblue");
var c2 = new THREE.Color(1,1,0);  // yellow
var c3 = new THREE.Color(0x98fb98);  // pale green
```

Examples (THREE js code: setting the renders clear colour)

```
renderer.setClearColor( new THREE.Color(0.6, 0.4, 0.1) );
renderer.setClearColor( "darkgray" );
renderer.setClearColor( 0x112233 );
```

A Line Strip Example

Definitions (THREE constructors)

.Line can represent either a line strip or a set of disconnected line segments—what would be called GL_LINE_STRIP or GL_LINES in OpenGL

Examples (THREE js code: setting the geometry for a traingle)

```
var lineGeom = new Geometry();
lineGeom.vertices.push( new THREE.Vector3(−2,−2,0) );
lineGeom.vertices.push( new THREE.Vector3(2,−2,0) );
lineGeom.vertices.push( new THREE.Vector3(0,2,0) );
lineGeom.vertices.push( new THREE.Vector3(−2,−2,0) );
```

A Line Strip Example cont.

Examples (THREE js code: setting the geometry for triangle)

```
lineGeom.vertices = [
    new THREE.Vector3(-2,-2,0),
    new THREE.Vector3(2,-2,0),
    new THREE.Vector3(0,2,0),
    new THREE.Vector3(-2,-2,0)
];
```

A Line Strip Example cont.

Definitions (THREE constructors)

.LineBasicMaterial(...) linbe material can be represented by an object of this type.

Examples (THREE js code: setting the line strip material)

```
var lineMat = new THREE.LineBasicMaterial( {
    color:  0xA000A0,  // purple; the default is white
    linewidth: 2       // 2 pixels; the default is 1
} );
```

A Line Strip Example cont.

Definitions (THREE constructors)

.Line(geometry,material,lineType) line material can be represented by an object of this type.

Definitions (THREE constants)

.LineStrip a set of connected lines aka GL_LINE_STRIP

.LinePieces disconnected line segments aka GL_LINES

Examples (THREE js code: creating the line strip)

```
var line = new THREE.Line( lineGeom , lineMat , THREE.LineStrip );
```

A Line Strip Example cont.

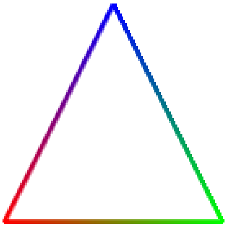


Figure: Basic line strip triangle

Examples (THREE js code: putting it all together)

```
...
scene.add(triangle);  // scene is of type THREE.Scene
```

Mesh Hacks

- A mesh object in three.js corresponds to the OpenGL primitive GL_TRIANGLE.
- The geometry object for a mesh must specify the triangles, in addition to the vertices.
- We will do the full version later.

Mesh Hack Cylinder

Definitions (THREE constructors)

.CylinderGeometry(radiusTop,radiusBottom,height, radiusSegments,heightSegments,openEnded, thetaStart,thetaLength) creates a cylinder mesh geometry object

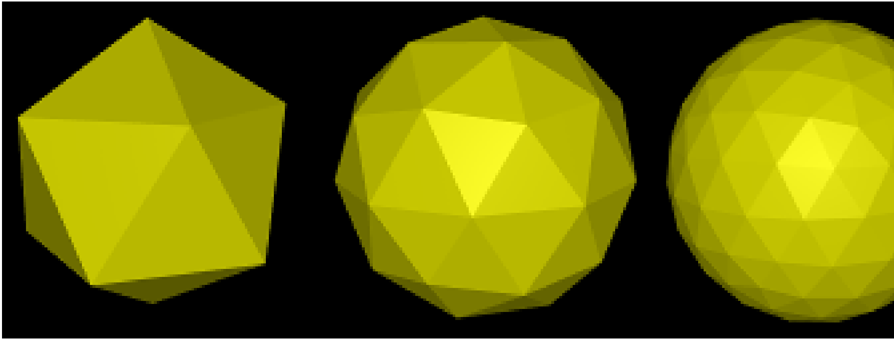
Some others

- .BoxGeometry creates the geometry of a rectangular box
- .PlaneGeometry creates a rectangle lying in the xy-plane
- .RingGeometry creates an annulus in the xy-plane
- .SphereGeometry creates a sphere with axis along the y-axis
- .TorusGeometry creates a torus lying in the xy-plane

Mesh Hack Cylinder detour.

Still others

- .TetrahedronGeometry, .OctahedronGeometry, .DodecahedronGeometry, and THREE.IcosahedronGeometry.



Mesh Hack Cylinder cont.

There are three types of mesh materials

Definitions (THREE contructors)

.MeshBasicMaterial(...) Colour not affected by lighting

.MeshLambertMaterial(...) uses Lambert shading for lighting

.MeshPhongMaterial(...) used Phong shading for lighting

Mesh Hack Cylinder detour. Shading

Definitions

Lambert shading A technique for computing pixel colors on a primitive using a lighting equation that takes into account ambient and diffuse reflection. In Lambert shading, the lighting equation is applied only at the vertices of the primitive. Color values for pixels in the primitive are calculated by interpolating the values that were computed for the vertices. Lambert shading is named after Johann Lambert, who developed the theory on which it is based in the eighteenth century.

Mesh Hack Cylinder detour. Shading cont.

Definitions

Phong shading A technique for computing pixel colors on a primitive using a lighting equation that takes into account ambient, diffuse, and specular reflection. In Phong shading, the lighting equation is applied at each pixel. Normal vectors are specified only at the vertices of the primitive. The normal vector that is used in the lighting equation at a pixel is obtained by interpolating the normal vectors for the vertices. Phong shading is named after Bui Tuong Phong, who developed the theory in the 1970s.

Mesh Hack Cylinder cont.

Examples (THREE js code: setting up a mesh Phong material)

```
var mat = new THREE.MeshPhongMaterial( {
          color: 0xbbbb00,      // reflectivity for diffuse light
          ambient: 0xbbbb00,    // reflectivity for ambient light
          emissive: 0,          // emission color; this is the default (black)
          specular: 0x070707,   // reflectivity for specular light
          shininess: 50,         // controls size of specular highlights

//Optional set to defaults here
                 wireframe: false,
                 wireframeLinewidth: 1,
                 visible: true,
                 side: THREE.FrontSide //.BackSide or .DoubleSide
                 shading: THREE.SmoothShading //.Flatshading
    } );
```

Mesh Hack Cylinder cont.

Examples (THREE js code: shiny, blue-green, open, five-sided tube)

```
var mat = new THREE.MeshPhongMaterial( {
        color: 0x0088aa,
        ambient: 0x0088aa,
        specular: 0x003344,
        shininess: 100,
        shading: THREE.FlatShading, // for flat-looking sides
        side: THREE.DoubleSide  // for drawing the inside of the tube
    } );
var geom = new THREE.CylinderGeometry(3,3,10,5,1,true);
var obj = new THREE.Mesh(geom,mat);
scene.add(obj);
```

Lets See That In Action

Demo

Three.js Mesh Object Viewer

Depth Test

Remember the depth test and polygon offset

Examples (THREE js code: making sure we can see the lines)

```
mat = new THREE.MeshLambertMaterial({
        polygonOffset: true,
        polygonOffsetUnits: 1,
        polygonOffsetFactor: 1,
        color: "yellow",
        ambient: "yellow",
        side: THREE.DoubleSide
   });
```

Lights

Introduction

A light object can be added to a scene and will then illuminate objects in the scene. We'll look at directional lights, point lights, ambient lights, and spotlights.

Lights

Definitions (THREE constructors)

.DirectionalLight(colour,intensity) colour is a colour object, intensity 0 to 1

.PointLight(colour,intensity,cutoff) cutoff 0 is to infinity else intesity decreases to zero by cutoff

.AmbientLight(colour)

.SpotLight(color,intensity,cutoff,coneAngle,exponent) a spotlight

Lights

Definitions

Attenuation Refers to the way that illumination from a point light or spot light decreases with distance from the light. Physically, illumination should decrease with the square of the distance, but computer graphics often uses a linear attenuation with distance, or no attenuation at at all.

Setting Up A Light

Examples (THREE js code: setting up some lights)

```
var light = new THREE.DirectionalLight(); // default white light
light.position.set( 0, 0, 1 );
scene.add(light);
var light = new THREE.PointLight( 0xffffcc , 1, 100 );
light.position.set( 10, 30, 15 );
scene.add(light);
scene.add( new THREE.AmbientLight(0x111100) );
spotlight = new THREE.SpotLight();
spotlight.position.set(0,0,5);
spotlight.target.position.set(2,2,0);
scene.add(spotlight);
scene.add(spotlight.target);
```

Spotlight And Materials

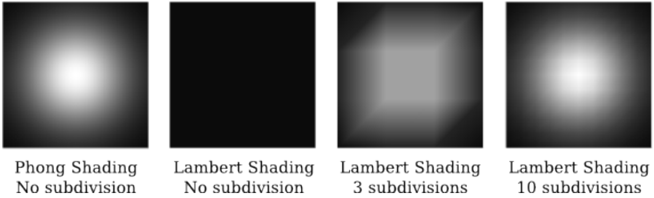


Figure: Spotlight on different materials

A Modelling Example

Introduction

You should be ready to go with your first three.js program at this point.

Lets See That In Action

Demo

threejs/diskworld-1.html

Outline

2. Building Objects
    - Indexed Face Sets
    - Curves and Surfaces
    - Textures
    - Transforms
    - Loading JSON Models

Building Objects

Introduction

In three.js, a visible object is constructed from a geometry and a material. We have seen how to create simple geometries that are suitable for point and line primitives, and we have encountered a variety of standard mesh geometries such as THREE.CylinderGeometry and THREE.IcosahedronGeometry. In this section, we will see how to create new mesh geometries from scratch We'll also look at some of the other support that three.js provides for working with objects and materials.

Indexed Face Sets

Introduction

A mesh in three.js is what we called an indexed face set.

A Pyramid

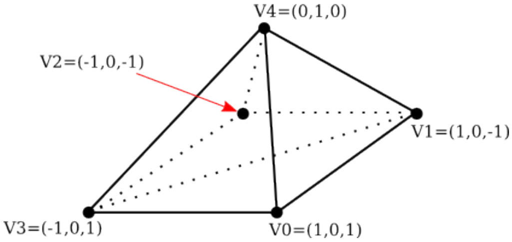


Figure: A Pyramid as a IFS

A Pyramid cont.

Examples (THREE js code: IFS for a pyramid)

```
var pyramidGeom = new THREE.Geometry();

pyramidGeom.vertices = [  // array of Vector3 giving vertex coordinates
        new THREE.Vector3( 1, 0, 1 ),    // vertex number 0
        new THREE.Vector3( 1, 0, -1 ),   // vertex number 1
        new THREE.Vector3( -1, 0, -1 ),  // vertex number 2
        new THREE.Vector3( -1, 0, 1 ),   // vertex number 3
        new THREE.Vector3( 0, 1, 0 )     // vertex number 4
    ];

pyramidGeom.faces = [  // array of Face3 giving the triangular faces
        new THREE.Face3( 3, 2, 1 ),  // one half of the bottom face
        new THREE.Face3  3, 1, 0 ),  // second half of the bottom face
        new THREE.Face3( 3, 0, 4 ),  // remaining faces are the four sides
        new THREE.Face3( 0, 1, 4 ),
        new THREE.Face3( 1, 2, 4 ),
        new THREE.Face3( 2, 3, 4 )
    ];
```

Calculate The Normals

Definitions (Geometry method)

.computeFaceNormals() calculates the normals for you, assumes flat shading

.computeVertexNormals() calculates smooth shading surface normals

.face.normal stores the flat shading normals

.face.vertexNormals stores the vertex normals

Examples (THREE js code: calculating the normals)

```
pyramidGeom.computeFaceNormals(); //must be done before calculating the vertex
pyramidGeom.computeVertexNormals();
```

A Material Per A Face

Definitions (THREE constructor)

.MeshFaceMaterial(...)

Examples (THREE js code: Setting up a mesh face)

```
var cubeGeom = new THREE.BoxGeometry(10,10,10);
var cubeMaterial = new THREE.MeshFaceMaterial( [
    new THREE.MeshPhongMaterial( { color: "red" } ),
    new THREE.MeshPhongMaterial( { color: "cyan" } ),
    new THREE.MeshPhongMaterial( { color: "green" } ),
    new THREE.MeshPhongMaterial( { color: "magenta" } ),
    new THREE.MeshPhongMaterial( { color: "blue" } ),
    new THREE.MeshPhongMaterial( { color: "yellow" } )
] );
var cube = new THREE.Mesh( cubeGeom, cubeMaterial );

pyramidGeom.faces[0].materialIndex = 0;
for (var i = 1; i <= 5; i++) {
    pyramidGeom.faces[i].materialIndex = i-1;
}
```

A Material Per A Face cont.

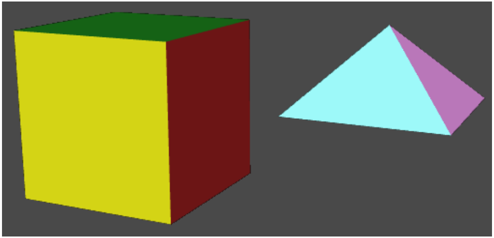


Figure: How that mesh face looks

Lets See That In Action

Demo

Animating Mesh Vertices and Colours

Curves and Surfaces

Introduction

In addition to letting you build indexed face sets, three.js has support for working with curves and surfaces that are defined mathematically.

A Parametric Surface

Definitions (THREE constructors)

.ParametricGeometry(func,slices,stacks) where func is the JavaScript function and slices and stacks determine the number of points in the grid

Examples (THREE js code: example surface)

```
function surfaceFunction( u, v ) {
    var x,y,z;  // A point on the surface, calculated from u,v.
                // u  and v range from 0 to 1.
    x = 20 * (u - 0.5);  // x and z range from -10 to 10
    z = 20 * (v - 0.5);
    y = 2*(Math.sin(x/2) * Math.cos(z));
    return new THREE.Vector3( x, y, z );
}

var surfaceGeometry = new THREE.ParametricGeometry(surfaceFunction, 64, 64);
var surface = new THREE.Mesh( surfaceGeometry, material );
```

A Parametric Surface cont.

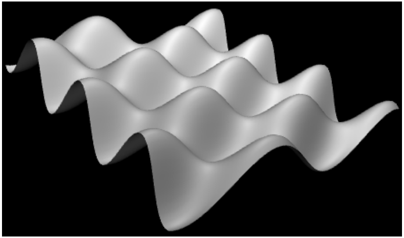


Figure: The parametric surface

A Tube Geometry Curve

Examples (THREE js code: A Tube Geometry Curve)

```
var helix = new THREE.Curve();
helix.getPoint = function(t) {
   var s = (t - 0.5) * 12*Math.PI;
         // As t ranges from 0 to 1, s ranges from -6*PI to 6*PI
   return new THREE.Vector3(
        5*Math.cos(s),
        s,
        5*Math.sin(s)
   );
}

tubeGeometry1 = new THREE.TubeGeometry( helix, 128, 2.5, 32 );
```

Tube Curve cont.

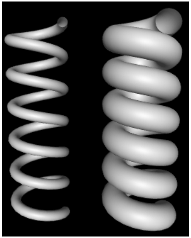


Figure: Tube Geometry

Lathe Curve

Definitions

lathing A technique for producing a surface by rotating a planar curve about a line that lies in the same plane as the curve. As each point rotates about the line, it generates a circle. The surface is the union of the circles generated by all the points on the curve. Lathing imitates shapes that can be produced by a mechanical lathe.

Examples (THREE js code:)

```
new THREE.LatheGeometry( points , slices )
```

Lathe Curve cont.

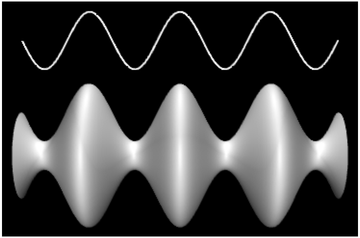


Figure: Lathe Geometry

Extrusion Curve

Definitions

extrusion A technique for producing a solid from a 2D shape by moving the shape along a curve in 3D. The solid is the set of points through which the shape passes as it moves along the curve. The most common case is moving the shape along a line segment that is perpendicular to the plane that contains the shape. In practice, in computer graphics, the object that is produced by extrusion is just the surface of the extruded solid.

Extrusion Curve cont.

Examples (THREE js code: shape of an extrusion curve)

```
var path = new THREE.Shape();
path.moveTo(0,10);
path.bezierCurveTo( 0,5, 20,-10, 0,-10 );
path.bezierCurveTo( -20,-10, 0,5, 0,10 );

var shapeGeom = new THREE.ShapeGeometry( path, {
        curveSegments: 32
    });
```

Extrusion Curve cont.

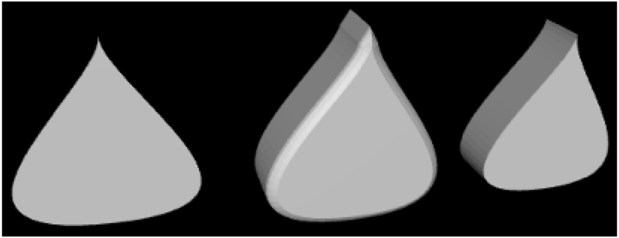


Figure: Extrusion Geometry

Lets See That In Action

Demo

sample program (http://math.hws.edu/graphicsbook/source/threejs/curves-and-surfaces.html)

Textures

Introduction

A texture can be used to add visual interest and detail to an object. In three.js, an image texture is represented by an object of type THREE.Texture.

Textures

Definitions (THREE constructors)

Texture(image,mapping,wrapS,wrapT,magFilter,minFilter,format,type,anis

Create a texture to apply to a surface or as a reflection or refraction map.

Definitions (THREE.ImageUtils methods)

.loadTexture(imageURL,mapping,onLoad,onError) loads a texture asynchronously, calls function onLoad on success.

Textures cont.

Examples (THREE js code: loading a texture)

```
var texture = THREE.ImageUtils.loadTexture( "brick.png", undefined, render );
...
material.map = texture;
material.needsUpdate = true;
```

Definitions (THREE.Texture properties)

.wrapS= THREE.(RepeatWrapping,MirroredRepeatWrapping)

.wrapT= THREE.(RepeatWrapping,MirroredRepeatWrapping)

.offset.(x,y) translation of the texture

.repeat(x,y) effectively does scaling with repeat wrapping

Warning

Warning

Remember that a positive horizontal offset will move the texture to the left on the objects, because the offset is applied to the texture coordinates not to the texture image itself.

Lets See That In Action

Demo

Three.js Textures

A Pyramid Texture Example

Figure: Textured Pyramid

Pyramid cont.

Definitions (THREE.Geometry properties)

.faceVertexUvs an array that maps texture S,T coordinates onto the face U,V coordinates

Examples (THREE js code: mapping a texture coordinates to a face coordinates)

```
pyramidGeometry.faceVertexUvs = [[
   [ new THREE.Vector2(0,0), new THREE.Vector2(0,1), new THREE.Vector2(1,1) ],
   [ new THREE.Vector2(0,0), new THREE.Vector2(1,1), new THREE.Vector2(1,0) ],
   [ new THREE.Vector2(0,0), new THREE.Vector2(1,0), new THREE.Vector2(0.5,1) ],
   [ new THREE.Vector2(1,0), new THREE.Vector2(0,0), new THREE.Vector2(0.5,1) ],
   [ new THREE.Vector2(0,0), new THREE.Vector2(1,0), new THREE.Vector2(0.5,1) ],
   [ new THREE.Vector2(1,0), new THREE.Vector2(0,0), new THREE.Vector2(0.5,1) ],
]];
```

Lets See That In Action

Demo

threejs/textured-pyramid.html

Transforms

Introduction

In order to understand how to work with objects effectively in three.js, it can be useful to know more about how it implements transforms.

Object3D Transforms

Definitions (THREE.Object3D methods)

- **.matrix** represents the object transformation as a matrix
- **.matrixAutoUpdate** controls whether .matrix is computed automatically
- **.updateMatrix()** compute the matrix from the current values of .position, .scale, and .rotation

Object3D Transforms cont.

Definitions (THREE.Object3D methods)

.translateX(dx), move the object by a specified amount in the X direction

.translateY(dy), move the object by a specified amount in the Y direction

.translateZ(dz) move the object by a specified amount in the Z direction

Object3D Transforms cont.

Definitions (THREE.Object3D methods)

.rotateX(angle), rotate the object about the X-coordinate axes

.rotateY(angle), rotate the object about the Y-coordinate axes

.rotateZ(angle) rotate the object about the Z-coordinate axes

Object3D Transforms cont.

Definitions (THREE.Object3D methods)

.rotateOnAxis(axis,angle) rotates the object through the angle about the axis

.lookAt(vec) which rotates the object so that it is facing towards a given point

.up the up directioon of an object default $(0, 1, 0)$

Warning

Warning

Translation and rotation functions modify the position and rotation properties of the object. That is, they apply in object coordinates, not world coordinates, and they are applied when the object is rendered, in the order scale, then rotate, then translate.

Loading JSON Models

Introduction

Although it is possible to create mesh objects by listing their vertices and faces, it would be difficult to do it by hand for all but very simple objects. It's much easier, for example, to design an object in an interactive modeling program such as Blender and then import it.

JSON

Definitions (THREE.JSONLoader methods)

.load(url,callback) asynchronously load the JSON object from url

Examples (THREE js code: Loading a model)

```
function loadModel( url ) {  // Call this function to load the model.
    var loader = new THREE.JSONLoader();
    loader.load( url, modelLoaded ); // Start load, call modelLoaded when done.
}

function modelLoaded( geometry, materials ) { // callback function for loader
    var mat = new THREE.MeshFaceMaterial(materials);
    var object = new THREE.Mesh( geometry, mat );
    scene.add(object);
    render(); // (only need this if there is no animation running)
}
```

Lets See That In Action

Demo

threejs/json-model-viewer.html

Demo

Three.js Mesh Animation

Outline

Other Features

Introduction

Some other cool things in three.js and some extra theory that goes with it.

Anaglyph Stereo

Introduction

Three.js has support for Anaglyph stereo.

User Input

Introduction

Most real programs require some kind of user interaction. For a web application, of course, the program can get user input using HTML widgets such as buttons and text input boxes. But direct mouse interaction with a 3D world is more natural in many programs.

Resources

- You need two classes that are not part of the main three.js file
  - OrbitControls.js
  - TrackballControls.js

Using OrbitControls

Examples (THREE js code: using OrbitControls)

```
camera = new THREE.PerspectiveCamera(45, canvas.width/canvas.height, 0.1, 100);
camera.position.set(0,15,35);
camera.lookAt( new THREE.Vector3(0,0,0) ); // camera looks toward origin

var light = new THREE.PointLight(0xffffff, 0.7);
camera.add(light);  // viewpoint light moves with camera
scene.add(camera);

controls = new THREE.OrbitControls( camera, canvas );
```

Examples (THREE js code: updating)

```
...
controls.update(); //must call whenever the mouse is dragged
render()
...
```

Selecting With A Raycaster

Definitions (THREE.Raycaster methods)

.set(startingPoint,direction) two Vector3 parameters starting point and direction as if a gun must be normalized

.setFromCamera(screenCoords,camera) screenCoords Vector2 in clip coordinates from camera

.intersectsObjects(objectArray,recursive) recursively search scenegraph starting with objects in objectArray

Examples (THREE js code: THREE.Raycaster)

```
raycaster = new THREE.Raycaster();
raycaster.set( startingPoint , direction ); //ray to use
```

Calculating From Canvas Coordinates To Clip Coordinates

Examples (THREE js code: Setting up a raycaster)

```
var r = canvas.getBoundingClientRect();
var x = evt.clientX - r.left; // convert mouse location to canvas pixel coords
var y = evt.clientY - r.top;

var a = 2*x/canvas.width - 1; // convert canvas pixel coords to clip coords
var b = 1 - 2*y/canvas.height;

raycaster.setFromCamera( new THREE.Vector2(a,b), camera );
```

Examples (THREE js code: finding objects that intersect the ray)

```
objects = raycaster.intersectsObjects( scene.children, true );
objects[0].object //the first intersected object
objects[0].point //point of intersection as a Vector3
```

Lets See That In Action

Demo

Using a Raycaster for Input

Some Code

Examples (THREE js code: the example)

```
...
if ( intersects[0].object != ground ) {
    world.remove( intersects[0].object );
    render();
}
...
item = intersects[0];
if (item.object == ground) {
    var locationX = item.point.x;  // world coords of intersection point
    var locationZ = item.point.z;
    var coords = new THREE.Vector3(locationX, 0, locationZ); // y is always 0
    world.worldToLocal(coords); // transform to local coords
    addCylinder(coords.x, coords.z); // adds a cylinder at corrected location
    render();
}
```

Shadows

Introduction

One thing that has been missing in our 3D images is shadows. Even if you didn't notice the lack consciously, it made many of the images look wrong. Shadows can add a nice touch of realism to a scene, but OpenGL, including WebGL, cannot generate shadows automatically.

Shadows

Definitions

Shadow mapping A technique for determining which parts of a scene are illuminated and which are in shadow from a given light source. The technique involves rendering the scene from the point of the view of the light source, but uses only the depth buffer from that rendering. The depth buffer is the "shadow map." Along a given direction from the light source, the object that is illuminated by the light is the one that is closest to the light. The distance to that object is essentially encoded in the depth buffer. Objects at greater distance are in shadow.

Lets See That In Action

Demo

Three.js Shadow Demo

Cubemap Textures and Skyboxes

Introduction

It would be nice to put our scenes in an "environment" such as the interior of a building, a nature scene, or a public square. It's not practical to build representations of such complex environments out of geometric primitives, but we can get a reasonably good effect using textures.

Skyboxes

Definitions

Skybox large cube that surrounds a scene and is textured with images that form a background for that scene, in all directions.

Cubemap Texture A texture made up of six images, one for each of the directions positive x, negative x, positive y, negative y, positive z, and negative z. The images are intended to include everything that can be seen from a given point. Cubemap textures are used for environment mapping and skyboxes.

CubeMap Textures

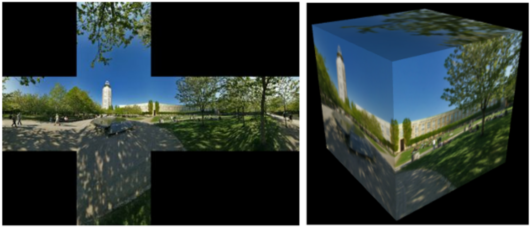


Figure: Emil Persson, has made a large number of cube maps available for download at `http://www.humus.name/index.php?page=Textures` under a creative commons license.

Getting Cubemap In Code

Examples (THREE js code: setting up a cubemap)

```
var textureURLs = [  // URLs of the six faces of the cube map
        "cubemap-textures/park/posx.jpg",   // Note:  The order in which
        "cubemap-textures/park/negx.jpg",   //
the images are listed is
        "cubemap-textures/park/posy.jpg",   //   important!
        "cubemap-textures/park/negy.jpg",
        "cubemap-textures/park/posz.jpg",
        "cubemap-textures/park/negz.jpg"
   ];

var materials = [];
for (var i = 0; i < 6; i++) {
    var texture = THREE.ImageUtils.loadTexture( textureURLs[i] );
    materials.push( new THREE.MeshBasicMaterial( {
        color: "white",  // Color will be multiplied by texture color.
        side: THREE.BackSide,  // IMPORTANT: To see the inside of the cube,
                               //
back faces must be rendered!
        map: texture
    } ) );

}

cube = new THREE.Mesh( new THREE.CubeGeometry(100,100,100),
```

Achieving It With A Shader Material

Examples (THREE js code: setting up a cubemap)

```
var texture = THREE.ImageUtils.loadTextureCube( textureURLs );

var shader = THREE.ShaderLib[ "cube" ]; // contains the required shaders
shader.uniforms[ "tCube" ].value = texture; // data for the shaders
var material = new THREE.ShaderMaterial( {
        // A ShaderMaterial uses custom vertex and fragment shaders.
    fragmentShader: shader.fragmentShader,
    vertexShader: shader.vertexShader,
    uniforms: shader.uniforms,
    depthWrite: false,
    side: THREE.BackSide
} );

cube = new THREE.Mesh( new THREE.CubeGeometry( 100, 100, 100 ), material );
```

Lets See That In Action

Demo

threejs/skybox.html

Reflection and Refraction

Introduction

A reflective surface shouldn't just reflect light—it should reflect its environment.

Reflection Through Environmental Mapping

Definitions

Environment mapping A way of simulating mirror-like reflection from the surface of an object. The environment that is to be reflected from the surface is represented as a cubemap texture. To determine what point in the texture is visible at a given point on the object, a ray from the viewpoint is reflected from the surface point, and the reflected ray is intersected with the texture cube. Environment mapping is also called reflection mapping.

Simple Enough In three.js

Examples (THREE js code: using a cubmap as a reflection)

```
var geometry = new THREE.SphereGeometry(1,32,16);
var material = new THREE.MeshBasicMaterial( {
      color: "white",  // Color will be multiplied by the environment map.
      envMap: texture  // Cubemap texture to be used as an environment map.
   } );
var mirrorSphere = new THREE.Mesh( geometry, material );
```

And In Pictures

Figure: Cubemap as a reflection

Lets See That In Action

Demo

threejs/reflection.html

Demo

Skybox and Reflection Mapping

Refraction

Definitions

Refraction The bending of light as it passes from one transparent or translucent medium into another.

Examples (THREE js code: loading a refraction object)

```
//Note the last parameter
texture = THREE.ImageUtils.loadTextureCube(
                        textureURLs, THREE.CubeRefractionMapping );


var material = new THREE.MeshBasicMaterial( {
        color: "white",
        envMap: texture,
        refractionRatio: 0.6
    } );
```

And In Pictures

Figure: Cubemap as a refraction

Lets See That In Action

Demo

threejs/refraction.html

The Cube Camera

Note

This all works well if you have a cube box, but what happens if you want to see the things in your scene? Then what you need is a THREE.CubeCamera.

David J. Eck; Introduction to Computer Graphics; 2016; http://math.hws.edu/graphicsbook/