Lab 02: What Can We Do With THREE.js

CS423: Computer Graphics

Contents

1 Overview

This lab is an illustration of what we can do with `THREE.js`. There are some things in this lab that anticipate future lectures so roll with me when we hit those parts of lab. The purpose is twofold: give you idea of what we can do with the library and give a bit of an overview of the underlying workflow.

2 Instructions

Starting from our WebGL HTML template from Lab 01, create a new file named `01-basic-scene.html` with the following HTML:

```
<!DOCTYPE html>
<HTML>
<HEAD>
    <TITLE>Example 02.01 - Basic Scene</TITLE>
    <SCRIPT TYPE="text/javascript" SRC="../libs/three.js"></SCRIPT>

    <SCRIPT TYPE="text/javascript" SRC="../libs/stats.min.js"></SCRIPT>
    <SCRIPT type="text/javascript" src="../libs/dat.gui.min.js"></SCRIPT>
    <STYLE>
        body {
            /* set margin to 0 and overflow to hidden, to go fullscreen */
            margin: 0;
            overflow: hidden;
        }
    </STYLE>
</HEAD>
<BODY>

<DIV id="Stats-output">
</DIV>
<!-- Div which will hold the Output -->
<DIV id="WebGL-output">
</DIV>
<!-- Javascript code that runs our Three.js examples -->
<SCRIPT TYPE="text/javascript" SRC="01-basic-scene.js">
</SCRIPT>
</BODY>
</HTML>
```

Let's look at the additions. First, we are using two more libraries. The `stats` library is used to gather statistics about the performance of application. We will be using this to get data about the number of frames per second generated by the WebGL renderer. The second library is going to be used to add some user interface chrome to our webpage.

Note also that we're going to be using a separate file for our Javascript code. There are differing schools of thought about this coding pattern. Some people prefer to do everything buried in script tags in your HTML so that you're managing one file. But others, myself included, will use a separate file for my Javascript to try to keep things better organized.

Let's start building that Javacript file. Create a new file named `01-basic-scene.js` and add the following:

```
//
// File:   01-basic-scene.js
// Author: adam.lewis@athens.edu
// Purpose:
// Demo some of the basics of working with the scenegraph.
// This is an extension of code from the Learning THREE.js textbook
// once everything is loaded, we run our Three.js stuff.
function init() {

    var stats = initStats();

}
window.onload = init;
```

This gets things started as we did previously by defining a function `init()` and then setting the window `onload` handler to that function. We'll do all of the work within that function. Note also that we will implementing everything as inner functions with `init()`.

To that end, let's get the stuff in place we need to get the stats library up and running:

```
function init() {

    var stats = initStats();

    function initStats() {

        var stats = new Stats();

        stats.setMode(0); // 0: fps, 1: ms

        // Align top-left
        stats.domElement.style.position = 'absolute';
        stats.domElement.style.left = '0px';
        stats.domElement.style.top = '0px';

        document.getElementById("Stats-output").appendChild(stats.domElement);

        return stats;
    }
}
```

A few points about this code:

- Tied to Javascript is the `Domain Object Model` (DOM) concept where your browser keeps a tree structure that represents the structure of the web page that is being rendered
  - In this case, we are going to associating the DOM entry for the stats object as a child of the `Stats-ouput` division of our web page.
- In addition we are setting some CSS properties of this element.

- And remember... inner functions.

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

    // create a camera, which defines where we're looking at.
    var camera = new THREE.PerspectiveCamera(45,
                         window.innerWidth /window.innerHeight,
                         0.1,
                         1000);
    scene.add(camera);

    // create a render and set the size
    var renderer = new THREE.WebGLRenderer();

    renderer.setClearColor(new THREE.Color(0xEEEEEE, 1.0));
    renderer.setSize(window.innerWidth, window.innerHeight);
    renderer.shadowMap.enabled = true;
```

There's a bunch of `THREE.js` stuff here but roll with me as we'll explain the details over the next few class sessions. In this case, you see the things you have to do in every `THREE.js`-based `WebGL` application to get things started: (1) create a scene, create a camera, and then build a renderer to draw the stuff we want to draw.

Stuff, you say? What sort of stuff. Well, let's add a ground plane with some lighting effects:

```
    // create the ground plane
    var planeGeometry = new THREE.PlaneGeometry(60, 40, 1, 1);
    var planeMaterial = new THREE.MeshLambertMaterial({color: 0xffffff});
    var plane = new THREE.Mesh(planeGeometry, planeMaterial);
    plane.receiveShadow = true;

    // rotate and position the plane
    plane.rotation.x = -0.5 * Math.PI;
    plane.position.x = 0;
    plane.position.y = 0;
    plane.position.z = 0;

    // add the plane to the scene
    scene.add(plane);

    // position and point the camera to the center of the scene
    camera.position.x = -30;
    camera.position.y = 40;
    camera.position.z = 30;
    camera.lookAt(scene.position);

    // add subtle ambient lighting
    var ambientLight = new THREE.AmbientLight(0x0c0c0c);
    scene.add(ambientLight);

    // add spotlight for the shadows
    var spotLight = new THREE.SpotLight(0xffffff);
    spotLight.position.set(-40, 60, -10);
    spotLight.castShadow = true;
    scene.add(spotLight);

    // add the output of the renderer to the html element
    document.getElementById("WebGL-output").appendChild(renderer.domElement);
```

Again, lot's of stuff that makes you say "Wut?". But the workflow is more important here: (1) create and configure a drawing object, (2) position that drawing object in 3d space, and (3) add it to the scene to be rendered. Note how we connect the `WebGL` renderer to the matching HTML element in the DOM.

Now let's add the user interface controls that we need to add to make our script do something interesting:

```
var controls = new function () {
    this.rotationSpeed = 0.02;
    this.numberOfObjects = scene.children.length;

    this.removeCube = function () {
        var allChildren = scene.children;
        var lastObject = allChildren[allChildren.length - 1];
        if (lastObject instanceof THREE.Mesh) {
            scene.remove(lastObject);
            this.numberOfObjects = scene.children.length;
        }
    };

    this.addCube = function () {

        var cubeSize = Math.ceil((Math.random() * 3));
        var cubeGeometry = new THREE.BoxGeometry(cubeSize, cubeSize, cubeSize);
        var cubeMaterial = new THREE.MeshLambertMaterial({color: Math.random() * 0xffffff});
        var cube = new THREE.Mesh(cubeGeometry, cubeMaterial);
        cube.castShadow = true;
        cube.name = "cube-" + scene.children.length;
```

```
23
            // position the cube randomly in the scene
25
            cube.position.x = -30 + Math.round((Math.random() * planeGeometry.parameters.width));
27          cube.position.y = Math.round((Math.random() * 5));
            cube.position.z = -20 + Math.round((Math.random() * planeGeometry.parameters.height));
29
            // add the cube to the scene
31          scene.add(cube);
            this.numberOfObjects = scene.children.length;
33      };

35      this.outputObjects = function () {
            console.log(scene.children);
37      }
    };
39  var gui = new dat.GUI();

41  gui.add(controls, 'rotationSpeed', 0, 0.5);
    gui.add(controls, 'addCube');
43  gui.add(controls, 'removeCube');
    gui.add(controls, 'outputObjects');
45  gui.add(controls, 'numberOfObjects').listen();
```

This creates a set of user interface controls that can be used to add cube shaped objects onto the scene. There is code is used Javascript anonymous functions (lambda functions) to set event handlers for the controls. But the body of those functions are following the same pattern as the previous code: (1) create and configure an object, and (2) add it to the scene to be rendered.

Now can do the rendering:

```
function render() {
2   stats.update();

4   // rotate the cubes around its axes
    scene.traverse(function (e) {
6       if (e instanceof THREE.Mesh && e != plane) {

8           e.rotation.x += controls.rotationSpeed;
            e.rotation.y += controls.rotationSpeed;
10          e.rotation.z += controls.rotationSpeed;
        }
12  });

14  // render using requestAnimationFrame
    requestAnimationFrame(render);
16  renderer.render(scene, camera);
}
18
render();
```

Save both the HTML and JS files and load them into your web browser.

3 Submission instructions

Please create a PDF file with the following:

- A screen-shot of both your webapps displayed in the browser.
- HTML and JS files for each webapp

Attach this PDF file to the submission link in Blackboard.