3D board game in a browser using WebGL and Three.js, part 1

[0 comm.](https://www.osd.net/blog/web-development/3d-board-game-in-a-browser-using-webgl-and-three-js-part-1/#respond)

posted by [CAM](https://www.osd.net/blog/author/adrian/), April 29, 2013  
Categories: [JavaScript](https://www.osd.net/blog/category/web-development/javascript/), [Web Development](https://www.osd.net/blog/category/web-development/)  
Taggs [3D](https://www.osd.net/blog/tag/3d/), [three.js](https://www.osd.net/blog/tag/three-js/), [WebGL](https://www.osd.net/blog/tag/webgl/)

This is the first part from a series of articles that will show you how to build a 3D board game that will run in a modern browser.

Checkers is the game that will be used to demonstrate the browser’s 3D capabilities. To find out more about the game go [here](http://en.wikipedia.org/wiki/English_draughts).

Some notes before starting:

* This and the following articles will not show you how to build a complete checkers game, but instead will show you only the basics because the focus will be on the actual 3D rendering.
* The 3D assets needed will be provided to you and there is no info on how to create them other than some basic guidelines specific to our goal.

What you’ll learn in this part

This article will cover the basic setup of the game, like directory structure and needed files. In the end you’ll have a wireframe cube and you’ll be able to orbit around it. You can scroll down at the end of the article if you want to see it in all its glory.

Now let’s clarify some things.

What is WebGL?

Let’s see what Wikipedia has to say:

**WebGL** (Web Graphics Library) is a **JavaScript API** for rendering interactive **3D graphics** and **2D graphics** within any compatible web browser without the use of plug-ins. WebGL is integrated completely into all the web standards of the browser allowing **GPU accelerated** usage of physics and image processing and effects as part of the web page canvas.

That pretty much says it all. To find out more you can read the full description [here](http://en.wikipedia.org/wiki/WebGL).

To test if your browser and graphics card supports WebGL go to [get.webgl.org](http://get.webgl.org/). It should tell you that *“Your browser supports WebGL”* and you should see a spinning cube.

The only browsers that currently support WebGL out of the box are Mozilla Firefox and Google Chrome. Safari and Opera have implemented support for WebGL also, but it’s disabled by default.

What is Three.js?

[Three.js](http://threejs.org/) is a **JavaScript 3D library** that takes away the complexity of WebGL low-level API. Here’s an official description:

The aim of the project is to create a **lightweight 3D library** with a very low level of complexity — in other words, for dummies. The library provides <canvas>, <svg>, CSS3D and **WebGL renderers**.

We will be using this library to build our 3D board game.

Before we begin

You should be familiar with the following concepts in JavaScript:

[Object-Oriented JavaScript](https://developer.mozilla.org/en-US/docs/JavaScript/Introduction_to_Object-Oriented_JavaScript)

[Self-Invoking Anonymous Function](http://sarfraznawaz.wordpress.com/2012/01/26/javascript-self-invoking-functions/)

[Private Members in JavaScript](http://javascript.crockford.com/private.html)

[Strict Mode](http://www.yuiblog.com/blog/2010/12/14/strict-mode-is-coming-to-town/)

You may also want to read about **ECMAScript 5** new features:

[ECMAScript 5 Objects and Properties](http://ejohn.org/blog/ecmascript-5-objects-and-properties/)

[ECMAScript 5 Strict Mode, JSON, and More](http://ejohn.org/blog/ecmascript-5-strict-mode-json-and-more/)

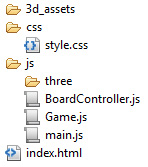
About testing

Because the 3D assets will be loaded via AJAX, opening the index.html file in the browser will only work in Firefox.

In Google Chrome you’ll see the following error message in Developer tools console: *“Cross origin requests are only supported for HTTP”.*To fix that you either use a local web server to access the project or you open Google Chrome with the --disable-web-security argument ([Disable same origin policy](http://stackoverflow.com/questions/3102819/chrome-disable-same-origin-policy)).

The beginining

Create a directory for our game and make it to reassemble the following structure:

[](http://www.osd.net/blog/wp-content/uploads/2013/04/webgl_3d_checkers_directory_structure.jpg)

**3d\_assets/:** the directory where the 3D content files will resides.

**css/style.css**: some basic CSS styles.

|  |  |
| --- | --- |
| 1 | html, body { |
| 2 | height: 100%; | |

|  |  |  |
| --- | --- | --- |
| 3 | } | |
| 4 |  |

|  |  |
| --- | --- |
| 5 | body { |
| 6 | margin: 0; | |

|  |  |  |
| --- | --- | --- |
| 7 | overflow: hidden; | |
| 8 | } |

|  |  |  |
| --- | --- | --- |
| 9 |  | |
| 10 | | #boardContainer { | |

|  |  |
| --- | --- |
| 11 | width: 100%; |
| 12 | height: 100%; | |

|  |  |  |
| --- | --- | --- |
| 13 | background: #e6e0cf; | |
| 14 | } |

**js/three/:** the three.js library home.

**js/main.js:** the checkers game will start from here.

**js/Game.js:** it will handle game logic and it will create a BoardController.

**js/BoardController.js:** it will handle all the actual drawing and mouse interaction.

**index.html:**

|  |  |  |
| --- | --- | --- |
| 1 | <!DOCTYPE html> | |
| 2 | <html> |

|  |  |
| --- | --- |
| 3 | <head> |
| 4 | <link rel="stylesheet" type="text/css" href="css/style.css" /> | |

|  |  |  |
| --- | --- | --- |
| 5 | </head> | |
| 6 | <body> |

|  |  |  |
| --- | --- | --- |
| 7 | <div id="boardContainer"></div> | |
| 8 |  |

|  |  |  |
| --- | --- | --- |
| 9 | <script src="js/three/three.js"></script> | |
| 10 | | <script src="js/three/OrbitControls.js"></script> | |

|  |  |
| --- | --- |
| 11 | <script src="js/Game.js"></script> |
| 12 | <script src="js/BoardController.js"></script> | |

|  |  |  |
| --- | --- | --- |
| 13 | <script src="js/main.js"></script> | |
| 14 | </body> |

|  |  |
| --- | --- |
| 15 | </html> |

[Download three.js](http://github.com/mrdoob/three.js/zipball/master) library and copy into **js/three/** the following files: **build/three.js** and **examples/js/controls/OrbitControls.js**.

Starting the game

Our game will be started from **main.js**:

|  |  |
| --- | --- |
| 1 | (function () { |
| 2 | 'use strict'; | |

|  |  |
| --- | --- |
| 3 |  |
| 4 | var game = new CHECKERS.Game({ | |

|  |  |
| --- | --- |
| 5 | // The DOM element in which the drawing will happen. |
| 6 | containerEl: document.getElementById('boardContainer'), | |

|  |  |
| --- | --- |
| 7 |  |
| 8 | // The base URL from where the BoardController will load its data. | |

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | assetsUrl: '3d\_assets/' | | |
| 10 | | }); |

|  |  |
| --- | --- |
| 11 |  |
| 12 | })(); | |

The *Game* object will be placed in the *CHECKERS* namespace and it will automatically start a new game on instantiation. The properties passed to the configuration object will be passed to *BoardController*‘s instantiation also.

Let’s have a look at **Game.js**:

|  |  |  |
| --- | --- | --- |
| 1 | var CHECKERS = {}; | |
| 2 |  |

|  |  |  |
| --- | --- | --- |
| 3 | CHECKERS.Game = function (options) { | |
| 4 | 'use strict'; |

|  |  |
| --- | --- |
| 5 |  |
| 6 | options = options || {}; | |

|  |  |
| --- | --- |
| 7 |  |
| 8 | var boardController = null; | |

|  |  |  |
| --- | --- | --- |
| 9 |  | |
| 10 | | function init() { | |

|  |  |  |
| --- | --- | --- |
| 11 | boardController = new CHECKERS.BoardController({ | |
| 12 | containerEl: options.containerEl, |

|  |  |  |
| --- | --- | --- |
| 13 | assetsUrl: options.assetsUrl | |
| 14 | }); |

|  |  |
| --- | --- |
| 15 |  |
| 16 | boardController.drawBoard(); | |

|  |  |  |
| --- | --- | --- |
| 17 | } | |
| 18 |  |

|  |  |  |
| --- | --- | --- |
| 19 | init(); | |
| 20 | }; |

On line 8 we define a private property that will holds the *BoardController* instance. On line 10 the *init* function is defined and on line 19 the same function is called.

**Note:** The *init* function should be kept always at the bottom.

The job of the *init* function is to create an instance of *BoardController* and to request the board drawing at line 16.

Let’s move on to **BoardController.js**:

|  |  |  |
| --- | --- | --- |
| 1 | CHECKERS.BoardController = function (options) { | |
| 2 | 'use strict'; |

|  |  |
| --- | --- |
| 3 |  |
| 4 | options = options || {}; | |

|  |  |
| --- | --- |
| 5 |  |
| 6 | var containerEl = options.containerEl || null; | |

|  |  |
| --- | --- |
| 7 |  |
| 8 | var assetsUrl = options.assetsUrl || ''; | |

|  |  |  |
| --- | --- | --- |
| 9 |  | |
| 10 | | this.drawBoard = function () { | |

|  |  |  |
| --- | --- | --- |
| 11 | console.log('drawBoard'); | |
| 12 | }; |

|  |  |
| --- | --- |
| 13 | }; |

Load/reload the *index.html* file in the browser and you should see the word ‘drawBoard’ printed in the browser’s console.

Creating the 3D virtual space

*The rest of the article will add code to BoardController.js file.*

Before drawing some actual 3D elements we need to create the WebGL renderer, a scene that will hold the 3D objects, a camera and a camera controller that will allow the mouse to control the camera position.

First let’s declare the needed variables:

|  |  |  |
| --- | --- | --- |
| 1 | assetsUrl: options.assetsUrl || ''; | |
| 2 |  |

|  |  |  |
| --- | --- | --- |
| 3 | var renderer; | |
| 4 | var scene; |

|  |  |
| --- | --- |
| 5 | var camera; |
| 6 | var cameraController; | |

We will initialize the variables in a function named **initEngine** called from *drawBoard*:

|  |  |  |
| --- | --- | --- |
| 1 | this.drawBoard = function () { | |
| 2 | initEngine(); |

|  |  |  |
| --- | --- | --- |
| 3 | }; | |
| 4 |  |

|  |  |
| --- | --- |
| 5 | function initEngine() { |
| 6 | var viewWidth = containerEl.offsetWidth; | |

|  |  |  |
| --- | --- | --- |
| 7 | var viewHeight = containerEl.offsetHeight; | |
| 8 |  |

|  |  |  |
| --- | --- | --- |
| 9 | // instantiate the WebGL Renderer | |
| 10 | | renderer = new THREE.WebGLRenderer({ | |

|  |  |  |
| --- | --- | --- |
| 11 | antialias: true | |
| 12 | }); |

|  |  |  |
| --- | --- | --- |
| 13 | renderer.setSize(viewWidth, viewHeight); | |
| 14 |  |

|  |  |
| --- | --- |
| 15 | // create the scene |
| 16 | scene = new THREE.Scene(); | |

|  |  |
| --- | --- |
| 17 |  |
| 18 | // create camera | |

|  |  |  |
| --- | --- | --- |
| 19 | camera = new THREE.PerspectiveCamera(35, viewWidth / viewHeight, 1, 1000); | |
| 20 | camera.position.set(0, 120, 150); |

|  |  |  |
| --- | --- | --- |
| 21 | cameraController = new THREE.OrbitControls(camera, containerEl); | |
| 22 | // |

|  |  |  |
| --- | --- | --- |
| 23 | scene.add(camera); | |
| 24 |  |

|  |  |  |
| --- | --- | --- |
| 25 | containerEl.appendChild(renderer.domElement); | |
| 26 | } |

The code should be pretty self-explanatory. To see what all those arguments passed to the *PerspectiveCamera* creation on line 19 means, have a look at the [documentation](http://threejs.org/docs/58/#Reference/Cameras/PerspectiveCamera).

All objects added to the scene will be positioned in the center, so on line 20 we move the camera up and towards us. On line 21 the camera controller is created by passing to the constructor the camera and the element that will listen for mouse events.

Drawing 3D objects

With the virtual world created now we are ready to add some 3D objects in it. That will be accomplished in a **initObjects** function added below the *initEngine*:

|  |  |
| --- | --- |
| 1 | function initObjects(callback) { |
| 2 | var cube = new THREE.Mesh(new THREE.CubeGeometry(50, 50, 50)); | |

|  |  |  |
| --- | --- | --- |
| 3 | scene.add(cube); | |
| 4 |  |

|  |  |  |
| --- | --- | --- |
| 5 | callback(); | |
| 6 | } |

The first thing to notice is that the function will accept a callback that is called at the end. For now only a cube is created and added to the scene on  line 2 and 3, but later on the checkers board and the pieces will be loaded via AJAX and we’ll want to know when everything have been loaded.

The *initObjects* function should be be called from *drawBoard* after *initEngine* like this:

|  |  |  |
| --- | --- | --- |
| 1 | initObjects(function () { | |
| 2 | onAnimationFrame(); |

|  |  |
| --- | --- |
| 3 | }); |

After the objects have been loaded we need to tell the WebGL renderer to render the scene on each animation frame.

**The render loop.**

After the *initObjects* function add the following:

|  |  |
| --- | --- |
| 1 | function onAnimationFrame() { |
| 2 | requestAnimationFrame(onAnimationFrame); | |

|  |  |
| --- | --- |
| 3 |  |
| 4 | cameraController.update(); | |

|  |  |
| --- | --- |
| 5 |  |
| 6 | renderer.render(scene, camera); | |

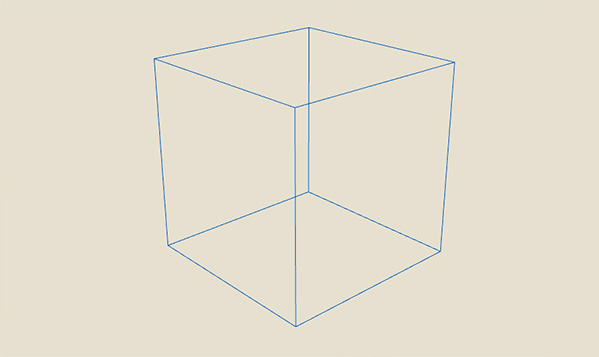
|  |  |
| --- | --- |
| 7 | } |

On line 2 an animation frame is requested ([info about requestAnimationFrame](https://developer.mozilla.org/en-US/docs/DOM/window.requestAnimationFrame)).

The camera controller needs to be updated on each frame, so we do that on line 4.

On line 6 the scene is getting rendered.

Now if you test in the browser you should see a wireframe cube:

[](http://www.osd.net/blog/wp-content/uploads/2013/04/webgl_3d_checkers_testing_cube.jpg)

Your cube may look different since the colors of the lines are changing randomly with each refresh.

You can orbit around with left mouse button down and drag. To zoom use the mouse scroll and to pan drag with the right mouse button.

End of part 1

That’s all folks! In the next article we will load the checkers board and the pieces; we’ll also add some lights in scene.

You can download the current phase of the project from [here](http://www.osd.net/blog/wp-content/uploads/2013/04/webgl_3d_checkers_part1.zip).

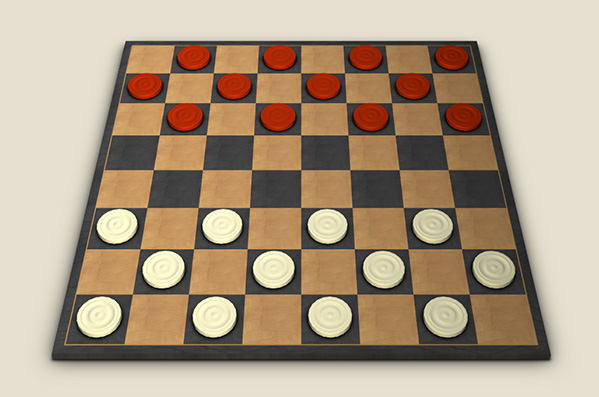
If you have questions, suggestions or improvements don’t hesitate to add a comment and let me know about them.

## 3D board game in a browser using WebGL and Three.js, part 2

[4 comm.](http://www.osd.net/blog/web-development/3d-board-game-in-a-browser-using-webgl-and-three-js-part-2/#comments)

posted by [CAM](http://www.osd.net/blog/author/adrian/), June 5, 2013  
Categories: [JavaScript](http://www.osd.net/blog/category/web-development/javascript/), [Web Development](http://www.osd.net/blog/category/web-development/)  
Taggs [3D](http://www.osd.net/blog/tag/3d/), [three.js](http://www.osd.net/blog/tag/three-js/), [WebGL](http://www.osd.net/blog/tag/webgl/)

In a [previous article](http://www.osd.net/blog/web-development/3d-board-game-in-a-browser-using-webgl-and-three-js-part-1/) we’ve built the basic structure of our 3D checkers game and ended up with drawing a simple cube. In this article we’ll add the board and the pieces. By the end of this part you’ll have something like that looks like this:

[](http://www.osd.net/blog/wp-content/uploads/2013/06/webgl_3d_checkers_board_with_pieces2.jpg)

First, download and copy the content from the 3D assets [archive](http://www.osd.net/blog/wp-content/uploads/2013/06/webgl_checkers_3d_assets.zip) into 3d\_assets/ folder.

**Note:** Unless specified otherwise, most of the code from this article will go into **js/BoardController.js**.

Let’s start with some variable declarations:

|  |  |  |
| --- | --- | --- |
| 1 | var cameraController; | |
| 2 |  |

|  |  |
| --- | --- |
| 3 | var lights = {}; |
| 4 | var materials = {}; | |

|  |  |
| --- | --- |
| 5 |  |
| 6 | var pieceGeometry = null; | |

|  |  |
| --- | --- |
| 7 | var boardModel; |
| 8 | var groundModel; | |

|  |  |  |
| --- | --- | --- |
| 9 |  | |
| 10 | | var squareSize = 10; | |

The name of the variables should be self-explanatory and as we go along they will make more sense.

## Creating a light

Before adding any objects into our scene, we need to create at least one light or else the objects will appear black. So, create a **initLights** function after initEngine:

|  |  |  |
| --- | --- | --- |
| 1 | function initLights() { | |
| 2 | // top light |

|  |  |  |
| --- | --- | --- |
| 3 | lights.topLight = new THREE.PointLight(); | |
| 4 | lights.topLight.position.set(0, 150, 0); |

|  |  |  |
| --- | --- | --- |
| 5 | lights.topLight.intensity = 1.0; | |
| 6 |  |

|  |  |  |
| --- | --- | --- |
| 7 | // add the lights in the scene | |
| 8 | scene.add(lights.topLight); |

|  |  |
| --- | --- |
| 9 | } |

For now just a point light is created and placed above the scene. The newly created function must be called from drawBoard after the initEngine call.

## Creating the materials

Now we need to create some materials for the objects so that they will not appear in wireframe like the cube from the previous article. So, create a **initMaterials** function after initLights:

|  |  |  |
| --- | --- | --- |
| 1 | function initMaterials() { | |
| 2 | // board material |

|  |  |
| --- | --- |
| 3 | materials.boardMaterial = new THREE.MeshLambertMaterial({ |
| 4 | map: THREE.ImageUtils.loadTexture(assetsUrl + 'board\_texture.jpg') | |

|  |  |  |
| --- | --- | --- |
| 5 | }); | |
| 6 |  |

|  |  |
| --- | --- |
| 7 | // ground material |
| 8 | materials.groundMaterial = new THREE.MeshBasicMaterial({ | |

|  |  |  |
| --- | --- | --- |
| 9 | transparent: true, | |
| 10 | | map: THREE.ImageUtils.loadTexture(assetsUrl + 'ground.png') | |

|  |  |  |
| --- | --- | --- |
| 11 | }); | |
| 12 |  |

|  |  |
| --- | --- |
| 13 | // dark square material |
| 14 | materials.darkSquareMaterial = new THREE.MeshLambertMaterial({ | |

|  |  |  |
| --- | --- | --- |
| 15 | map: THREE.ImageUtils.loadTexture(assetsUrl + 'square\_dark\_texture.jpg') | |
| 16 | }); |

|  |  |
| --- | --- |
| 17 | // |
| 18 | // light square material | |

|  |  |
| --- | --- |
| 19 | materials.lightSquareMaterial = new THREE.MeshLambertMaterial({ |
| 20 | map: THREE.ImageUtils.loadTexture(assetsUrl + 'square\_light\_texture.jpg') | |

|  |  |  |
| --- | --- | --- |
| 21 | }); | |
| 22 |  |

|  |  |
| --- | --- |
| 23 | // white piece material |
| 24 | materials.whitePieceMaterial = new THREE.MeshPhongMaterial({ | |

|  |  |  |
| --- | --- | --- |
| 25 | color: 0xe9e4bd, | |
| 26 | shininess: 20 |

|  |  |  |
| --- | --- | --- |
| 27 | }); | |
| 28 |  |

|  |  |
| --- | --- |
| 29 | // black piece material |
| 30 | materials.blackPieceMaterial = new THREE.MeshPhongMaterial({ | |

|  |  |  |
| --- | --- | --- |
| 31 | color: 0x9f2200, | |
| 32 | shininess: 20 |

|  |  |  |
| --- | --- | --- |
| 33 | }); | |
| 34 |  |

|  |  |
| --- | --- |
| 35 | // pieces shadow plane material |
| 36 | materials.pieceShadowPlane = new THREE.MeshBasicMaterial({ | |

|  |  |
| --- | --- |
| 37 | transparent: true, |
| 38 | map: THREE.ImageUtils.loadTexture(assetsUrl + 'piece\_shadow.png') | |

|  |  |  |
| --- | --- | --- |
| 39 | }); | |
| 40 | } |

On line 3 a non-shiny material for the board is created with a texture for the color map.

On line 8 a flat material is created for the ground. The ground will fake the board shadow, so a transparent texture is loaded for the color map. When using transparent images the material’s transparent property must be set to true.

The board’s squares will be created as individual objects, so on lines 14 and 19 the materials for the dark and light square is created.

On lines 24 and 30 a material is created for each piece type. Because we want some shininess on the pieces, a [MeshPhongMaterial](http://threejs.org/docs/58/#Reference/Materials/MeshPhongMaterial) was chosen.

The shadow cast by the pieces on the board will be faked using planes placed under them with a material as shown on line 36.

Now that we have the initMaterials function created, let’s call it from drawBoard just after the initLights call.

## Loading 3D objects from external files

With the lights and materials created, we’re ready to load the board and the piece. If you look in the 3d\_assets/ folder you’ll find the following files: **board.js**, board.obj, **piece.js** and piece.obj.

The .js files are the ones that will be loaded. These are created from the .obj files using the python script from three.js/utils/converters/obj/convert\_obj\_three.py like this:

|  |  |
| --- | --- |
| 1 | python convert\_obj\_three.py -i board.obj -o board.js |

**Note:** You’ll need Python 2.x installed on you computer to run that command.

If you want to modify the 3D objects or maybe to just have a look at them, you can open the .obj files in a 3D modeling application like Autodesk Maya or 3ds Max.

**Note:** The pivot point for the board was modified to sit on top of it, at the top-left corner; the pivot point for the piece was placed at the base.

Now let’s modify our initObjects function like this:

|  |  |
| --- | --- |
| 1 | function initObjects(callback) { |
| 2 | var loader = new THREE.JSONLoader(); | |

|  |  |
| --- | --- |
| 3 | var totalObjectsToLoad = 2; // board + the piece |
| 4 | var loadedObjects = 0; // count the loaded pieces | |

|  |  |
| --- | --- |
| 5 |  |
| 6 | // checks if all the objects have been loaded | |

|  |  |  |
| --- | --- | --- |
| 7 | function checkLoad() { | |
| 8 | loadedObjects++; |

|  |  |  |
| --- | --- | --- |
| 9 |  | |
| 10 | | if (loadedObjects === totalObjectsToLoad && callback) { | |

|  |  |  |
| --- | --- | --- |
| 11 | callback(); | |
| 12 | } |

|  |  |  |
| --- | --- | --- |
| 13 | } | |
| 14 |  |

|  |  |
| --- | --- |
| 15 | // load board |
| 16 | loader.load(assetsUrl + 'board.js', function (geom) { | |

|  |  |  |
| --- | --- | --- |
| 17 | boardModel = new THREE.Mesh(geom, materials.boardMaterial); | |
| 18 |  |

|  |  |  |
| --- | --- | --- |
| 19 | scene.add(boardModel); | |
| 20 |  |

|  |  |  |
| --- | --- | --- |
| 21 | checkLoad(); | |
| 22 | }); |

|  |  |
| --- | --- |
| 23 |  |
| 24 | // load piece | |

|  |  |  |
| --- | --- | --- |
| 25 | loader.load(assetsUrl + 'piece.js', function (geometry) { | |
| 26 | pieceGeometry = geometry; |

|  |  |
| --- | --- |
| 27 |  |
| 28 | checkLoad(); | |

|  |  |  |
| --- | --- | --- |
| 29 | }); | |
| 30 |  |

|  |  |  |
| --- | --- | --- |
| 31 | scene.add(new THREE.AxisHelper(200)); | |
| 32 | } |

On line 2 an instance of [JSONLoader](http://threejs.org/docs/58/#Reference/Loaders/JSONLoader) is created that will be used to load the external 3D models. Since the objects are loaded via AJAX we can’t call the success callback right away, but after they are loaded. So between the lines 3 and 13 we have some code that will help us to do that.

On line 16 the board model is loaded by passing a URL and a callback function to the loader’s load method. After the board is loaded we create a mesh from the loaded geometry and add it to the scene. On line 21 we verify if all the needed objects have been loaded so that we could call the success callback passed to initObjects function.

On line 25 the piece is loaded. Since we’ll need to create multiple pieces from the loaded geometry we save it for later use. We’ll add the pieces into our scene  later.

On line 31 we’ll add an **AxisHelper** into the scene to be able to see the XYZ axes to help us visualize the objects position.

Loading our project in the browser right now you should see something like this:

[](http://www.osd.net/blog/wp-content/uploads/2013/06/webgl_3d_checkers_board1.jpg)

As you can see the board is automatically positioned so that the top-left corner is at (0,0,0). This happened because the pivot point position was changed in the 3D modeling application before exporting the model. In this way we’ll only have to work with positive values for X and Z axes when placing the squares and pieces.

Now let’s add the squares and the ground that will fake the board shadow:

|  |  |
| --- | --- |
| 1 | // load board |
| 2 | loader.load(assetsUrl + 'board.js', function (geom) { | |

|  |  |  |
| --- | --- | --- |
| 3 | boardModel = new THREE.Mesh(geom, materials.boardMaterial); | |
| 4 | boardModel.position.y = -0.02; |

|  |  |
| --- | --- |
| 5 |  |
| 6 | ... | |

|  |  |
| --- | --- |
| 7 |  |
| 8 | // load piece | |

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | ... | | |
| 10 | |  |

|  |  |
| --- | --- |
| 11 | // add ground |
| 12 | groundModel = new THREE.Mesh(new THREE.PlaneGeometry(100, 100, 1, 1), materials.groundMaterial); | |

|  |  |  |
| --- | --- | --- |
| 13 | groundModel.position.set(squareSize \* 4, -1.52, squareSize \* 4); | |
| 14 | groundModel.rotation.x = -90 \* Math.PI / 180; |

|  |  |
| --- | --- |
| 15 | // |
| 16 | scene.add(groundModel); | |

|  |  |
| --- | --- |
| 17 |  |
| 18 | // create the board squares | |

|  |  |  |
| --- | --- | --- |
| 19 | var squareMaterial; | |
| 20 | // |

|  |  |
| --- | --- |
| 21 | for (var row = 0; row < 8; row++) { |
| 22 | for (var col = 0; col < 8; col++) { | |

|  |  |
| --- | --- |
| 23 | if ((row + col) % 2 === 0) { // light square |
| 24 | squareMaterial = materials.lightSquareMaterial; | |

|  |  |
| --- | --- |
| 25 | } else { // dark square |
| 26 | squareMaterial = materials.darkSquareMaterial; | |

|  |  |  |
| --- | --- | --- |
| 27 | } | |
| 28 |  |

|  |  |  |
| --- | --- | --- |
| 29 | var square = new THREE.Mesh(new THREE.PlaneGeometry(squareSize, squareSize, 1, 1), squareMaterial); | |
| 30 |  |

|  |  |
| --- | --- |
| 31 | square.position.x = col \* squareSize + squareSize / 2; |
| 32 | square.position.z = row \* squareSize + squareSize / 2; |

|  |  |  |
| --- | --- | --- |
| 33 | square.position.y = -0.01; | |
| 34 |  |

|  |  |  |
| --- | --- | --- |
| 35 | square.rotation.x = -90 \* Math.PI / 180; | |
| 36 |  |

|  |  |  |
| --- | --- | --- |
| 37 | scene.add(square); | |
| 38 | } |

|  |  |
| --- | --- |
| 39 | } |

To be sure that the pieces and the squares do not intersect the board is moved slightly in negative Y axis on line 4. We’ve used a value of -0.02 because the squares we’ll be positioned at -0.01 along the Y axis.

On line 12 the ground is created and on line 13 it’s moved to match the board position using the squareSize property. Remember that we’ve set the value for this property to 10 in the beginning. The board and the piece have been modeled especially for that value. The -1.52 value for the Y axis was taken from the 3D modeling software by measuring the board height and by adding the 0.02 value used to move the board down.

On line 14 the ground needs to be rotated -90 degree along the X axis because a THREE.PlaneGeometry will be positioned along the XY plane.

**Note:** Three.js works with radians, so if we want to use degrees we need to convert the values.

From line 19 the board’s squares are added.

Right now the camera will look and orbit around the board’s top-left corner, around the (0,0,0) position. Let’s change that by going into the initEngine function and modifying the camera properties:

|  |  |
| --- | --- |
| 1 | // create camera |
| 2 | camera = new THREE.PerspectiveCamera(35, viewWidth / viewHeight, 1, 1000); | |

|  |  |
| --- | --- |
| 3 | camera.position.set(squareSize \* 4, 120, 150); |
| 4 | cameraController = new THREE.OrbitControls(camera, containerEl); | |

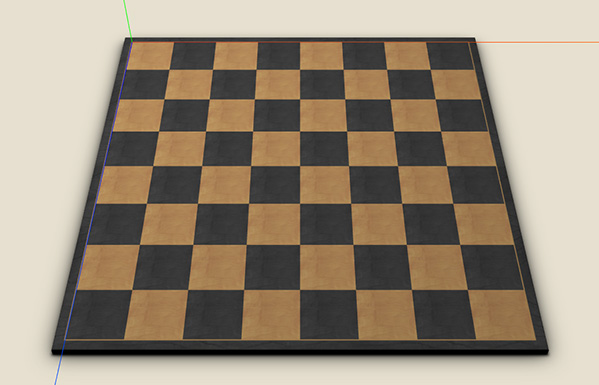
|  |  |
| --- | --- |
| 5 | cameraController.center = new THREE.Vector3(squareSize \* 4, 0, squareSize \* 4); |

On line 3 we’ve modified the X position of the camera and on line 5 we’ve told the camera controller to look at and move around the board’s center position.

The top light also needs to be repositioned:

|  |  |
| --- | --- |
| 1 | lights.topLight.position.set(squareSize \* 4, 150, squareSize \* 4); |

Refresh the browser and you should see this:

[](http://www.osd.net/blog/wp-content/uploads/2013/06/webgl_3d_checkers_board2.jpg)

## Adding the checkers pieces

With the board added and looking nice we are now ready to add the pieces in their starting position. To make that happen we’ll need a variable that will hold the pieces and one public method to add tthem. With that in mind modify the BoardController.js file like this:

|  |  |
| --- | --- |
| 1 | ... |
| 2 | var squareSize = 10; | |

|  |  |
| --- | --- |
| 3 |  |
| 4 | var board = [ | |

|  |  |
| --- | --- |
| 5 | [0, 0, 0, 0, 0, 0, 0, 0], |
| 6 | [0, 0, 0, 0, 0, 0, 0, 0], |

|  |  |
| --- | --- |
| 7 | [0, 0, 0, 0, 0, 0, 0, 0], |
| 8 | [0, 0, 0, 0, 0, 0, 0, 0], |

|  |  |  |
| --- | --- | --- |
| 9 | [0, 0, 0, 0, 0, 0, 0, 0], | |
| 10 | | [0, 0, 0, 0, 0, 0, 0, 0], | |

|  |  |  |
| --- | --- | --- |
| 11 | [0, 0, 0, 0, 0, 0, 0, 0], | |
| 12 | [0, 0, 0, 0, 0, 0, 0, 0] |

|  |  |  |
| --- | --- | --- |
| 13 | ]; | |
| 14 |  |

|  |  |  |
| --- | --- | --- |
| 15 | ... | |
| 16 |  |

|  |  |  |
| --- | --- | --- |
| 17 | this.drawBoard = function () { | |
| 18 | ... |

|  |  |  |
| --- | --- | --- |
| 19 | }; | |
| 20 |  |

|  |  |
| --- | --- |
| 21 | this.addPiece = function (piece) { |
| 22 | var pieceMesh = new THREE.Mesh(pieceGeometry); | |

|  |  |  |
| --- | --- | --- |
| 23 | var pieceObjGroup = new THREE.Object3D(); | |
| 24 | // |

|  |  |
| --- | --- |
| 25 | if (piece.color === CHECKERS.WHITE) { |
| 26 | pieceObjGroup.color = CHECKERS.WHITE; | |

|  |  |  |
| --- | --- | --- |
| 27 | pieceMesh.material = materials.whitePieceMaterial; | |
| 28 | } else { |

|  |  |
| --- | --- |
| 29 | pieceObjGroup.color = CHECKERS.BLACK; |
| 30 | pieceMesh.material = materials.blackPieceMaterial; | |

|  |  |  |
| --- | --- | --- |
| 31 | } | |
| 32 |  |

|  |  |
| --- | --- |
| 33 | // create shadow plane |
| 34 | var shadowPlane = new THREE.Mesh(new THREE.PlaneGeometry(squareSize, squareSize, 1, 1), materials.pieceShadowPlane); | |

|  |  |  |
| --- | --- | --- |
| 35 | shadowPlane.rotation.x = -90 \* Math.PI / 180; | |
| 36 |  |

|  |  |
| --- | --- |
| 37 | pieceObjGroup.add(pieceMesh); |
| 38 | pieceObjGroup.add(shadowPlane); | |

|  |  |
| --- | --- |
| 39 |  |
| 40 | pieceObjGroup.position = boardToWorld(piece.pos); | |

|  |  |
| --- | --- |
| 41 |  |
| 42 | board[ piece.pos[0] ][ piece.pos[1] ] = pieceObjGroup; | |

|  |  |
| --- | --- |
| 43 |  |
| 44 | scene.add(pieceObjGroup); | |

|  |  |
| --- | --- |
| 45 | }; |

On line 4 we’ll use a 2 dimensional 8×8 array to create an internal board representation.

From line 21 the **addPiece** method is defined. This function will expect as its only parameter an object with properties related to the piece, like color and position.

Because we’ll use a textured plane under the piece to fake the shadow on the board, on line 23 an instance of [Object3D](http://threejs.org/docs/58/#Reference/Core/Object3D) is created that will group the piece mesh and the shadow plane together.

Between line 25 and 31 the piece group gets assigned a WHITE or BLACK color and the right material is assigned to the piece mesh.

On line 34 the shadow plane is created and on line 37 and 38 the piece mesh and the shadow plane are grouped together.

On line 40 the piece group is positioned using the pos property from the function’s parameter object. Since its value will be an array holding the position in a 8×8 array, we need to find out the position in our 3D space using the boardToWorld function that will be created immediately.

On line 42 the piece group is stored in the internal board representation and also added in the scene on line 44.

Now let’s add the **boardToWorld** function right after onAnimationFrame:

|  |  |
| --- | --- |
| 1 | function boardToWorld (pos) { |
| 2 | var x = (1 + pos[1]) \* squareSize - squareSize / 2; | |

|  |  |  |
| --- | --- | --- |
| 3 | var z = (1 + pos[0]) \* squareSize - squareSize / 2; | |
| 4 |  |

|  |  |  |
| --- | --- | --- |
| 5 | return new THREE.Vector3(x, 0, z); | |
| 6 | } |

The above function will return the center position of the right square.

The last thing we need to do in BoardController.js is to make the drawBoard function receive a callback:

|  |  |  |
| --- | --- | --- |
| 1 | this.drawBoard = function (callback) { | |
| 2 | ... |

|  |  |  |
| --- | --- | --- |
| 3 | initObjects(function () { | |
| 4 | onAnimationFrame(); |

|  |  |
| --- | --- |
| 5 |  |
| 6 | callback(); | |

|  |  |  |
| --- | --- | --- |
| 7 | }); | |
| 8 | } |

Now open the **Game.js** file and modify it like this:

|  |  |  |
| --- | --- | --- |
| 1 | var CHECKERS = { | |
| 2 | WHITE: 1, |

|  |  |  |
| --- | --- | --- |
| 3 | BLACK: 2 | |
| 4 | }; |

|  |  |
| --- | --- |
| 5 |  |
| 6 | ... | |

|  |  |
| --- | --- |
| 7 |  |
| 8 | var boardController = null; | |

|  |  |  |
| --- | --- | --- |
| 9 |  | |
| 10 | | var board = [ | |

|  |  |
| --- | --- |
| 11 | [0, 0, 0, 0, 0, 0, 0, 0], |
| 12 | [0, 0, 0, 0, 0, 0, 0, 0], |

|  |  |
| --- | --- |
| 13 | [0, 0, 0, 0, 0, 0, 0, 0], |
| 14 | [0, 0, 0, 0, 0, 0, 0, 0], |

|  |  |
| --- | --- |
| 15 | [0, 0, 0, 0, 0, 0, 0, 0], |
| 16 | [0, 0, 0, 0, 0, 0, 0, 0], |

|  |  |  |
| --- | --- | --- |
| 17 | [0, 0, 0, 0, 0, 0, 0, 0], | |
| 18 | [0, 0, 0, 0, 0, 0, 0, 0] |

|  |  |  |
| --- | --- | --- |
| 19 | ]; | |
| 20 |  |

|  |  |  |
| --- | --- | --- |
| 21 | ... | |
| 22 |  |

|  |  |  |
| --- | --- | --- |
| 23 | function init() { | |
| 24 | ... |

|  |  |  |
| --- | --- | --- |
| 25 | boardController.drawBoard(onBoardReady); | |
| 26 | } |

|  |  |
| --- | --- |
| 27 |  |
| 28 | function onBoardReady() { | |

|  |  |  |
| --- | --- | --- |
| 29 | // setup the board pieces | |
| 30 | var row, col, piece; |

|  |  |
| --- | --- |
| 31 | // |
| 32 | for (row = 0; row < board.length; row++) { | |

|  |  |
| --- | --- |
| 33 | for (col = 0; col < board[row].length; col++) { |
| 34 | if (row < 3 && (row + col) % 2) { // black piece | |

|  |  |
| --- | --- |
| 35 | piece = { |
| 36 | color: CHECKERS.BLACK, | |

|  |  |  |
| --- | --- | --- |
| 37 | pos: [row, col] | |
| 38 | }; |

|  |  |  |
| --- | --- | --- |
| 39 | } else if (row > 4 && (row + col) % 2) { // white piece | |
| 40 | piece = { |

|  |  |  |
| --- | --- | --- |
| 41 | color: CHECKERS.WHITE, | |
| 42 | pos: [row, col] |

|  |  |
| --- | --- |
| 43 | }; |
| 44 | } else { // empty square | |

|  |  |  |
| --- | --- | --- |
| 45 | piece = 0; | |
| 46 | } |

|  |  |
| --- | --- |
| 47 |  |
| 48 | board[row][col] = piece; | |

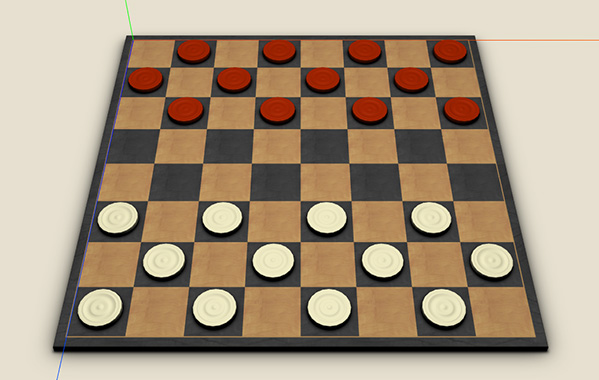
|  |  |
| --- | --- |
| 49 |  |
| 50 | if (piece) { | |

|  |  |  |
| --- | --- | --- |
| 51 | boardController.addPiece(piece); | |
| 52 | } |

|  |  |  |
| --- | --- | --- |
| 53 | } | |
| 54 | } |

|  |  |
| --- | --- |
| 55 | } |

The above code should be self-explanatory and if you do a refresh in the browser you should see something like this:

[](http://www.osd.net/blog/wp-content/uploads/2013/06/webgl_3d_checkers_board_with_pieces.jpg)

You’ll notice that the pieces don’t look as nice as in the image from the beginning of the article. That’s because more lights are needed in the scene. You can download the sample code and have a look in initLights function and onAnimationFrame to see how to add the new lights.

## End of part 2

That’s all folks! In the next and final article we will allow the pieces to be dragged and some checkers game logic will be added also.

You can download the current phase of the project from [here](http://www.osd.net/blog/wp-content/uploads/2013/06/webgl_3d_checkers_part2.zip).

If you have questions, suggestions or improvements don’t hesitate to add a comment and let me know about them.