Create virtual reality panoramas with three.js

November 23, 2015

The web is for everyone, but sometimes technology arrives that creates a barrier to this. Virtual reality is one of those barriers. Unless you have a very expensive device like an Oculus Rift or Samsung Gear you can't access VR content.

GET THE TUTORIAL CODE

However, if you have a smartphone then you can buy an inexpensive virtual reality viewer, like Google Cardboard, for the price of a film (between ten and twenty pounds). These low-tech devices are little more than a cleverly constructed piece of cardboard with two lenses. They take the left and right images on a screen and trick your eyes into believing it's a single image. Its simple design means it's easy to replicate and cheap to manufacture.

You'll learn how to use three.js to create a 360-degree video panorama that the user can look around with a device and Google Cardboard – all within the browser!

Much of the terminology we'll use comes from the videogames world. As we're using three.js, there's much more interactivity that you can add to this. It allows you to add any 3D object into the world, and using ray casting (to detect what the user is looking at) you could trigger interactions within it. We're going to keep it simple by introducing the basic concepts and provide a world for the user to look around and be immersed within.

---

1. Create index.html

Start by creating a blank page with a video element and a container. The video element's behaviour is controlled by attributes, in this case we've told it to play automatically and loop indefinitely. The more sources you provide, the greater the number of browsers that will be able to play it.

```
<div id="canvas"></div>
<video id="video" autoplay loop>
<source src="videos/panorama.mp4" type="video/mp
4"></source>
Your browser does not support the video element
</video>
```

2. Reference script files

Much of the heavy lifting has been done for us in the form of three.js plugins. The official ones that we have made use of are StereoEffect, DeviceOrientationControls, OrbitControls. Lastly paper.js will glue all of it together.

```
<script src="https://cdnjs.cloudflare.com/ajax/l
ibs/three.js/r71/three.min.js"></script>
<script src="js/third-party/threejs/StereoEffec
t.js"></script>
<script src="js/third-party/threejs/DeviceOrient
ationControls.js"></script>
<script src="js/third-party/threejs/OrbitControl
s.js"></script>
<script src="js/paper.js"></script>
```

3. Create paper.js

Most three.js files have three lifecycle functions: init, render and animate. Within paper.js, we'll kick off by declaring some variables which will be accessed by more than one of these.

```
(function() {
'use strict';
var camera, scene, renderer;
var effect, controls;
var element, container;
var videoTexture, videoMesh;
})();
```

4. Clock and initialise

The clock is a simple but extremely handy utility. This is useful for us because when animating or moving around, we need to know how fast to animate and how much time has passed since the app started or was last called.

```
var clock = new THREE.Clock();
init();
```

5. Start initialising scene

The renderer is the heart of outputting pixels to the page. The element is the DOM element of the Canvas. The scene is the space that everything is put into. Just like a movie, digital scenes contain objects, lights and cameras.

```
function init() {
renderer = new THREE.WebGLRenderer();
element = renderer.domElement;
container = document.getElementById('canvas');
container.appendChild(element);
```

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

6. Create sphere geometry

For the panorama, create a sphere and look in the inside of it. To make the sphere we generate its geometry – its mathematical 3D representation. Then apply a matrix, inverting it so the outside plane is on the inside.

```
var sphere = new THREE.SphereGeometry(500, 60, 4
0);
sphere.applyMatrix(new THREE.Matrix4().makeScale
(-1, 1, 1));
```

7. Play video on devices

On iOS and Android, videos can't be automatically played, it needs a user to interact with the page beforehand. This could be linked to a button, or in this case, any click on the page will trigger the video to play. On iOS the video will be paused as you pan around it.

```
var video = document.getElementById('video');
function bindPlay () {
video.play();
document.body.removeEventListener('click', bindP
lay);
}
document.body.addEventListener('click', bindPla
y, false);
```

8. Create a video texture

The video texture maps the video to the sphere. Create the texture by passing it the video element, then set the min filter to linear as the size is unlikely to be a power of 2 (eg 16 by 16). The material describes the appearance of the

object. The basic mesh will show as a flat polygon.

```
var videoTexture = new THREE.VideoTexture(vide
o);
videoTexture.minFilter = THREE.LinearFilter;
var videoMaterial = new THREE.MeshBasicMaterial
({
map: videoTexture
});
videoMesh = new THREE.Mesh(sphere, videoMateria
l);
```

9. Camera effects

The stereo effect works by passing the renderer to it and rendering everything out twice, but slightly offset. This gives the illusion of depth and VR its appeal. To see the scene we need to place a camera within it. In this case, the perspective camera is used for a first-person view.

```
effect = new THREE.StereoEffect(renderer);
camera = new THREE.PerspectiveCamera(95, 1, 0.00
1, 700);
```

10. Set camera’s position

The perspective camera takes arguments in the following order: field of view, aspect ratio, depth to start rendering objects (near), and depth to stop rendering objects (far). Once created, positioning the camera is as simple as setting its 3D coordinates: x, y and z.

```
camera.position.set(100, 100, 100);
scene.add(camera);
```

11. Add controls for mouse

Next up we'll add orbit controls. This allows you to click and drag to look around, which is useful for debugging when not on a device. We then set the starting position of the controls to the same position as the camera.

```
controls = new THREE.OrbitControls(camera, eleme
nt);
controls.rotateUp(Math.PI / 4);
controls.target.set(
camera.position.x + 0.1,
camera.position.y,
camera.position.z
);
controls.noZoom = true;
controls.noPan = true;
```

12. Change controls

If the environment that our code is running in fires the device orientation event then instead of using orbit controls it'll switch to using device orientation controls. This means users can simply rotate their device to look around instead of tapping and dragging.

```
function setOrientationControls(e) {
if (!e.alpha) {
return;
}
controls = new THREE.DeviceOrientationControls(c
amera, true);
controls.connect();
controls.update();
```

13. Remove event listener

Once the controls are set to use device orientation, we won't want to reinstantiate those controls every time the event is fired. To fix this, remove setOrientationControls at the bottom of the previous function.

```
window.removeEventListener('deviceorientation',
setOrientationControls, true);
}
```

14. Device orientation

The device orientation event is fired when the accelerometer detects a change in the device's orientation. We're interested in the z axis, otherwise known as alpha (beta and gamma are x and y respectively). Alpha goes from 0 to 360. In the init function we'll add the listener for device orientation.

```
window.addEventListener('deviceorientation', set
OrientationControls, true);
```

15. Add sphere to scene

The final part of init adds the video mesh, the culmination of the sphere and video texture, to the scene. Attach a resize handler to ensure that browser resizing doesn't look strange, and kick off animate.

```
scene.add(videoMesh);
window.addEventListener('resize', resize, fals
e);
animate();
```

16. Resize function

Resize is responsible for making sure the aspect ratio is maintained when resizing the window (or if going from portrait to landscape). Then the renderer and stereo effect are updated with the new width and height.

```
function resize () {
```

```
var width = container.offsetWidth;
var height = container.offsetHeight;
camera.aspect = width / height;
camera.updateProjectionMatrix();
renderer.setSize(width, height);
effect.setSize(width, height);
}
```

17. Update function

The update function calls resize and updates the controls with a new delta from the clock. The delta is the number of seconds since the clock's getDelta method was last called. It'll be called by the animate function, which will be written shortly, and invoke getDelta.

```
function update (dt) {
resize();
controls.update(dt);
}
```

18. Render function

The render function outputs everything to the screen. 'Effect' here is the stereo imaging and sets up the left and right images which behave as separate cameras. Internally it then uses the renderer (which we provided much earlier) to output to the Canvas.

```
function render () {
effect.render(scene, camera);
}
```

19. Animate function

Animate keeps our panorama responsive to movement. At each frame it calls update and render. Crucially, it also calls itself as a requestAnimationFrame callback. This ensures

that the camera is perpetually updated.

```
function animate () {
requestAnimationFrame(animate);
update(clock.getDelta());
render();
}
```

20. Fullscreen ahead

Earlier we referenced a full-screen function which is triggered when the user clicks the Canvas element. This should be straightforward but there are many vendor prefixed versions that we need to accommodate. The first of the if statements is the standard, nonprefixed one and we cascade down Microsoft, Mozilla and WebKit.

```
function fullscreen () {
if (container.requestFullscreen) {
container.requestFullscreen();
} else if (container.msRequestFullscreen) {
container.msRequestFullscreen();
} else if (container.mozRequestFullScreen) {
container.mozRequestFullScreen();
} else if (container.webkitRequestFullscreen) {
container.webkitRequestFullscreen();
}
}
```

21. Finishing touches

Through the goodwill of the developer community, the technology exists to place 360-degree panoramas in the browser. There are currently limitations on playing back video within iOS which needs to be addressed but as an interim, an image-only panorama can be used.

```
#canvas {
position: absolute;
```

```
top: 0; bottom: 0;
left: 0; right: 0;
}
#video {
position: absolute;
left: -9999em;
}
```

SHARE