HackMe 2.0 - XperienZ

HKME-26 – Stereoscopic Video Render using Three.js

Version 1.0

Published on 01/10/2017

Document Revision History

Date	Version	Description	Author
01/09/2017	1.0	First Draft	Saurabh Nutan Sohoni

References

No.	Artefact Name
1.	https://www.sitepoint.com/bringing-vr-to-web-google-cardboard-three-js/
2.	https://gist.github.com/chrisprobst/8441970
3.	https://threejs.org/examples/

Contents

1. Introduction

1.1 Purpose

The purpose of this document is to guide developers in creating a sample application which converts 2D videos (video shot by a normal camera) to 3D using JavaScript for enhanced experience. 3D output can be viewed in Virtual Reality (VR) headsets like Google cardboard, Oculus Rift for real-like experience.

1.2 Scope

Scope of this project is limited to development of a simple application which renders a 2D video into 3D. Render covers only visual aspect, audio aspect being played as it is. In a conventional VR headset, two lenses with different optical separation are installed which creates depth perception. Videos with two frames separated by a distance are played in the application and when viewed using the VR headset, a real-like experience is created. This render is done using Three.js, a JavaScript library which uses WebGL as render engine.

1.3 Hardware requirements

The only hardware required for this application is a mobile device of the size same as the size of VR headset. For better results, any smartphone with screen size greater than 5 inch will be desirable.

1.4 Software requirements

This project uses a JavaScript library by the name Three.js for 2D to 3D render. Three.js has been in use for more than 5 years now and is available for use under MIT license.

2. OVERVIEW OF CONCEPT AND METHODOLOGY USED

This section begins with a brief overview of Three.js. Then it lists some important concepts of video rendering in general. It also lists some methodologies related to positioning of camera, renderer and canvas; all used in a rendering engine. With brief idea of these principles, this guide will dive deep into implementation of desired application.

2.1 What is Three.js?

Three.js is a cross-browser JavaScript library used to create and display animated 3D graphic content in a web browser. The render engine used by Three.js is WebGL. A typical Three.js application starts with creating a scene with a 3D object for e.g.: A sphere or a cube. Then some graphic content is imposed upon this object and viewed through a camera placed strategically to create an immersive effect. There are also variety of filter available

for smoothening and sharpening of objects placed into scene for creating a differential effect based on illumination.

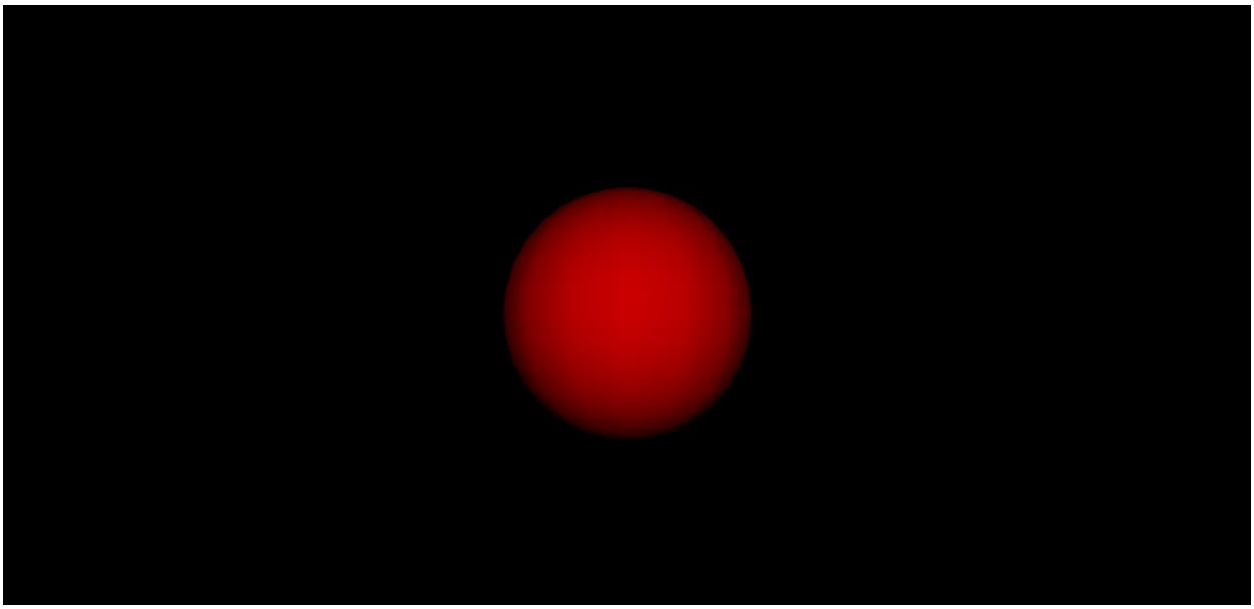
2.2 Starting with computer graphics – Basic terminology

Rendering a graphic on screen is achieved by following similar principles as followed by an artist while drawing/painting a picture or nature or someone's portrait. It has following elements:

a. A canvas – Obviously, place to draw on
b. A scene – In case of the artist, it is what he/she is looking at. In case of computer graphics, it is the background on which graphic is to be imposed.
c. A camera – Audience in case of artist, end user in case of computer graphic
d. A renderer – Without the artist, or in our case the render engine, there is nothing but an empty canvas
e. A filter, a mesh, an illumination – These are optional elements which add to the artistic beauty in the rendeLet's use them in Three.js

All of the terms mentioned above are named and used as it is in Three.js environment. Hence, it is easy for a developer to kick start and create a sample application. The following code snippet creates an application to implement render:

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

    // Add the camera to the scene.
    scene.add(camera);

    // Start the renderer.
    renderer.setSize(WIDTH, HEIGHT);

    // Attach the renderer-supplied
    // DOM element.
    container.appendChild(renderer.domElement);

    // Create a new mesh with
    // sphere geometry - we will cover
    // the sphereMaterial next!
    const sphere = new THREE.Mesh(

        new THREE.SphereGeometry(
            RADIUS,
            SEGMENTS,
            RINGS),

        sphereMaterial);

    // Move the Sphere back in Z so we
    // can see it.
    sphere.position.z = -300;

    // Finally, add the sphere to the scene.
    scene.add(sphere);

    function update () {
        // Draw!
        renderer.render(scene, camera);

        // Schedule the next frame.
        requestAnimationFrame(update);
    }

    // Schedule the first frame.
    requestAnimationFrame(update);
```



As shown above, the code snippet renders a sphere with some variation in illumination on a black colored canvas. The canvas object created in the code is attached to a container in HTML. All objects rendered in the screen are within the canvas DOM element.

2.3 requestAnimationFrame

requestAnimationFrame() is a powerful API available as a method on global window object in JavaScript. In contrast to conventional setInterval() method, it requests browser to perform an animation by calling a specified function before the next repaint occurs. This method takes an argument as a callback to be invoked before the repaint. In example given above, the update() method is called repeatedly before every browser repaint takes place. The refresh rate is usually 60 frames per second in normal desktop browser. Also, unlike setInterval(), requestAnimationFrame() doesn't keep JavaScript code running when the browser tab is not active, thereby saving a significant amount of memory.

```
function update () {
    // Draw!
    renderer.render(scene, camera);

    // Schedule the next frame.
    requestAnimationFrame(update);
}

// Schedule the first frame.
requestAnimationFrame(update);
```

3.2D to 3D VIDEO RENDER IN ACTION

We are all set to launch our application now. Sample code base is included in the root folder where this guide is found.
Few guidelines on code developed are given below:

3.1 Hidden <video> element

As stated earlier, only visual elements from the 2D video will be rendered into stereoscopic form; audio will be played in the background. To achieve this, a <video> element will be added into the markup but its display will be hidden either by using display property directly or by positioning the element out of viewport. Also, the video be added by using a source tag and it will be on auto play as soon as the application bootstraps.

3.2 Auto play of video in mobile browsers

Most mobile browsers do play a video as soon as the application bootstraps unless there is some kind of user interaction observed. This behavior is taken into account and a touch event listener on body or a start button can be used. As soon as we get some interaction, the video can be played by using play() method on the element.

3.3 Device resize, orientation and full screen mode

Most web applications on a mobile are opened in portrait mode and then a simple rotate converts it to landscape mode. Since the end user would be viewing the application in a VR headset, landscape mode will only be desirable. This simple conversion can be done by adding a listener on resize event on window. Also, for best results, this application should be viewed in full screen mode. This is achieved by using a handy JavaScript library screenfull.js.

And we are all set, let's run the application now!

3.4 Run and Render

To run the application, follow steps given below:

a. Go to root folder and execute command: npm install
b. Then start local server by using command: npm start. This will start the server at localhost, port no 8000.
c. Type the URL in desktop browser and check if the application is creating two screen renders; one for left eye and other for right eye.
d. Grab ab VR headset and open the application for an enhanced experience.

Below are few snapshots from the test run:

a. Render on a desktop screen:

b. Render as viewed from a VR headsett: