COMP27112 Introduction to Visual Computing
Coursework Assignment 1. Simple Shading
Tim Morris

Introduction

The aim of this exercise is to render a spinning, shaded cuboid on screen. To complete the exercise you'll need a web browser to display the output and a simple text editor[1] to create the html.

This piece of coursework is not assessed. But you'll need to do it to familiarise yourself with how objects are created and rendered in three.js.

Getting Started

The following code will set up a basic webpage for using three.js. Type it into your chosen text editor, save it as an html or htm file and load it into a browser. You should see very little – a blank page with no title. This skeleton will also be the starting point of the lab 2 exercise. Note that if you copy and paste this, the quotation marks will change and the hyphen will be removed, so the code won't work.

```
<html>
     <head>
     <style>
         body {
             margin 0;
             overflow: hidden;
         }
     </style>
     </head>
     <body>
          <script type = "module">
          import * as THREE from
"https://web.cs.manchester.ac.uk/three/three.js-
master/build/three.module.js";

          // Your Javascript will go here.

          </script>
     </body>
</html>
```

[1] You should ensure that your text editor saves your html files as plain text. Some editors (e.g. TextEdit on the Mac add all kinds of stuff to what you think is a plain html file, meaning that you just see the text of your file when you open it in your browser. Don't forget to give your files the correct extension.
Using an IDE will make editing easier, some are listed on Blackboard.

Creating the Scene

To be able to display anything using three.js, you need a scene, a camera and a renderer. The scene defines what you'll see, the camera defines the viewpoint, and the renderer defines the look and feel of what's on the screen. It's simplest to type code under "Your Javascript will go here", but it's probably better software engineering to make an initialisation function and write the code as the body of that function. If you do the latter you need to think about what variables should be global.

```
    var scene = new THREE.Scene();
    var camera = new THREE.PerspectiveCamera(75,
window.innerWidth / window.innerHeight, 0.1, 1000 );
    camera.position.z = 5;
    var renderer = new THREE.WebGLRenderer();
    renderer.setClearColor(0x000000, 1.0);
    renderer.setSize( window.innerWidth, window.innerHeight );
    document.body.appendChild( renderer.domElement );
    shaderCube();
```

`shaderCube()` is a function I've used which will define the cube and add it to the scene. Its body is defined in the following section.

This snippet of code sets up a WebGL object for rendering the scene. Most modern browsers support WebGL, but there are exceptions. You can check out browser compatibility at https://caniuse.com/webgl.

Adding the Cube

The final part is to add the cube itself. At the moment it will be unshaded and static. Remember that three.js objects are made up of meshes, and a mesh has a geometry that defines its shape and a material that defines its appearance. Create another few variables: `geometry, mesh, material`, and the code to create the cube by adding the following code to the end of the initialising code/function :

```
    var geometry = new THREE.BoxGeometry(2, 1, 1);
    var material = new THREE.ShaderMaterial({
        fragmentShader: fragmentShader(),
        vertexShader: vertexShader(),
    })
    mesh = new THREE.Mesh(geometry, material);
    scene.add(mesh)
```

This has defined colours for the faces of the cube, and locations for the vertices. It's added the definition of the cube into something called a "mesh" and added the mesh to a "scene". Add stub functions for `fragmentShader()` and `vertexShader()`. The vertex shader will determine what the geometry will look like, the fragment shader determines the

resulting colour. The fragment shader is always called AFTER the vertex shader. You can load this into your browser, but you'll still see nothing: nothing has been rendered yet.

Animation

Add a function called `animate()`, and call it after the call to the initialisation function. The body of the function should have the following lines

```
    renderer.render(scene, camera);
    requestAnimationFrame(animate);
```

This will result in a static, uniformly shaded cuboid.

Rotation

Rotate the cuboid by adding the following to the `animate()` function:

```
    mesh.rotation.x += 0.011;
    mesh.rotation.y += 0.013;
```

For an additional two marks, add in some other movement to the cube,

Shading

Add the following to the body of `vertexShader()`:

```
  return `
    vec4 p;
    varying vec3 vposInterpolated;

    void main() {
    p = projectionMatrix * modelViewMatrix * vec4(position,
1.0);
    gl_Position = p;
    vposInterpolated = p.xyz;
    }
  `
```

Note that `position` is a default vertex attribute representing the vertices

And add the following to the body of `fragmentShader()`:

```
return `
  varying vec3 vposInterpolated;

  void main() {
    gl_FragColor = vec4(vposInterpolated, 1.0);
  }
`
```