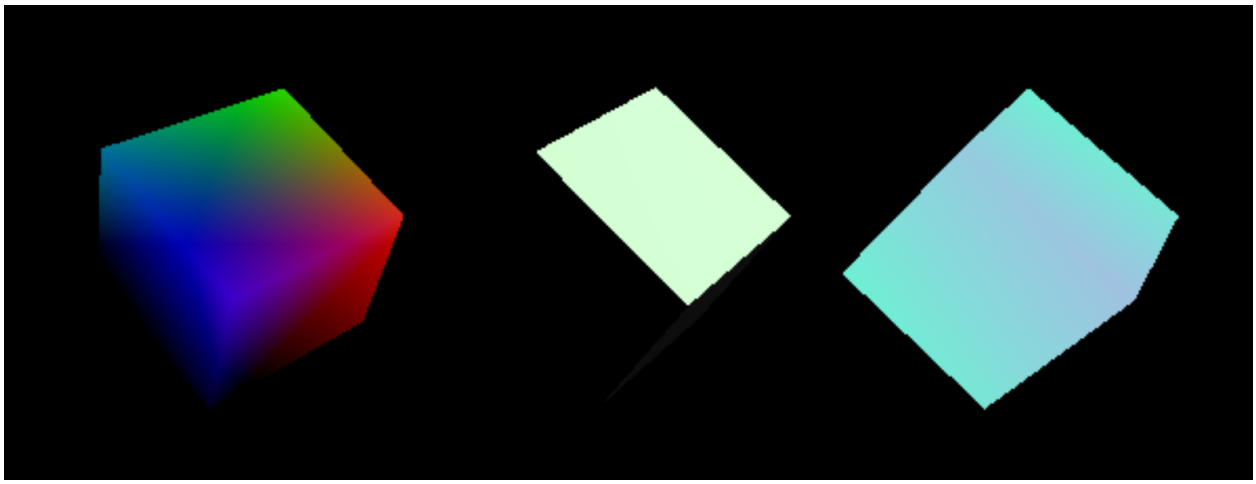
Lab 3: Materials & Shaders in Three.js



Due 4/20 Midnight

Video tutorial for this lab

To get checked off:

- Submit the link to your CMPM163Labs github (with the code from the lab). The README should have a Lab 3 heading with the following:
    - A link to a google drive video with the right sharing settings. The video should show all the cubes you made in this lab:
        - A cube with a green specular highlight made with three.js phong material
        - At least one more cube made with Three.js library materials
        - A cube with color interpolation made with your own shaders (choose your own colors)
        - At least one more cube made with your own shaders doing something different than the color interpolation
    - In the README describe how you made each cube from left to right (ex: “For the first cube on the left I interpolated between blue and pink using my own shaders.”)

- Make sure you commit frequently so we can see your history & that you push your code to github before handing it in (you will get a 0 if the code for the lab is not on your github repo)

Part 1

1. Make a new folder in your CMPM163Labs github folder called lab3.
2. In this new folder make a new html file (lab3.html) with the following code to initialize the scene and the renderer. This code sets up the rotating cube that we made in the lab last week!

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8">
		<title>Lab 3: Shaders in Three.js</title>
		<style>
			body { margin: 0; }
			canvas { display: block; }
		</style>
	</head>
	<body>
		<script src="../three.js-master/build/three.js"></script>
		<script>
			// setup the scene
			var scene = new THREE.Scene();
			var camera = new THREE.PerspectiveCamera(75,
				window.innerWidth/window.innerHeight, 0.1, 1000);
			var renderer = new THREE.WebGLRenderer();
			renderer.setSize(window.innerWidth, window.innerHeight);
			document.body.appendChild(renderer.domElement);
			camera.position.z = 5;

			// setup the cube
			var geometry = new THREE.BoxGeometry();
			var material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
			var cube = new THREE.Mesh( geometry, material );
			scene.add(cube);

			// add the light
			var light = new THREE.PointLight(0xffffff, 1, 1000);
			light.position.set(10, 10, 10);
			scene.add(light);

			function animate() {
				requestAnimationFrame(animate);
				geometry.rotateX(0.01);
				geometry.rotateY(0.01);
				renderer.render(scene, camera);
			}
			animate();
```

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

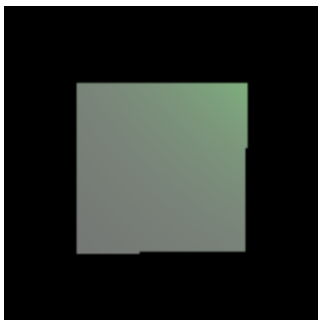
3. Test the cube in your browser! Don't forget to run your server from your CMPM163Labs folder (see last week's lab for help).

```
python -m http.server
http://localhost:8000/lab3/lab3.html
```

4. You should see a rotating green cube, and now we are going to change its material by using the built in Phong lighting model. Replace the current material with this code:

```
var material = new THREE.MeshPhongMaterial( { color: 0xdddddd,
      specular: 0x00ff00, shininess: 30 });
```

5. Now when you run the code, you should notice a specular highlight of green that appears on a grey cube while it rotates. This is the specular highlight defined in the Phong model!



6. Here we are using built in Material functionality that comes with three.js! Check out the source code of this three.js material example to see several examples of materials you can create. Create at least one new cube with a new material! You can play with transparency, reflection, color, and more!

Part 2

1. Now that you've familiarized yourself with materials in the Three.js library, we are going to learn to hook up our own shaders!
2. In your lab3 folder add another folder named "shaders."
3. In your shaders folder create a file called `vertexShader.vert`. The vertex shader runs for every vertex of a mesh of geometry. We are right now going to have a vertex shader that simply maintains the position of the vertices of the mesh, so that we keep our cube intact. The vertex shader passes the position of the vertex to the fragment shader for calculations. Put this code in your vertex shader:

```
varying vec3 vUv;

void main() {
  vUv = position;

  vec4 modelViewPosition = modelViewMatrix * vec4(position, 1.0);
  gl_Position = projectionMatrix * modelViewPosition;
}
```

4. Create a file in your shaders folder called `fragmentShader.frag`. The fragment shader outputs the pixel color for the fragment we are shading. The fragments are the triangles that form 3d shapes, each triangle has a color/shading which creates the shading for the object to appear 3D based on the lighting. Notice the varying vec3 vUv: we will use this in our fragment shader later! We are receiving the position from when it is being assigned above in the vertex shader. Add the following code to the fragment shader:

```
varying vec3 vUv;

void main() {
  gl_FragColor = vec4(0.0, 0.0, 1.0, 1.0); //rgba, return blue
}
```

5. Now we need to load these shaders into our javascript. We can continue to use the same html file we used in part 1, but we are going to add another cube to it. The tricky part is that we need to make sure both shader files have been loaded by Three.js's FileLoader before we create our material with them. Initialize the following variables in your javascript section:

```
THREE.Cache.enabled = true;
var count = 0;
var loader = new THREE.FileLoader();
var fshader, vshader;
```

6. Now we can use the loader to load in the vertex shader.

```
loader.load('shaders/vertexShader.vert',
      // onLoad callback
      function (data) {
            console.log(data); // output the text to the console
            vshader = data;
            count += 1;
```

```
            //addCoolCube(); // we will write this method
        },
        // onProgress callback
        function (xhr) {
            console.log((xhr.loaded/xhr.total * 100)+ '% loaded');
        },
        // onError callback
        function (err) {
            console.error('An error happened');
        });
```

7. Now we can load the fragment shader. Check to see if your code compiles! Look at the console and make sure that it is printing out the code from the shader files.

```
loader.load('shaders/fragmentShader.frag',
        // onLoad callback
        function (data) {
            console.log(data); // output the text to the console
            fshader = data;
            count += 1;
            //addCoolCube(); // we will write this method
        },
        // onProgress callback
        function (xhr) {
            console.log((xhr.loaded/xhr.total * 100)+ '% loaded');
        },
        // onError callback
        function (err) {
            console.error('An error happened');
        });
```

8. In your browser check the console to make sure the shaders are being printed and there aren't any errors.
9. Let's write the addCoolCube method. We need to make sure we only actually create the cube if both files have been loaded (when count is two). Uncomment both calls to `addCoolCube()` in both file loading calls, and add the following code:

```
var geometry1, material1, mesh1;

function addCoolCube() {
    if(count == 2) {
        geometry1 = new THREE.BoxGeometry(1, 1, 1);
        material1 =  new THREE.ShaderMaterial({
```

```
                        fragmentShader: fshader,
                        vertexShader: vshader,
                        precision: "mediump"});

            mesh1 = new THREE.Mesh(geometry1, material1);
            mesh1.position.x = 2;
            scene.add(mesh1);
      }
}
```

10. Now we need to add our cube to the animate function so that it also spins.

```
function animate() {
      requestAnimationFrame(animate);
      geometry.rotateX(0.01);
      geometry.rotateY(0.01);

      if(geometry1) {
            geometry1.rotateX(0.01);
            geometry1.rotateY(0.01);
      }

      renderer.render(scene, camera);
}
```

11. Check to make sure you now have a blue spinning cube! If you don't I would go over all the code to check for spelling errors. Check the console to make sure the shader code being printed is what you want. If you see the cube, feel free to adjust its position so it can be in a good place with your other spinning cubes. Now we are going to make our shader even cooler! We want to add input to our shaders from our javascript, and in GLSL Add the following uniforms to the top of your fragment shader:

```
uniform vec3 colorA;
uniform vec3 colorB;
```

12. For our shader instead of outputting the color blue for every fragment, we are going to interpolate between two input colors using the vUv's z coordinate (a * (1 - uv) + b * uv). Now change the `gl_FragColor` output to return the interpolation instead of the color blue:

```
gl_FragColor = vec4(mix(colorA, colorB, vUv.z), 1.0);
```

13. Let's choose the colors! Inside addCoolCube()'s if statement add the uniform statement. Choose your own hex colors to interpolate between!

```
if(count == 2) {
      let uniforms = {
            colorB: {type: 'vec3', value: new
                  THREE.Color(0xACB6E5)},
            colorA: {type: 'vec3', value: new
                  THREE.Color(0x74ebd5)}
       };
```

14. Then we need to add the uniforms to `material1` so they will be passed to our fragment shader.

```
material1 =  new THREE.ShaderMaterial({
                  uniforms: uniforms,
                  fragmentShader: fshader,
                  vertexShader: vshader,
                  precision: "mediump" });
```

15. Now test your cube! Do you see some cool color interpolation?

16. Make one additional cube with your own shaders! Do something different. You could try and figure out how I made the rainbow cube in the header!
17. To get checked off for the lab, take a video of all of the cubes spinning. (At a minimum: there should be 4 cubes, two made with Three.js materials and two made with shaders). Like the last lab...add the video to your google drive, go to Sharing > Advanced, set it to anyone at UCSC can view, copy the link there, and paste it into your readme in your CMPM163Labs github under a Lab3 Heading. To get credit for the lab you'll also need to describe how you made each cube's material from left to right in the readme by the video link. Be sure to push all your code to github, too (you will get a 0 if the code isn't

pushed to github)! (check the last lab to remember the four github commands) Then you're all set!