Assignment #5: Projection, Lighting, and Shading in a WebGL Application

Shane Steiner
T00622768

3/20/2021

Problem descriptions :
Get practice with lighting and projections

```
<!-- File name: shadedcube.html-->
<!-- Programmer name: Shane Steiner -->
<!-- Description: html and shaders -->
<!-- Creation Date: 3/20/2021 -->

    <body>
        <canvas id="canvas"></canvas>
        <div id="uiContainer">
            <div id="ui">
                <div id="cubeSize"></div>
                <div id="ConeBaseWidth"></div>
                ---
                <div id="cordShiftX"></div>
                <div id="cordShiftY"></div>
                <div id="cordShiftZ"></div>
                ---
                <div id="rotateX"></div>
                <div id="rotateY"></div>
                <div id="rotateZ"></div>
                ---
                <div></div>
                <input type="color" id="shapeColor" name="head"
value="#e61465">
                <label for="head">color</label>
                <div id="shininess"></div>
                ---
                <div id="projection"></div>
                ---
                <div></div>
                <input type="color" id="ambientColor" name="head"
value="#e61465">
                <label for="head">ambient</label>
                <div id="ambientIntensity"></div>
                ---
                <div></div>
```

```
                <input type="color" id="diffuseColor" name="head"
value="#e61465">
                <label for="head">diffuse</label>
                <div id="diffuseIntensity"></div>
                ---
                <div></div>
                <input type="color" id="specularColor" name="head"
value="#e61465">
                <label for="head">specular</label>
                <div id="specularIntensity"></div>
                ---
                <div id="lightShiftX"></div>
                <div id="lightShiftY"></div>
                <div id="lightShiftZ"></div>
                ---
                <div id="lightsOff"></div>
            </div>
        </div>
        <button id="ButtonX">Rotate X</button>
        <button id="ButtonY">Rotate Y</button>
        <button id="ButtonZ">Rotate Z</button>
        <button id="ButtonT">Toggle Rotation</button>
    </body>




<script id="vertex-shader" type="x-shader/x-vertex">
#version 300 es

in  vec4 aPosition;
in  vec3 aNormal;
out vec4 vColor;

uniform vec4 uAmbientProduct, uDiffuseProduct, uSpecularProduct;
uniform float uAmbientMult, uDiffuseMult, uSpecularMult;

uniform mat4 uModelViewMatrix;
uniform mat4 uProjectionMatrix;
```

```
uniform vec4 uLightPosition;
uniform float uShininess;

void main()
{


    vec3 pos = -(uModelViewMatrix * aPosition).xyz;

    //fixed light postion

    vec3 light = uLightPosition.xyz;
    vec3 L = normalize(light - pos);



    vec3 E = normalize(-pos);
    vec3 H = normalize(L + E);

    vec4 NN = vec4(aNormal,0);

    // Transform vertex normal into eye coordinates

    vec3 N = normalize((uModelViewMatrix*NN).xyz);

    // Compute terms in the illumination equation
    vec4 ambient = uAmbientProduct;

    float Kd = max(dot(L, N), 0.0);
    vec4  diffuse = Kd*uDiffuseProduct;

    float Ks = pow( max(dot(N, H), 0.0), uShininess );
    vec4  specular = Ks * uSpecularProduct;

    if( dot(L, N) < 0.0 ) {
      specular = vec4(0.0, 0.0, 0.0, 1.0);
    }

    gl_Position = uProjectionMatrix * uModelViewMatrix *aPosition;
    vColor = (ambient * uAmbientMult + diffuse *uDiffuseMult +specular
*uSpecularMult);
```

```
    vColor.a = 1.0;
}
</script>

<script id="fragment-shader" type="x-shader/x-fragment">
#version 300 es

precision mediump float;


in vec4 vColor;
out vec4 fColor;


void
main()
{
    fColor = vColor ;
}
</script>

<script src="../assignment5New/resources/webgl-lessons-ui.js"></script>
<script src="../assignment5New/resources/webgl-utils.js"></script>
<script type="text/javascript"
src="../assignment5New/Common/initShaders.js"></script>
<script type="text/javascript"
src="../assignment5New/Common/MVnew.js"></script>
<script type="text/javascript"
src="../assignment5New/shadedCube.js"></script>

<link rel="stylesheet" type="text/css"
href="../assignment5New/resources/maincss.css">
```

```
// <!-- File name: shadedcube.js-->
// <!-- Programmer name: Shane Steiner -->
// <!-- Description: main, render -->
// <!-- Creation Date: 3/20/2021 -->

"use strict";

var gl;
var program;
var shadedCube = function () {

   var canvas;

   var numPositions = 36;

   var positionsArray = [];
   var normalsArray = [];

   var vertices = [
      vec4(-0.5, -0.5, 0.5, 1.0),
      vec4(-0.5, 0.5, 0.5, 1.0),
      vec4(0.5, 0.5, 0.5, 1.0),
      vec4(0.5, -0.5, 0.5, 1.0),
      vec4(-0.5, -0.5, -0.5, 1.0),
      vec4(-0.5, 0.5, -0.5, 1.0),
      vec4(0.5, 0.5, -0.5, 1.0),
      vec4(0.5, -0.5, -0.5, 1.0)
   ];


   var lightPosition = vec4(1.0, 1.0, 1.0, 0.0);
   var lightAmbient = vec4(0.2, 0.2, 0.2, 1.0);
   var lightDiffuse = vec4(1.0, 1.0, 1.0, 1.0);
   var lightSpecular = vec4(1.0, 1.0, 1.0, 1.0);

   var materialAmbient = vec4(1.0, 0.0, 1.0, 1.0);
   var materialDiffuse = vec4(1.0, 0.8, 0.8, 1.0);
   var materialSpecular = vec4(1.0, 0.8, 0.0, 1.0);
   var materialShininess = 100.0;
```

```
var ctm;
var ambientColor, diffuseColor, specularColor;
var modelViewMatrix, projectionMatrix;
var viewerPos;

var xAxis = 0;
var yAxis = 1;
var zAxis = 2;
var axis = 0;
var theta = vec3(0, 0, 0);

var thetaLoc;

var flag = false;

function quad(a, b, c, d) {

   var t1 = subtract(vertices[b], vertices[a]);
   var t2 = subtract(vertices[c], vertices[b]);
   var normal = cross(t1, t2);
   normal = vec3(normal);


   positionsArray.push(vertices[a]);
   normalsArray.push(normal);
   positionsArray.push(vertices[b]);
   normalsArray.push(normal);
   positionsArray.push(vertices[c]);
   normalsArray.push(normal);
   positionsArray.push(vertices[a]);
   normalsArray.push(normal);
   positionsArray.push(vertices[c]);
   normalsArray.push(normal);
   positionsArray.push(vertices[d]);
   normalsArray.push(normal);
}


function colorCube() {
   quad(1, 0, 3, 2);
```

```
      quad(2, 3, 7, 6);
      quad(3, 0, 4, 7);
      quad(6, 5, 1, 2);
      quad(4, 5, 6, 7);
      quad(5, 4, 0, 1);
   }
   //---- ui ----
   //----
   var scaleFactor = 1;
   var cordShiftX = 0;
   var cordShiftY = 0;
   var cordShiftZ = 0;

   var rotateX = 0;
   var rotateY = 0;
   var rotateZ = 0;

   var ambientScale = 1;
   var diffuseScale = 1;
   var specularScale = 1;

   var lightShiftX = 0;
   var lightShiftY = 0;
   var lightShiftZ = 0;

   webglLessonsUI.setupSlider("#cubeSize", { value: (scaleFactor), slide:
updateBase, step: 0.01, min: .5, max: 1.5 });
   function updateBase(event, ui) {
      scaleFactor = ui.value;
   };

   ///cord shift
   webglLessonsUI.setupSlider("#cordShiftX", { value: (cordShiftX), slide:
updateX, step: 0.01, min: -1, max: 1 });
   function updateX(event, ui) {
      cordShiftX = (ui.value);
   }
   webglLessonsUI.setupSlider("#cordShiftY", { value: (cordShiftY), slide:
updateY, step: 0.01, min: -1, max: 1 });
   function updateY(event, ui) {
```

```
      cordShiftY = (ui.value);
   }
   webglLessonsUI.setupSlider("#cordShiftZ", { value: (cordShiftZ), slide:
updateZ, step: 0.01, min: -10, max: 0 });
   function updateZ(event, ui) {
      cordShiftZ = (ui.value);
   }

   //rotate
   webglLessonsUI.setupSlider("#rotateX", { value: (rotateX), slide:
updateRotateX, step: 0.02, min: 0, max: 180 });
   function updateRotateX(event, ui) {
      rotateX = (ui.value);
   }
   webglLessonsUI.setupSlider("#rotateY", { value: (rotateY), slide:
updateRotateY, step: 0.02, min: 0, max: 180 });
   function updateRotateY(event, ui) {
      rotateY = (ui.value);
   }
   webglLessonsUI.setupSlider("#rotateZ", { value: (rotateZ), slide:
updateRotateZ, step: 0.02, min: 0, max: 180 });
   function updateRotateZ(event, ui) {
      rotateZ = (ui.value);
   }
   //shininess
   webglLessonsUI.setupSlider("#shininess", { value: (rotateZ), slide:
updateshine, step: 0.02, min: 5, max: 100 });
   function updateshine(event, ui) {
      materialShininess = (ui.value);
   }
   //projection
   webglLessonsUI.setupSlider("#projection", { value: (rotateZ), slide:
updateProjection, step: 1, min: 0, max: 1, name: "Projection 1=ortho
0=perspective" });
   function updateProjection(event, ui) {
      if (ui.value == 0) {
         projectionMatrix = ProjectionPerspective(projectionMatrix);
      }
      else {
         projectionMatrix = ProjectionOrtho(projectionMatrix);
```

```
      }
   }

   //ambient intensity
   webglLessonsUI.setupSlider("#ambientIntensity", { value: (rotateZ),
slide: updateAmbientIntensity, step: 0.02, min: 0, max: 1.2 });
   function updateAmbientIntensity(event, ui) {
      ambientScale = (ui.value);
   }
   //ambient intensity
   webglLessonsUI.setupSlider("#diffuseIntensity", { value: (rotateZ),
slide: updateDiffuseIntensity, step: 0.02, min: 0, max: 1.2 });
   function updateDiffuseIntensity(event, ui) {
      diffuseScale = (ui.value);
   }
   //ambient intensity
   webglLessonsUI.setupSlider("#specularIntensity", { value: (rotateZ),
slide: updateSpecularIntensity, step: 0.02, min: 0, max: 1.2 });
   function updateSpecularIntensity(event, ui) {
      specularScale = (ui.value);
   }

   ///light shift
   webglLessonsUI.setupSlider("#lightShiftX", { value: (lightShiftX),
slide: updateLightX, step: 0.01, min: -1, max: 1 });
   function updateLightX(event, ui) {
      lightShiftX = (ui.value);
      lightPosition[0] = lightShiftX;
   }
   webglLessonsUI.setupSlider("#lightShiftY", { value: (lightShiftY),
slide: updateLightY, step: 0.01, min: -1, max: 1 });
   function updateLightY(event, ui) {
      lightShiftY = (ui.value);
      lightPosition[1] = lightShiftY;
   }
   webglLessonsUI.setupSlider("#lightShiftZ", { value: (lightShiftZ),
slide: updateLightZ, step: 0.01, min: 0, max: 10 });
   function updateLightZ(event, ui) {
      lightShiftZ = (ui.value);
      lightPosition[2] = lightShiftZ;
```

```
}

var colorPicker = document.querySelector("#shapeColor");
colorPicker.addEventListener("input", updateFirst, false);
function updateFirst(event) {
   materialDiffuse[0] = hexToRgb(event.target.value).r / 255;
   materialDiffuse[1] = hexToRgb(event.target.value).g / 255;
   materialDiffuse[2] = hexToRgb(event.target.value).b / 255;

   materialAmbient[0] = hexToRgb(event.target.value).r / 255;
   materialAmbient[1] = hexToRgb(event.target.value).g / 255;
   materialAmbient[2] = hexToRgb(event.target.value).b / 255;

   materialShininess[0] = hexToRgb(event.target.value).r / 255;
   materialShininess[1] = hexToRgb(event.target.value).g / 255;
   materialShininess[2] = hexToRgb(event.target.value).b / 255;
}
//ambient picker
var colorPicker = document.querySelector("#ambientColor");
colorPicker.addEventListener("input", updateAmbient, false);
function updateAmbient(event) {
   lightAmbient[0] = hexToRgb(event.target.value).r / 255;
   lightAmbient[1] = hexToRgb(event.target.value).g / 255;
   lightAmbient[2] = hexToRgb(event.target.value).b / 255;
}
//diffuse picker
var colorPicker = document.querySelector("#diffuseColor");
colorPicker.addEventListener("input", updateDiffuse, false);
function updateDiffuse(event) {
   lightDiffuse[0] = hexToRgb(event.target.value).r / 255;
   lightDiffuse[1] = hexToRgb(event.target.value).g / 255;
   lightDiffuse[2] = hexToRgb(event.target.value).b / 255;
}
//specular picker
var colorPicker = document.querySelector("#specularColor");
colorPicker.addEventListener("input", updateSpecular, false);
function updateSpecular(event) {
   lightSpecular[0] = hexToRgb(event.target.value).r / 255;
   lightSpecular[1] = hexToRgb(event.target.value).g / 255;
   lightSpecular[2] = hexToRgb(event.target.value).b / 255;
```

```
   }

   //toggle lights on/ off
   var lightPosCopy;
   webglLessonsUI.setupSlider("#lightsOff", { value: (lightShiftZ), slide:
toggleLight, step: 1, min: 0, max: 1 });
   function toggleLight(event, ui) {
      if(ui.value == 1)
      {
        diffuseScale = 0;
        specularScale =0;
        ambientScale = 0;
      }
      else
      {
         diffuseScale = 1;
        specularScale =1;
        ambientScale = 1;
      }


   }

   //----

   window.onload = function init() {
      canvas = document.getElementById("canvas");

      gl = canvas.getContext('webgl2');
      if (!gl) alert("WebGL 2.0 isn't available");

      webglUtils.resizeCanvasToDisplaySize(gl.canvas);

      // Tell WebGL how to convert from clip space to pixels
      gl.viewport(0, 0, gl.canvas.height, gl.canvas.height);

      //gl.viewport(0, 0, canvas.width, canvas.height);
      gl.clearColor(1.0, 1.0, 1.0, 1.0);

      gl.enable(gl.DEPTH_TEST);
```

```
      //
      //  Load shaders and initialize attribute buffers
      //
      program = initShaders(gl, "vertex-shader", "fragment-shader");
      gl.useProgram(program);

      colorCube();

      var nBuffer = gl.createBuffer();
      gl.bindBuffer(gl.ARRAY_BUFFER, nBuffer);
      gl.bufferData(gl.ARRAY_BUFFER, flatten(normalsArray),
gl.STATIC_DRAW);

      var normalLoc = gl.getAttribLocation(program, "aNormal");
      gl.vertexAttribPointer(normalLoc, 3, gl.FLOAT, false, 0, 0);
      gl.enableVertexAttribArray(normalLoc);

      var vBuffer = gl.createBuffer();
      gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);
      gl.bufferData(gl.ARRAY_BUFFER, flatten(positionsArray),
gl.STATIC_DRAW);

      var positionLoc = gl.getAttribLocation(program, "aPosition");
      gl.vertexAttribPointer(positionLoc, 4, gl.FLOAT, false, 0, 0);
      gl.enableVertexAttribArray(positionLoc);

      thetaLoc = gl.getUniformLocation(program, "theta");

      viewerPos = vec3(0.0, 0.0, -20.0);


      projectionMatrix = ProjectionPerspective(projectionMatrix);


      document.getElementById("ButtonX").onclick = function () { axis =
xAxis; };
      document.getElementById("ButtonY").onclick = function () { axis =
yAxis; };
```

```
      document.getElementById("ButtonZ").onclick = function () { axis =
zAxis; };
      document.getElementById("ButtonT").onclick = function () { flag =
!flag; };



      render();
   }

   var render = function () {


      var scaleMat = mat4(scaleFactor, 0, 0, 0,
         0, scaleFactor, 0, 0,
         0, 0, scaleFactor, 0,
         0, 0, 0, 1);
      var translationMat = mat4(1, 0, 0, cordShiftX,
         0, 1, 0, cordShiftY,
         0, 0, 1, cordShiftZ,
         0, 0, 0, 1);


      gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

      if (flag) theta[axis] += 1.0;

      modelViewMatrix = mat4();
      modelViewMatrix = mult(modelViewMatrix, scaleMat);
      modelViewMatrix = mult(modelViewMatrix, translationMat);
      modelViewMatrix = mult(modelViewMatrix, rotate(theta[xAxis] +
rotateX, vec3(1, 0, 0)));
      modelViewMatrix = mult(modelViewMatrix, rotate(theta[yAxis] +
rotateY, vec3(0, 1, 0)));
      modelViewMatrix = mult(modelViewMatrix, rotate(theta[zAxis] +
rotateZ, vec3(0, 0, 1)));



      gl.uniformMatrix4fv(gl.getUniformLocation(program,
```

```
         "uModelViewMatrix"), false, flatten(modelViewMatrix));
         //diffuse
         gl.uniform1f(gl.getUniformLocation(program,
            "uDiffuseMult"), diffuseScale);
      var diffuseProduct = mult(lightDiffuse, materialDiffuse);
      gl.uniform4fv(gl.getUniformLocation(program, "uDiffuseProduct"),
      diffuseProduct);
         //ambient
      gl.uniform1f(gl.getUniformLocation(program,
         "uAmbientMult"), ambientScale);
         var ambientProduct = mult(lightAmbient, materialAmbient);
         gl.uniform4fv(gl.getUniformLocation(program, "uAmbientProduct"),
         ambientProduct);
         //specular
         gl.uniform1f(gl.getUniformLocation(program,
            "uSpecularMult"), specularScale);
         var specularProduct = mult(lightSpecular, materialSpecular);
      gl.uniform4fv(gl.getUniformLocation(program, "uSpecularProduct"),
         specularProduct);
         //light location
      gl.uniform4fv(gl.getUniformLocation(program, "uLightPosition"),
         lightPosition);

      gl.uniform1f(gl.getUniformLocation(program,
         "uShininess"), materialShininess);

      gl.drawArrays(gl.TRIANGLES, 0, numPositions);


      requestAnimationFrame(render);
   }

}

//helpers
function hexToRgb(hex) {
   var result = /^#?([a-f\d]{2})([a-f\d]{2})([a-f\d]{2})$/i.exec(hex);
   return result ? {
      r: parseInt(result[1], 16),
      g: parseInt(result[2], 16),
```

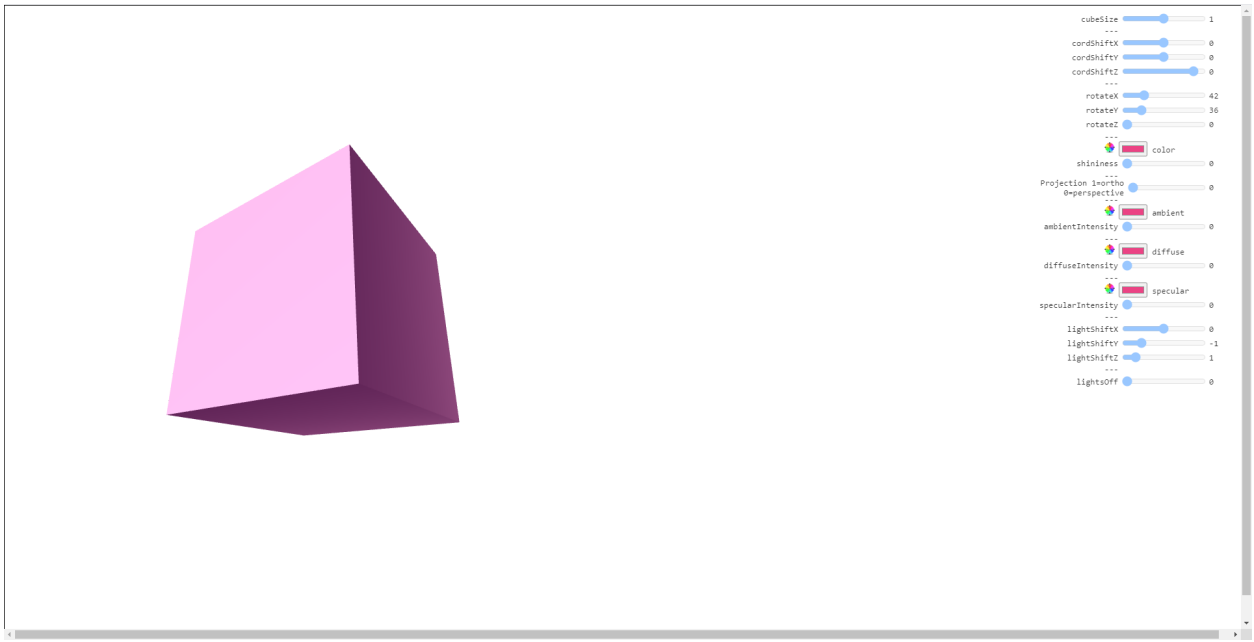
```
      b: parseInt(result[3], 16)
   } : null;
}

//change projection
//
var translationMat2 = mat4(1, 0, 0, 0,
   0, 1, 0, 0,
   0, 0, 1, -2,
   0, 0, 0, 1);
function ProjectionOrtho(projectionMatrix) {
   projectionMatrix = ortho(-1, 1, -1, 1, -100, 100);

   projectionMatrix = mult(projectionMatrix, translationMat2);
   gl.uniformMatrix4fv(gl.getUniformLocation(program,
"uProjectionMatrix"),
      false, flatten(projectionMatrix));
}
function ProjectionPerspective(projectionMatrix) {
   var fovy = 10000, aspect = 1, near = 0.01, far = 100;
   projectionMatrix = perspective(fovy, aspect, near, far);

   projectionMatrix = mult(projectionMatrix, translationMat2);
   gl.uniformMatrix4fv(gl.getUniformLocation(program,
"uProjectionMatrix"),
      false, flatten(projectionMatrix));
}
//

shadedCube();
```


cubeSize 1
---
cordShiftX 0
cordShiftY 0
cordShiftZ 0
---
rotateX 42
rotateY 36
rotateZ 0
---
color
shininess 0

Projection 1=ortho 0=perspective 0
---
ambient
ambientIntensity 0
---
diffuse
diffuseIntensity 0
---
specular
specularIntensity 0
---
lightShiftX 0
lightShiftY -1
lightShiftZ 1
---
lightsOff 0


cubeSize 1
---
cordShiftX 0
cordShiftY 0
cordShiftZ 0
---
rotateX 42
rotateY 106
rotateZ 85
---
color
shininess 15

Projection 1=ortho 0=perspective 0
---
ambient
ambientIntensity 0
---
diffuse
diffuseIntensity 1
---
specular
specularIntensity 1
---
lightShiftX 0
lightShiftY -1
lightShiftZ 1
---
lightsOff 0


cubeSize 1
---
cordShiftX 0
cordShiftY 0
cordShiftZ 0
---
rotateX 99
rotateY 45
rotateZ 85
---
color
shininess 15

Projection 1=ortho 0=perspective 0
---
ambient
ambientIntensity 0
---
diffuse
diffuseIntensity 1
---
specular
specularIntensity 1
---
lightShiftX 0
lightShiftY -1
lightShiftZ 1
---
lightsOff 1