CS-733 Final Exam
December 20, 2022, 14:00 – 17:00, CL313
D. Hepting

This is a closed book exam. You must maintain the confidentiality of your examination; do not provide any opportunity for others to copy any of your work. Electronic devices are NOT permitted during the exam. Please turn off and put away all cell phones and other electronic devices during the exam period.

ANSWER ALL QUESTIONS. All answers must be written on this exam in the space provided. You have 180 minutes to complete the exam. Please plan your answers, favour quality over quantity, do not exceed the space provided, and do your best to write legibly. QUESTIONS ARE ON BOTH SIDES OF THE PAPER. This exam contributes 40 percent towards your final grade.

Name (printed): _____

Student Number: _____

Signature: _____

***

Q1. (4 marks) What are the different senses of pixel in different uses?

Q2. (8 marks) Starting from a black screen, describe or sketch how to get colours. white, using the 3 primitive colours? Describe how to get colours when printing onto a sheet of paper.

Q3. (6 marks) Describe the following concepts: continuous, discrete, sampling, and quantization. What is the relationship amongst them? Are they used in WebGL?

Q4. (4 marks) How do a you represent a point in homogeneous coordinates? Why are homogeneous coordinates important in computer graphics?

Q5. (8 marks) Describe the content and function of roboticArm.html and roboticArm.js (both provided) as a sample of a program from the text.

Q6. (2 marks) What is an advantage of WebGL when it comes to interaction using web standards?

Q7. (6 marks) Briefly describe how interpolation is used in Gouraud and Phong shading algorithms. Relate the algorithms to vertex and fragment shaders.

Q8. (4 marks) Describe how the interaction of a light with other elements in a scene is calculated (you may use the provided source code for reference).

Q9. (2 marks) Why is the flatten function required in code from the textbook?

Q10. (4 marks) What kinds of transformations are needed to express any scene to be rendered in terms of clipping coordinates [the cube from (-1,-1,-1) to (1,1,1)]?

Q11. (4 marks) What is the purpose of a z-buffer? Briefly explain how it is used.

Q12. (8 marks) Describe bump mapping, texture mapping, and environment mapping. How are they similar and how are they different?

Q13. (8 marks) Is aliasing a problem with using images as textures? What is it and what are some possible solutions?

Q14. (6 marks) What is an advantage to creating models by instancing a single primitive?

Q15. (16 marks) Document the code in roboticArm.html by indicating line numbers and writing comments about them in the space below.

Q16.. (16 marks) Document the code in roboticArm.js by indicating line numbers and writing comments about them in the space below.

```
<!DOCTYPE html>
<html>
  <head>
    <script id="vertex-shader-1" type="x-shader/x-vertex">
    #version 300 es

    // per vertex lighting
    in vec4 aPosition;
    in vec3 aNormal;
    out vec4 vColor;

    uniform vec3 theta;

    uniform vec4 ambientProduct, diffuseProduct, specularProduct;
    uniform mat4 modelViewMatrix;
    uniform mat4 projectionMatrix;
    uniform vec4 lightPosition;
    uniform float shininess;

    void
    main()
    {
      vec3 pos = (modelViewMatrix * aPosition).xyz;

      // fixed light postion
      vec3 light = lightPosition.xyz;
      vec3 L = normalize(light - pos);
      vec3 E = normalize(pos);
      vec3 H = normalize(L + E);
      vec4 NN = vec4(normalize(aNormal), 0);

      // Transform vertex normal into eye coordinates
      vec3 N = normalize((modelViewMatrix * NN).xyz);

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

      float Kd = max(dot(L, N), 0.0);
      vec4 diffuse = Kd * diffuseProduct;
      float Ks = pow(max(dot(N, H), 0.0), shininess);
      vec4 specular = Ks * specularProduct;
      if (dot(L, N) < 0.0) {
           specular = vec4(0.0, 0.0, 0.0, 1.0);
      }

      gl_Position = projectionMatrix * modelViewMatrix * aPosition;

      vColor = ambient + diffuse + specular;
      vColor.a = 1.0;
    }
    </script>
    <script id="fragment-shader-1" type="x-shader/x-fragment">
    #version 300 es

    // pass through fragment shader
    precision mediump float;

    in vec4 vColor;
    out vec4 fColor;

    void
    main()
    {
        fColor = vColor;
    }
    </script>
    <script type="text/javascript" src="../Common/initShaders.js"></script>
    <script type="text/javascript" src="../Common/MVnew.js"></script>
    <script type="text/javascript" src="../Common/geometry.js"></script>
    <script type="text/javascript" src="roboticArm.js"></script>
  </head>
  <body>
    <table>
      <tr>
        <td>
          <div>
            <button id="toggle">
              Toggle Rotation
```

```
            </button>
          </div>
          <br/>
          <div>
            Body angle -180
            <input id="slider1" type="range"
            min="-180" max="180" step="1" value="0"
            />
            180
          </div>
          <br/>
          <div>
            Lower arm angle -90
            <input id="slider2" type="range"
            min="-90" max="90" step="1" value="0"
            />
            90
          </div>
          <br/>
          <div>
            Upper arm angle -90
            <input id="slider3" type="range"
            min="-90" max="90" step="1" value="0"
            />
            90
          </div>
          <br/>
        </td>
        <td>
          <canvas id="gl-canvas" width="512" height="512">
            Your browser does not support the HTML5 canvas element
          </canvas>
        </td>
      </tr>
    </table>
  </body>
</html>
```

```
'use strict'

let canvas, gl, program
let modelViewMatrix, modelViewMatrixLoc, projectionMatrix
let points = []
let normals = []
let angle = 0
let angleSine = 0
let rotateOn = true

const NumVertices = 36 // (6 faces)(2 triangles/face)(3 vertices/triangle)

// Parameters controlling the size of the Robot's arm
const BASE_HEIGHT = 2.0
const BASE_WIDTH = 5.0
const LOWER_ARM_HEIGHT = 5.0
const LOWER_ARM_WIDTH = 0.5
const UPPER_ARM_HEIGHT = 5.0
const UPPER_ARM_WIDTH = 0.5

// Array of rotation angles (in degrees) for each rotation axis
const theta = [0, 0, 0]
const Base = 0
const LowerArm = 1
const UpperArm = 2

window.onload = function init () {
  canvas = document.getElementById('gl-canvas')
  gl = canvas.getContext('webgl2')
  if (!gl) {
    window.alert('WebGL 2.0 is not available')
  }
  gl.viewport(0, 0, canvas.width, canvas.height)
  gl.clearColor(0.9, 0.9, 0.9, 1.0)
  gl.enable(gl.DEPTH_TEST)

  // from Common/geometry.js
  const myCube = cube()
  const myMaterial = goldMaterial()
  const myLight = light0()

  // Load shaders and use the resulting shader program
  program = initShaders(gl, 'vertex-shader-1', 'fragment-shader-1')
  gl.useProgram(program)

  // Create and initialize buffer objects
  points = myCube.TriangleVertices
  normals = myCube.TriangleNormals

  const vBuffer = gl.createBuffer()
  gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer)
  gl.bufferData(gl.ARRAY_BUFFER, flatten(points), gl.STATIC_DRAW)
  const positionLoc = gl.getAttribLocation(program, 'aPosition')
  gl.vertexAttribPointer(positionLoc, 4, gl.FLOAT, false, 0, 0)
  gl.enableVertexAttribArray(positionLoc)

  const nBuffer = gl.createBuffer()
  gl.bindBuffer(gl.ARRAY_BUFFER, nBuffer)
  gl.bufferData(gl.ARRAY_BUFFER, flatten(normals), gl.STATIC_DRAW)
  const normalLoc = gl.getAttribLocation(program, 'aNormal')
  gl.vertexAttribPointer(normalLoc, 3, gl.FLOAT, false, 0, 0)
  gl.enableVertexAttribArray(normalLoc)

  // products of material and light properties
  const ambientProduct = mult(myLight.lightAmbient,
      myMaterial.materialAmbient)
  const diffuseProduct = mult(myLight.lightDiffuse,
      myMaterial.materialDiffuse)
  const specularProduct = mult(myLight.lightSpecular,
      myMaterial.materialSpecular)

  document.getElementById('toggle').onclick = function (event) {
    rotateOn = !rotateOn
  }
  document.getElementById('slider1').onchange = function (event) {
    theta[Base] = event.target.value
  }
  document.getElementById('slider2').onchange = function (event) {
```

```
    theta[1] = event.target.value
  }
  document.getElementById('slider3').onchange = function (event) {
    theta[2] = event.target.value
  }

  modelViewMatrixLoc = gl.getUniformLocation(program, 'modelViewMatrix')
  projectionMatrix = ortho(-10, 10, -10, 10, -10, 10)
  gl.uniformMatrix4fv(gl.getUniformLocation(program, 'projectionMatrix'),
false, flatten(projectionMatrix))
  gl.uniform4fv(gl.getUniformLocation(program, 'ambientProduct'),
    flatten(ambientProduct))
  gl.uniform4fv(gl.getUniformLocation(program, 'diffuseProduct'),
    flatten(diffuseProduct))
  gl.uniform4fv(gl.getUniformLocation(program, 'specularProduct'),
    flatten(specularProduct))
  gl.uniform4fv(gl.getUniformLocation(program, 'lightPosition'),
    flatten(myLight.lightPosition))
  gl.uniform1f(gl.getUniformLocation(program,
    'shininess'), myMaterial.materialShininess)
  gl.uniformMatrix4fv(gl.getUniformLocation(program, 'projectionMatrix'),
    false, flatten(projectionMatrix))
  render()
}

function base () {
  const s = scale(BASE_WIDTH, BASE_HEIGHT, BASE_WIDTH)
  const instanceMatrix = mult(translate(0.0, 0.5 * BASE_HEIGHT, 0.0), s)
  const t = mult(modelViewMatrix, instanceMatrix)
  gl.uniformMatrix4fv(modelViewMatrixLoc, false, flatten(t))
  gl.drawArrays(gl.TRIANGLES, 0, NumVertices)
}

function upperArm () {
  const s = scale(UPPER_ARM_WIDTH, UPPER_ARM_HEIGHT, UPPER_ARM_WIDTH)
  const instanceMatrix = mult(translate(0.0, 0.5 * UPPER_ARM_HEIGHT, 0.0), s)
  const t = mult(modelViewMatrix, instanceMatrix)
  gl.uniformMatrix4fv(modelViewMatrixLoc, false, flatten(t))
  gl.drawArrays(gl.TRIANGLES, 0, NumVertices)
}

function lowerArm () {
  const s = scale(LOWER_ARM_WIDTH, LOWER_ARM_HEIGHT, LOWER_ARM_WIDTH)
  const instanceMatrix = mult(translate(0.0, 0.5 * LOWER_ARM_HEIGHT, 0.0), s)
  const t = mult(modelViewMatrix, instanceMatrix)
  gl.uniformMatrix4fv(modelViewMatrixLoc, false, flatten(t))
  gl.drawArrays(gl.TRIANGLES, 0, NumVertices)
}

function render () {
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT)
  if (rotateOn) {
    angle = ((angle + 1.0) % 360)
    angleSine = Math.sin(angle * Math.PI / 180)
    theta[Base] = angleSine * 180
    theta[LowerArm] = angleSine * 60
    theta[UpperArm] = angleSine * 30
  }
  // render the base
  document.getElementById('slider1').value = theta[Base]
  modelViewMatrix = rotate(theta[Base], vec3(0, 1, 0))
  base()
  // render the lowerArm
  document.getElementById('slider2').value = theta[LowerArm]
  modelViewMatrix = mult(modelViewMatrix, translate(0.0, BASE_HEIGHT, 0.0))
  modelViewMatrix = mult(modelViewMatrix, rotate(theta[LowerArm], vec3(0, 0,
1)))
  lowerArm()
  // render the UpperArm
  document.getElementById('slider3').value = theta[UpperArm]
  modelViewMatrix = mult(modelViewMatrix, translate(0.0, LOWER_ARM_HEIGHT,
0.0))
  modelViewMatrix = mult(modelViewMatrix, rotate(theta[UpperArm], vec3(0, 0,
1)))
  upperArm()

  window.requestAnimationFrame(render)
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <script id="vertex-shader-1" type="x-shader/x-vertex">
    #version 300 es

    // per vertex lighting
    in vec4 aPosition;
    in vec3 aNormal;
    out vec4 vColor;

    uniform vec3 theta;

    uniform vec4 ambientProduct, diffuseProduct, specularProduct;
    uniform mat4 modelViewMatrix;
    uniform mat4 projectionMatrix;
    uniform vec4 lightPosition;
    uniform float shininess;

    void
    main()
    {
      vec3 pos = (modelViewMatrix * aPosition).xyz;

      // fixed light postion
      vec3 light = lightPosition.xyz;
      vec3 L = normalize(light - pos);
      vec3 E = normalize(pos);
      vec3 H = normalize(L + E);
      vec4 NN = vec4(normalize(aNormal), 0);

      // Transform vertex normal into eye coordinates
      vec3 N = normalize((modelViewMatrix * NN).xyz);

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

      float Kd = max(dot(L, N), 0.0);
      vec4 diffuse = Kd * diffuseProduct;
      float Ks = pow(max(dot(N, H), 0.0), shininess);
      vec4 specular = Ks * specularProduct;
      if (dot(L, N) < 0.0) {
           specular = vec4(0.0, 0.0, 0.0, 1.0);
      }

      gl_Position = projectionMatrix * modelViewMatrix * aPosition;

      vColor = ambient + diffuse + specular;
      vColor.a = 1.0;
    }
    </script>
    <script id="fragment-shader-1" type="x-shader/x-fragment">
    #version 300 es

    // pass through fragment shader
    precision mediump float;

    in vec4 vColor;
    out vec4 fColor;

    void
    main()
    {
        fColor = vColor;
    }
    </script>
    <script type="text/javascript" src="../Common/initShaders.js"></script>
    <script type="text/javascript" src="../Common/MVnew.js"></script>
    <script type="text/javascript" src="../Common/geometry.js"></script>
    <script type="text/javascript" src="roboticArm.js"></script>
  </head>
  <body>
    <table>
      <tr>
        <td>
          <div>
            <button id="toggle">
              Toggle Rotation
```

```
            </button>
          </div>
          <br/>
          <div>
            Body angle -180
            <input id="slider1" type="range"
            min="-180" max="180" step="1" value="0"
            />
            180
          </div>
          <br/>
          <div>
            Lower arm angle -90
            <input id="slider2" type="range"
            min="-90" max="90" step="1" value="0"
            />
            90
          </div>
          <br/>
          <div>
            Upper arm angle -90
            <input id="slider3" type="range"
            min="-90" max="90" step="1" value="0"
            />
            90
          </div>
          <br/>
        </td>
        <td>
          <canvas id="gl-canvas" width="512" height="512">
            Your browser does not support the HTML5 canvas element
          </canvas>
        </td>
      </tr>
    </table>
  </body>
</html>
```