# CS-315+733 Final Exam
# December 9, 2024, 14:00 – 17:00, CL112
# 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 30 percent towards your final grade. The exam is marked out of 102. Referenced code is provided.

Name (printed): _____

Student Number (ID): _____

Signature: _____

***

Q1. (8 marks) Describe and explain the relationship between the various frames, or coordinate systems, that we have discussed this semester.

Q2. (4 marks) From RGB, describe the intuition behind the HSV (Hue Saturation Value) colour model.

Q3. (4 marks) What are the tradeoffs between bitmap and vector image formats?

Q4. (4 marks) What are the 2 things needed (1 in javascript and 1 in GLSL shader code) to access the features of WebGL2?

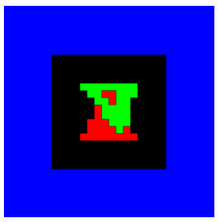
Q5. (8 marks) Describe an implementation of a fly-by of a model where the eye and lookAt points both change but the relationship between them remains constant.

Q6. (4 marks) When can quadrilaterals be used unambiguously in models? How can you test for this condition?

Q7. (6 marks) Compare and contrast bump mapping, texture mapping, and environment mapping.

Q8. (4 marks) Compare and contrast per-vertex colours and uniform colours.

Q9. (12 marks) Describe, with reference to the code, each use of framebuffer objects in the creation of this image from `render-FB`? How would you modify the code to cover everything with texture, repeated over and over?



**render-FB.html**

```
<!DOCTYPE html>
<html>

<script id="vertex-shader1" type="x-shader/x-vertex">
#version 300 es

in vec4 aPosition;

void main()
{
    gl_Position = aPosition;
}
</script>

<script id="vertex-shader2" type="x-shader/x-vertex">
#version 300 es
in vec4 aPosition;
in vec2 aTexCoord;
out vec2 vTexCoord;

void main()
{
  gl_Position = aPosition;
  vTexCoord = aTexCoord;
}
</script>

<script id="fragment-shader1" type="x-shader/x-fragment">
#version 300 es

precision mediump float;

out vec4 fColor;
uniform vec4 uColor;

void
main()
{
    fColor = uColor;
}
</script>
<script id="fragment-shader2" type="x-shader/x-fragment">
#version 300 es
precision mediump float;

in vec2 vTexCoord;
out vec4 fColor;

uniform sampler2D uTextureMap;
```

```
void main()
{
    fColor = texture(uTextureMap, vTexCoord);
}
</script>

<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MVnew.js"></script>
<script type="text/javascript" src="render-FB.js"></script>

<body>
<canvas id="gl-canvas" width="512" height="512">
    Your browser does not support the HTML5 canvas element
</canvas>
</body>
</html>
```

**Render-FB.js**

```
'use strict';

let gl;

const texSize = 16;
// background square texture coordinates that match the
// TRIANGLE vertices (6 for 2 triangles) that make up the
// background square
const texCoord = [
  vec2(0, 0),
  vec2(0, 1),
  vec2(1, 1),
  vec2(1, 1),
  vec2(1, 0),
  vec2(0, 0)];
// quad vertices
const vertices = [
  vec2(-0.5, -0.5),
  vec2(-0.5, 0.5),
  vec2(0.5, 0.5),
  vec2(0.5, 0.5),
  vec2(0.5, -0.5),
  vec2(-0.5, -0.5)];

// vertices for 2 intersecting triangles to be drawn in texture
const positionsArray = [
  vec4(-0.5, -0.5, 0, 1),
  vec4(0.5, -0.5, 0, 1),
  vec4(0.0, 0.5, 0, 1),
  vec4(-0.5, 0.5, 0, 1),
  vec4(0.5, 0.5, 0.25, 1),
  vec4(0.25, -0.5, -0.25, 1)];
```

```
let createTT; let useTT;
let texture;

let framebuffer;
let depthbuffer;

window.onload = function init() {
  const canvas = document.getElementById('gl-canvas');

  gl = canvas.getContext('webgl2');
  if (!gl) {
    alert('WebGL 2.0 is not available');
  }
  gl.activeTexture(gl.TEXTURE0);

  // Create an empty texture
  texture = gl.createTexture();
  gl.bindTexture(gl.TEXTURE_2D, texture);
  gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA,
      texSize, texSize, 0, gl.RGBA, gl.UNSIGNED_BYTE, null);
  gl.texParameteri(gl.TEXTURE_2D,
      gl.TEXTURE_MIN_FILTER, gl.NEAREST_MIPMAP_LINEAR);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
  // Allocate a frame buffer object
  framebuffer = gl.createFramebuffer();
  gl.bindFramebuffer(gl.FRAMEBUFFER, framebuffer);
  // Attach colour buffer
  gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0,
      gl.TEXTURE_2D, texture, 0);
  // Attach depth buffer
  depthbuffer = gl.createRenderbuffer();
  gl.bindRenderbuffer(gl.RENDERBUFFER, depthbuffer);
  gl.renderbufferStorage(gl.RENDERBUFFER, gl.DEPTH_COMPONENT16,
      texSize, texSize);
  gl.framebufferRenderbuffer(gl.FRAMEBUFFER, gl.DEPTH_ATTACHMENT,
      gl.RENDERBUFFER, depthbuffer);
  // check for completeness
  const status = gl.checkFramebufferStatus(gl.FRAMEBUFFER);
  if (status != gl.FRAMEBUFFER_COMPLETE) {
    alert('Frame buffer not complete');
  }
  //
  //  Load shaders and initialize attribute buffers
  //
  createTT = initShaders(gl, 'vertex-shader1', 'fragment-shader1');
  useTT = initShaders(gl, 'vertex-shader2', 'fragment-shader2');
```

```
// create the triangles texture
gl.useProgram(createTT);
gl.enable(gl.DEPTH_TEST);

// Create and initialize a buffer object with triangle vertices
const buffer1 = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, buffer1);
gl.bufferData(gl.ARRAY_BUFFER,
    flatten(positionsArray), gl.STATIC_DRAW);

// Initialize the vertex position attribute from the vertex shader
let positionLoc = gl.getAttribLocation(createTT, 'aPosition');
gl.vertexAttribPointer(positionLoc, 4, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(positionLoc);

// Render 2 triangle intersecting triangles
gl.viewport(0, 0, texSize, texSize);
gl.clearColor(0.5, 0.5, 0.5, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT );
gl.uniform4f(gl.getUniformLocation(createTT, 'uColor'), 0.9, 0.0, 0.0, 1.0);
gl.drawArrays(gl.TRIANGLES, 0, 3);
gl.uniform4f(gl.getUniformLocation(createTT, 'uColor'), 0.0, 0.9, 0.0, 1.0);
gl.drawArrays(gl.TRIANGLES, 3, 3);
gl.generateMipmap(gl.TEXTURE_2D);

// Bind to window system frame buffer
gl.bindFramebuffer(gl.FRAMEBUFFER, null);
gl.bindRenderbuffer(gl.RENDERBUFFER, null);
gl.disableVertexAttribArray(positionLoc);

// use the triangles texture just created
gl.useProgram(useTT);

// send data to GPU for normal render
const buffer2 = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, buffer2);
gl.bufferData(gl.ARRAY_BUFFER, flatten(vertices), gl.STATIC_DRAW);

positionLoc = gl.getAttribLocation(useTT, 'aPosition');
gl.vertexAttribPointer( positionLoc, 2, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(positionLoc);

const buffer3 = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, buffer3);
gl.bufferData( gl.ARRAY_BUFFER,
    flatten(texCoord), gl.STATIC_DRAW);
const texCoordLoc = gl.getAttribLocation(useTT, 'aTexCoord');
gl.vertexAttribPointer(texCoordLoc, 2, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(texCoordLoc);
```

```
  gl.uniform1i(gl.getUniformLocation(useTT, 'uTextureMap'), 0);
  gl.viewport(0, 0, 512, 512);

  render();
};

function render() {
  gl.clearColor(0.0, 0.0, 1.0, 1.0);
  gl.clear(gl.COLOR_BUFFER_BIT);

  // render background square with texture
  gl.drawArrays(gl.TRIANGLES, 0, 6);
}
```

Q10. (6 marks) What are the tradeoffs when choosing between Gouraud shading and Phong shading?

Q11. (4 marks) What are differences between parallel and perspective projections? Illustrate with a line or two of code how the projection is applied in a program.

Q12. (4 marks) Describe how an offscreen buffer can be used to implement picking. Why not just use the onscreen buffer?

Q13. (4 marks) What are the differences between directional lights and point lights when computing illumination?

Q14. (4 marks) What is the difference between vertex shaders and fragment shaders? Is it useful to have more than one of each?

Q15. (6 marks) Describe the transformation required to centre the rotation of a square at the point (1, 2). Write a code snippet to perform the matrix multiplication needed, using the MVnew.js library from the textbook. Is the order of multiplication important?

Q16. (4 marks) What are the two things we need to define a coordinate system (frame)?

Q17. (4 marks) What is the impact of the halfway vector on illumination calculations?

Q18. (12 marks) Use a hierarchical model to describe the robot in roboticArm.js. How would it change if a second arm (with a claw) was added? Is there any impact if the order in which the robot parts appear inside the code is changed?

**roboticArm.js**

```
'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)
}
```