CSCI 4250/5250   Take Home Test 2 (100 pts)   **Name __________________**

You are required to type your answers for these questions.

Open Book & Notes – You are on the honor system and you are to do this exam with no help from any other person. When you type your name below, you are indicating that you have adhered to these restrictions. Turn in this page with your answers to the questions. Turn in your typed answers in pdf file to the D2L Dropbox named "Test 2".

I, ____, worked all of the problems on this test completely on my own without any assistance from any other person. My only resources were the IR textbook, my notes, and Internet sources.

1. (10 pts) Fill in the blanks:

a. The ____view___ matrix transforms the object drawn from the world coordinate into the camera coordinate.

b. The ________modelview__ matrix is saved and restored using Push and Pop operations in WebGL during 2D and 3D drawing involving transformations.

c. The _____depth_______ buffer is used for hidden surface removal.

d. In WebGL lighting model, light scattered so much it is difficult to tell the source is called Ambient_ light. Light coming from one direction tending to bounce off a surface in mostly 1 direction is called Specular___ light. Light coming from one direction but is scattered in many directions is called Diffuse__ light.

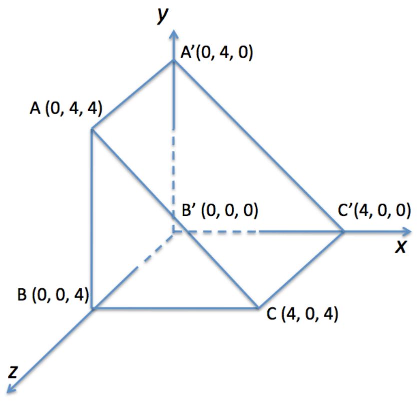
e. In texture mapping, the s and t values of the texel coordinates (s, t) is limited to the range between ___0___ and ___1_____.

f. In perspective projection, the vertices further away from the viewer appear to have smaller x and y coordinate values, this is the result of applying the step: Perspective_ Division__, in the graphics pipeline.

2. (45 pts) Explain how the vertices specified for a 3D object is processed step by step in the Vertex Shader using the execution model of graphics on GPU.
Given:
   - the vertices describing the 3D shape in the world coordinates,
   - the camera location, look at position and up direction specified in the lookAt function, as well as
   - the orthographic projection setup

shown in the figure below, answer the following questions:
   a. What is the view matrix generated at line 1 ?
   b. What is the modelviewMatrix matrix generated after line 4 ?
   c. What is the projection matrix generated at line 5?
   d. After the modelviewMatrix and projectionMatrix have been sent to the vertex shader, they are used to transform the individual vertices into their final coordinates in the clip-coordinates. Compute the coordinates of vertices B and C' in the clip coordinates.

```
modelviewMatrix = lookAt(0, 0, 8, 0, 2, 0, 0, 1, 0);      // line 1
t=translate(1, 0, 1);                                      // line 2
r=rotate(45, 0, 0, 1);                                     // line 3
modelviewMatrix = mult(mult(modelviewMatrix, r), t);   // line 4
projectionMatrix = ortho(-5, 5, -2, 6, -10, 10);           // line 5
```



a) $\text{viewMatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.97 & 0.24 & -1.94 \\ 0 & -0.24 & 0.97 & -7.76 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

b) $\text{modelviewMatrix} = \begin{bmatrix} 0.707 & -0.707 & 0 & 0.707 \\ 0.68 & 0.68 & 0.24 & -1.01 \\ -0.17 & -0.17 & 0.97 & -6.96 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

c) $\text{projectionMatrix}=\begin{bmatrix} 0.2 & 0 & 0 & 0 \\ 0 & 0.25 & 0 & -0.5 \\ 0 & 0 & -0.1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

d) P' = modelviewMatrix * projectionMatrix*P

P’ = $\begin{bmatrix} 0.141 & -0.176 & 0 & 1.06 \\ 0.137 & 0.17 & -0.24 & -1.35 \\ -0.03 & -0.04 & -0.09 & -6.87 \\ 0 & 0 & 0 & 1 \end{bmatrix}$*P

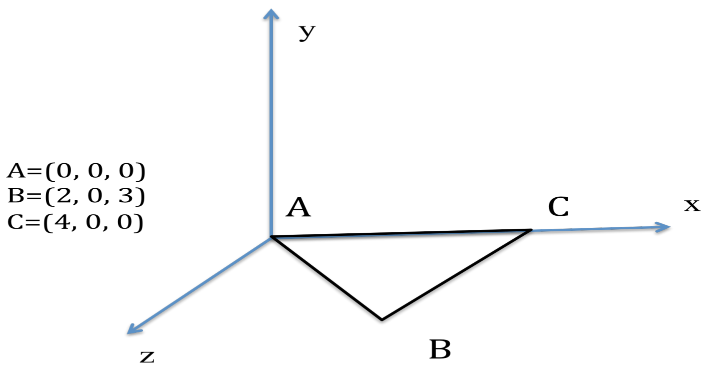
B:[0, 0, 4, 1]

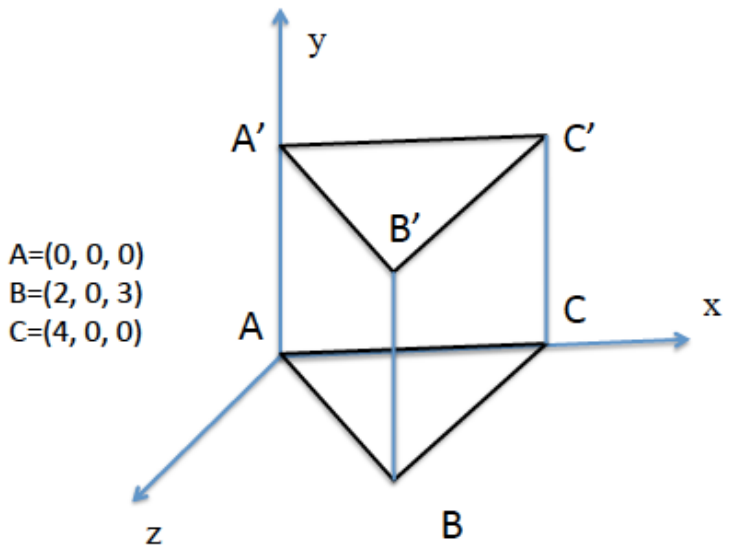
P’(B)=$\begin{bmatrix} 0.141 & -0.176 & 0 & 1.06 \\ 0.137 & 0.17 & -0.24 & -1.35 \\ -0.03 & -0.04 & -0.09 & -6.87 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 \\ 0 \\ 4 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.06 \\ -0.24 * 4 - 1.35 \\ -0.09 * 4 - 6.87 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.06 \\ -2.079 \\ -7.23 \\ 1 \end{bmatrix}$

C’:[4, 0, 0, 1]

P’(C’)=$\begin{bmatrix} 0.141 & -0.176 & 0 & 1.06 \\ 0.137 & 0.17 & -0.24 & -1.35 \\ -0.03 & -0.04 & -0.09 & -6.87 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 4 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.141 * 4 + 1.06 \\ 0.137 * 4 - 1.35 \\ -0.03 * 4 - 6.87 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.624 \\ -0.802 \\ -6.99 \\ 1 \end{bmatrix}$

3. (45 points) An extruded shape is formed from the base triangle shown below. The height of the extruded shape is 2.
    a. Define the extruded shape in terms of the vertex list, normal list, and face list. When normal to a face is not readily computable, apply Newell's method for computation. Show the computations involved.
    b. Suppose all the relevant data, e.g., the vertex positions, the faces, and the normals have all been stored in the appropriate arrays and pushed onto the vertex shader, show all the relevant WebGL code (in .js file) to setup proper lighting and object material properties to display a blue and shiny extruded Triangle. Use a white directional light with light direction [2, 2, 4].
    c. Show WebGL code (used in .js file) to put an image "scene.jpg" (the size of the image is not power of two) onto each side of the extruded shape as 2D texture.





(part a) vertex list

```
    var vertices=[ vec4(0, 0, 0, 1),   // A(0)
                   vec4(2, 0, 3, 1),   // B(1)
                   vec4(4, 0, 0, 1),   // C(2)
                   vec4(0, 2, 0, 1),   // A'(3)
                   vec4(2, 2, 3, 1),   // B'(4)
                   vec4(4, 2, 0, 1) ]; // C'(5)
```

face list	vertices	normal
1	A'ABB'	1, 1, 1, 1
2	B'BCC'	2, 2, 2, 2
3	C'CAA'	3, 3, 3, 3
4	A'B'C'	4, 4, 4
5	CBA	5, 5, 5

Normal list

1	computed below → vec3(-12, 0, 6) → normalize to vec3(-0.89, 0, 0.447)
2	computed below →vec3(12, 0, 6) → normalize to vec3(0.89, 0, 0.447)
3	vec3(0, 0, -1)
4	vec3(0, 1, 0)
5	vec3(0, -1, 0)

$$n_x = \sum_{i=0}^{N-1} (y_i - y_{ni})(z_i + z_{ni})$$

$$n_y = \sum_{i=0}^{N-1} (z_i - z_{ni})(x_i + x_{ni})$$

$$n_z = \sum_{i=0}^{N-1} (x_i - x_{ni})(y_i + y_{ni})$$

ni – neighbor of i

Compute normal vector for face 1: A'ABB' (Counterclock wise rotation of the face)
A' neighbor is A, A neighbor is B, B neighbor is B', B' neighbor is A'

	x	y	z
A'	0	2	0
A	0	0	0
B	2	0	3
B'	2	2	3

Nx = (A'y-Ay)*(A'z+Az) + (Ay-By)*(Az+Bz) + (By-B'y)*(Bz+B'z) + (B'y-A'y)*(B'z+A'z)
= 2*0 + 0*3 + (-2)*6 + 0*3 = -12
Ny=(A'z-Az)*(A'x+Ax) + (Az-Bz)*(Ax+Bx)+(Bz-B'z)*(Bx+B'x) + (B'z-A'z)*(B'x+A'x)
= 0*0 + (-3)*2 + 0*4 + 3*2 = 0
Nz=(A'x-Ax)*(A'y+Ay) + (Ax-Bx)*(Ay+By)+(Bx-B'x)*(By+B'y) + (B'x-A'x)*(B'y+A'y)
= 0*2 + (-2)*0 + 0*2 + 2*3 = 6

Compute normal vector for face 2: B'BCC' (Counterclock wise rotation of the face)
A' neighbor is A, A neighbor is B, B neighbor is B', B' neighbor is A'

	x	y	z
B'	2	2	3
B	2	0	3
C	4	0	0
C'	4	2	0

Nx = (B'y-By)*(B'z+Bz) + (By-Cy)*(Bz+Cz) + (Cy-C'y)*(Cz+C'z) +(C'y-B'y)*(C'z+B'z)
= 2*6 + 0*3 + (-2)*0 + 0*3 = 12

Ny=(B’z-Bz)*(B’x+Bx) + (Bz-Cz)*(Bx+Cx)+(Cz-C’z)*(Cx+C’x) + (C’z-B’z)*(C’x+B’x)
 = 0*4 + (3)*6 + 0*8 + (-3)*6 = 0
Nz=(B’x-Bx)*(B’y+By) + (Bx-Cx)*(By+Cy)+(Cx-C’x)*(Cy+C’y)+(C’x-B’x)*(C’y+B’y)
 = 0*2 + (-2)*0 + 0*2 + 2*3= 6

(part b)

```
var lightPosition = vec4(2, 2, 4, 0 );

var lightAmbient = vec4(0.2, 0.2, 0.2, 1.0 );
var lightDiffuse = vec4(.8, 0.8, 0.8, 1.0 );
var lightSpecular = vec4( .8, .8, .8, 1.0 );

var materialAmbient = vec4( 0.2, .2, .2, 1.0 );
var materialDiffuse = vec4( 0, 1, 0, 1.0);
var materialSpecular = vec4( 0, 1, 0, 1.0 );

var materialShininess = 90.0;

SetupLightingMaterial();

function SetupLightingMaterial()
{
  // set up lighting and material
  var ambientProduct = mult(lightAmbient, materialAmbient);
  var diffuseProduct = mult(lightDiffuse, materialDiffuse);
  var specularProduct = mult(lightSpecular, materialSpecular);

  // send lighting and material coefficient products to GPU
  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(lightPosition) );
  gl.uniform1f( gl.getUniformLocation(program, "shininess"),materialShininess );
}
```

(part c)

```
// texture coordinates
var texCoord = [
  vec2(0, 1),
  vec2(0, 0),
  vec2(1, 1),
  vec2(1, 0),
];

  // in init() function, add the following:
  // set up texture buffer, send texture coordinate info to fragment shader
  var tBuffer = gl.createBuffer();
  gl.bindBuffer( gl.ARRAY_BUFFER, tBuffer );
  gl.bufferData( gl.ARRAY_BUFFER, flatten(texCoordsArray), gl.STATIC_DRAW );
```

```
        var vTexCoord = gl.getAttribLocation( program, "vTexCoord" );
        gl.vertexAttribPointer( vTexCoord, 2, gl.FLOAT, false, 0, 0 );
        gl.enableVertexAttribArray( vTexCoord );

        EstablishTextures();

      function EstablishTextures()
      {
        texture1 = gl.createTexture();

        // create the image object
        texture1.image = new Image();

        // Tell the broswer to load an image
        texture1.image.src='scene.jpg';

        // register the event handler to be called on loading an image
        texture1.image.onload = function() {  loadTexture(texture1, gl.TEXTURE0); }
      }

      function loadTexture(texture, whichTexture)
      {
        // Flip the image's y axis
        gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);

        // Enable texture unit 1
        gl.activeTexture(whichTexture);

        // bind the texture object to the target
        gl.bindTexture( gl.TEXTURE_2D, texture);

        // set the texture image
        gl.texImage2D( gl.TEXTURE_2D, 0, gl.RGB, gl.RGB, gl.UNSIGNED_BYTE, texture.image );

        // version 1 (combination needed for images that are not powers of 2
        gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
        gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);

        // set the texture parameters
        gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
        gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
      }

  // in render function, add the following:
    // use texture0 to draw the sides of the extruded shape
    gl.uniform1i(gl.getUniformLocation(program, "texture"), 0);
    gl.drawArrays(gl.TRIANGLES, 0, 24);    // total 8 triangles
```