CSCI 4250/5250 Take Home Test 2 (100 pts) Name ___TIEN DINH___

Due: midnight, Sunday, Dec 3rd

You are required to type your answers for these questions.

Open Book & Notes – You are on the honor system and 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, ___________________TIEN DINH________________________, worked all of the problems on this test completely on my own without any assistance from any other person. My only resources were the textbook, online course material, my notes, and Internet sources.

1. **(10 pts) Fill in the blanks:**

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

   b. The ____________Model View________________ 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. **(10 pts) Short answer questions:**

   a. Compute the angle between these two vectors: **v1**=(2, 4, 2) and **v2**=(3, 5, 2)?

```
cos θ    = (v1 * v2) / (|v1|*|v2|)
         = (2*3 + 4*5 + 2*2) / ( sqrt(2^2 + 4^2 + 2^2) * sqrt(3^2 + 5^2 + 2^2) )
         = 30 / ( sqrt(24) * sqrt(38) )
         = 0.9933992678
```

θ = arccos(0.9933992678) = 6.587° or 0.0365 rad

   b. Given the coordinates of two points, how does the graphics system figure out the location of all the points in between these two vertices? Explain with the example: given two points: **A**=(2, 4, 2) and **B**=(3, 5, 2), compute the point P on the line segment AB, where P is at 1/4 the way from A to B? (i.e., t=1/4)

The process is called tweening, using this parametric form: $P(\alpha)=P_0 + \alpha\, d$
It gives set of all points that pass through P_0 in the direction of the vector d
Another representation: $P(t)=P_0 + t*(P_1-P_0)$

```
P(1/4)  = (2;4;2) + 1/4 * [(3;5;2) - (2;4;2)]
        = (2;4;2) + 1/4 * (1;1;0)
        = (2;4;2) + (1/4; 1/4; 0)
        = (2.25; 4.25; 2)
```

   c. Given three points on a plane: A(3, 2, 1), B(3, 4, 2), and C(2, 4, 2), compute the normal vector to this plane, i.e., the vector perpendicular to the plane containing the triangle ABC?

AB = (3; 4; 2) - (3; 2; 1) = (0; 2; 1)
CB = (3; 4; 2) - (2; 4; 2) = (1; 0; 0)

normal vector = AB x CB (cross product)

= (0; 2; 1) x (1; 0; 0)

$$= \begin{vmatrix} 2 & 1 \\ 0 & 0 \end{vmatrix} i + \begin{vmatrix} 0 & 1 \\ 1 & 0 \end{vmatrix} j + \begin{vmatrix} 0 & 2 \\ 1 & 0 \end{vmatrix} k$$

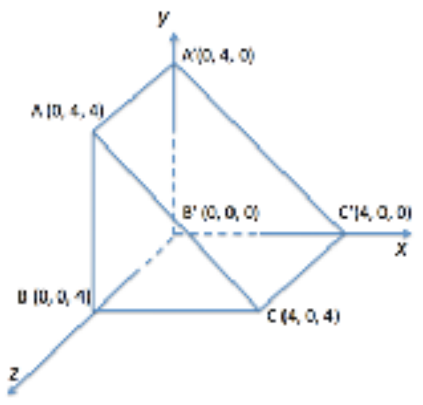
= (0; -1; -2)

So the vector perpendicular to triangle ABC is also (0; -1; -2)

3. (35 pts) 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, and
   - the orthographic projection setup,

   answer the following questions:
   a. What is the view matrix generated at line 1 ? Show all steps in how the view matrix is derived.
   b. What is the modelviewMatrix matrix generated from lines 2, 3, and 4 ?
   c. What is the projection matrix generated at line 5? Show all the matrix is derived.
   d. After the modelviewMatrix and projectionMatrix have been sent to the vertex shader, they are used to transform the individual vertices in each object into their final coordinates in the clip-coordinates. Compute the coordinates of vertices B in the clip coordinates.

```
modelviewMatrix = lookAt(0, 0, 8, 0, 0, 0, 0, 1, 0);      // line 1
t=translate(1, 0, 0);                                      // line 2
r=rotate(30, 0, 1, 0);                                     // line 3
modelviewMatrix = mult(mult(modelviewMatrix, r), t);   // line 4
projectionMatrix = ortho(-5, 5, -6, 6, 2, 10);             // line 5
```



a) **lookAt(0, 0, 8, 0, 0, 0, 0, 1, 0)**.
eye(0, 0, 8) look(0, 0, 0) at(0, 1, 0)
n = eye - look = (0, 8, 0) —-> normalize(n) = (0, 0, 1)
u = up x n = (0, 1, 0) x (0, 0, 8) = (8, 0, 0) —-> normalize(u) = (1, 0, 0)
v = n x u = (0, 0, 8) x (8, 0, 0) = (0, -64, 0) —-> normalize(v) = (0, -1, 0)

dx = -eye . u = -(0, 0, 8) . (1, 0 , 0) = 0
dy = -eye . v = -(0, 0, 8) . (0, -1, 0) = 0
dz = -eye . n = -(0, 0, 8) . (0, 0, 1) = -8

view Matrix:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -8 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

b) mult(mult(modelViewMatrix, t), r)

= view Matrix * **translate (1,0,0)**

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -8 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

then the result of View Matrix times translate, times with rotation matrix:

*** rotate (30, 0, 1, 0)**

$$\begin{bmatrix} \cos 30 & 0 & \sin 30 & 0 \\ 0 & 1 & 0 & 0 \\ -\sin 30 & 0 & \cos 30 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

result in modelViewMatrix:

$$\begin{bmatrix} 0.866 & 0 & 0.5 & 1 \\ 0 & 1 & 0 & 0 \\ -0.5 & 0 & 0.866 & -8 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

c) **ortho(-5, 5, -6, 6, 2, 10)**

Projection matrix: P = S * T

S * T

$$\begin{bmatrix} 2/(\text{right-left}) & 0 & 0 & 0 \\ 0 & 2/(\text{top-bottom}) & 0 & 0 \\ 0 & 0 & -2/(\text{far-near}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & -(\text{left+right})/2 \\ 0 & 1 & 0 & -(\text{bottom+top})/2 \\ 0 & 0 & 1 & (\text{near+far})/2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.2 & 0 & 0 & 0 \\ 0 & 0.167 & 0 & 0 \\ 0 & 0 & -0.25 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.2 & 0 & 0 & 0 \\ 0 & 0.167 & 0 & 0 \\ 0 & 0 & -0.25 & -1.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

d) **Point B (0,0,4)**. The position of point B in the clip coordinate is:

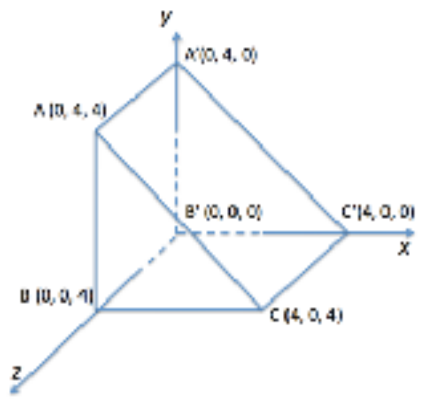
P * V * B =

$$\begin{bmatrix} 0.2 & 0 & 0 & 0 \\ 0 & 0.167 & 0 & 0 \\ 0 & 0 & -0.25 & -1.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0.866 & 0 & 0.5 & 1 \\ 0 & 1 & 0 & 0 \\ -0.5 & 0 & 0.866 & -8 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 \\ 0 \\ 4 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.6 \\ 0 \\ -0.366 \\ 1 \end{bmatrix}$$

So point B in the clip coordinate is: (0.6, 0, -0.366)

4. (10 pts) Assuming a perspective projection is used to view the object defined below:



modelviewMatrix = lookAt(0, 0, 8, 0, 0, 0, 0, 1, 0*);* *// the camera/eye is set up the same way*
// as in problem 3
projectionMatrix = **frustum**(-6, 6, -6, 6, 2, 12);

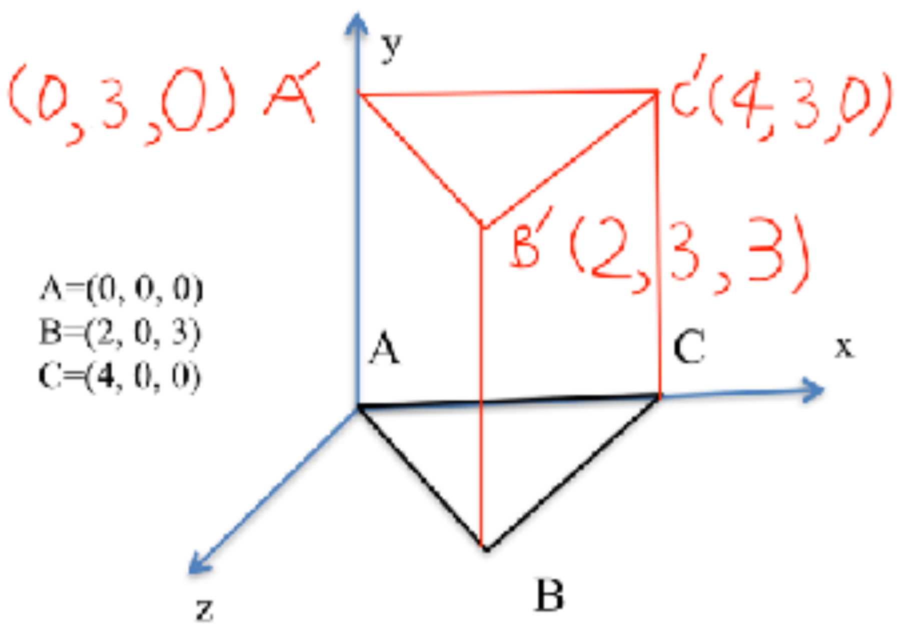
Compute the x and y coordinates of the points A and A' as they project on the near plane.

NewAx = N*Ax/(-Az) = 2*0/(-4) = 0
New Ay = N*Ay/(-Az) = 2*4/(-4) = -2
New A(x, y) = (0, -2)

New A'x = N*A'x/(-A'z) = 2*0/(0) = ∞
New A'y = N*A'y/(-A'z) = 2*4/(0) = ∞
New A'(x, y) = (∞, ∞)

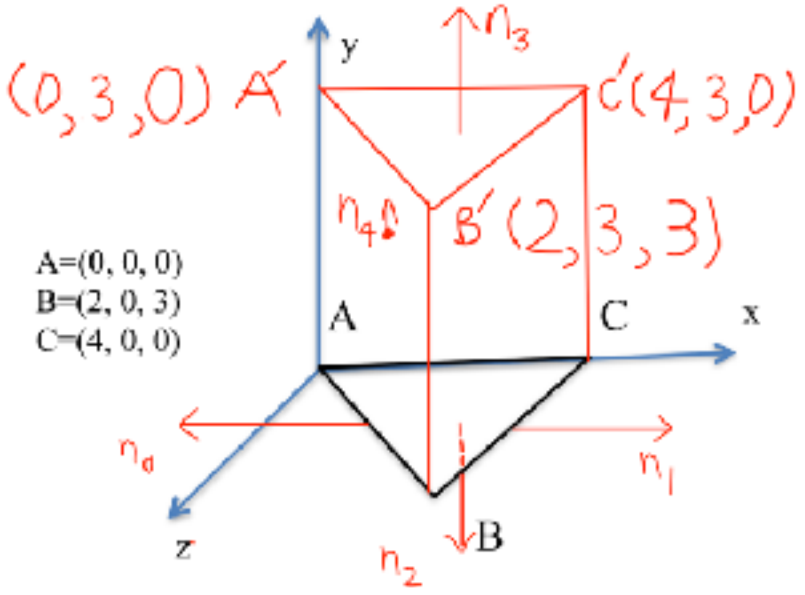
Which means that the new A' position will be at very far away from near plan. It goes to infinity.

5. (35 points) An extruded shape is formed from the base triangle shown below. The height of the extruded shape is 3.
    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 each list in a table format as discussed in class.
    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 set to [4, 2, 4, 0].
        i. Show WebGL code needed in .js file to put an image "scene.jpg" (the size of the image is not a power of two) onto each side of the extruded shape as 2D texture.



a)
Vertexes List:

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



Normal List:

Normal	Nx	Ny	Nz
0 (left): ABB'A'	-3	0	2
1 (right): BCC'B'	3	0	2
2 (bottom): ACB	0	-1	0
3 (top): A'B'C'	0	1	0
4 (back): A'C'CA	0	0	-1

Here is how I do Newell for ABB'A': => N = 4 (vertices)

i=0->A(0,0,0) i=1->B(2,0,3) i=2->B'(2,3,3) i=3->A'(0,3,0)

n_x $= (y_0 - y_1)*(z_0 + z_1) + (y_1 - y_2)*(z_1 + z_2) + (y_2 - y_3)*(z_2 + z_3) + (y_3 - y_0)*(z_3 + z_0)$

$= (0 - 0)*(\ldots) + (0 - 3)*(3 + 3) + (3 - 3)*(\ldots) + (\ldots)*(0 + 0)$

$= -18$

n_y $= (z_0 - z_1)*(x_0 + x_1) + (z_1 - z_2)*(x_1 + x_2) + (z_2 - z_3)*(x_2 + x_3) + (z_3 - z_0)*(x_3 + x_0)$

$= (0 - 3)*(0 + 2) + (3 - 3)*(\ldots) + (3 - 0)*(2 + 0) + (0 - 0)*(\ldots)$

$= -6 + 6 = 0$

n_y $= (x_0 - x_1)*(y_0 + y_1) + (x_1 - x_2)*(y_1 + y_2) + (x_2 - x_3)*(y_2 + y_3) + (x_3 - x_0)*(y_3 + y_0)$

$= (\ldots)*(0 + 0) + (2 - 2)*(\ldots) + (2 - 0)*(3 + 3) + (0 - 0)*(\ldots)$

$= 12$

Mathematically simplify vector (-18, 0, 12) to (-3, 0, 2)

Similarly by doing the same process, Newell normal vector of BCC'B' is: (18, 0, 12) simplify to (3, 0 , 2)

Face list:

FACE	VERTICES	NORMAL
0 (left): ABB'A'	A, B, B', A'	(-3, 0, 2)
1 (right): BCC'B'	B, C, C', B'	(3, 0, 2)
2 (bottom): ACB	A, C, B	(0, -1, 0)
3 (top): A'B'C'	A', B', C'	(0, 1, 0)
4 (back): A'C'CA	A', C', C, A	(0, 0, -1)

b)
```
var lightPosition = vec4(4, 2, 4, 0.0 );
var lightAmbient = vec4( 1.0, 1.0, 1.0, 1.0 );        // White light
var lightDiffuse = vec4( 1.0, 1.0, 1.0, 1.0 );
var lightSpecular = vec4( 1.0, 1.0, 1.0, 1.0 );

modelViewMatrix=mvMatrixStack.pop();
materialAmbient = vec4(0.0, 0.0, 1.0, 1.0);          // Blue object
materialDiffuse = vec4(0.0, 0.0, 1.0, 1.0);
materialSpecular = vec4( 1.0, 1.0, 1.0, 1.0 );
materialShininess = 50;              // Less number = more shiny
mvMatrixStack.push(modelViewMatrix);
```

c) Here is my program to achieve the task:
```
var canvas;
var gl;
var program;
var image;

var numVertices  = 18;

var pointsArray = [];
var colorsArray = [];
var texCoordsArray = [];

var texture;

var texCoord = [
    vec2(0, 0),
    vec2(0, 1),
    vec2(1, 1),
    vec2(1, 0)
];

var vertices = [
    vec4( 0.0, 0.0,  0.0, 1.0 ),        // A-0
    vec4( 2.0,  0.0,  3.0, 1.0 ),       // B-1
    vec4( 4.0,  0.0,  0.0, 1.0 ),       // C-2
    vec4( 0.0, 3.0,  0.0, 1.0 ),        // A'-3
    vec4( 2.0,  3.0,  3.0, 1.0 ),       // B'-4
    vec4( 4.0,  3.0,  0.0, 1.0 ),       // C'-5
];


function quad(a, b, c, d) {
     pointsArray.push(vertices[a]);
     colorsArray.push(vertexColors[a]);
     texCoordsArray.push(texCoord[0]);

     pointsArray.push(vertices[b]);
     colorsArray.push(vertexColors[a]);
     texCoordsArray.push(texCoord[1]);

     pointsArray.push(vertices[c]);
     colorsArray.push(vertexColors[a]);
     texCoordsArray.push(texCoord[2]);

     pointsArray.push(vertices[a]);
     colorsArray.push(vertexColors[a]);
     texCoordsArray.push(texCoord[0]);

     pointsArray.push(vertices[c]);
     colorsArray.push(vertexColors[a]);
     texCoordsArray.push(texCoord[2]);

     pointsArray.push(vertices[d]);
     colorsArray.push(vertexColors[a]);
     texCoordsArray.push(texCoord[3]);
}

function tri(a, b, c) {
     pointsArray.push(vertices[a]);
     colorsArray.push(vertexColors[a]);
     texCoordsArray.push(texCoord[0]);

     pointsArray.push(vertices[b]);
     colorsArray.push(vertexColors[a]);
     texCoordsArray.push(texCoord[1]);

     pointsArray.push(vertices[c]);
     colorsArray.push(vertexColors[a]);
     texCoordsArray.push(texCoord[2]);
}
```

```
function createObject()
{
   quad( 0, 1, 4, 3 );      // ABB'A'
   quad( 1, 2, 5, 4 );      // BCC'B'
   tri( 0, 2, 1 );          // ACB
   tri( 3, 4, 5 );          // A'B'C'
   quad( 3, 5, 2, 0 );      // A'C'CA
}

window.onload = function init() {

   canvas = document.getElementById( "gl-canvas" );

   gl = WebGLUtils.setupWebGL( canvas );
   if ( !gl ) { alert( "WebGL isn't available" ); }

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

   gl.enable(gl.DEPTH_TEST);

   // generate the mesh vertex positions, color, normal, textures, etc.
   createObject();

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

   //  initialize the attribute buffers
   //
   var cBuffer = gl.createBuffer();
   gl.bindBuffer( gl.ARRAY_BUFFER, cBuffer );
   gl.bufferData( gl.ARRAY_BUFFER, flatten(colorsArray), gl.STATIC_DRAW );

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

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

   var vPosition = gl.getAttribLocation( program, "vPosition" );
   gl.vertexAttribPointer( vPosition, 4, gl.FLOAT, false, 0, 0 );
```

```
    gl.enableVertexAttribArray( vPosition );

    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 );


    // ============== Establish Textures =================
    // create the texture object
    texture = gl.createTexture();

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

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

    // Tell the broswer to load an image
    texture.image.src='sky.jpg';

    render();
}

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

    // Enable texture unit 0
    gl.activeTexture(gl.TEXTURE0);

    // 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 );

    // set the texture parameters
    //gl.generateMipmap( gl.TEXTURE_2D );
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST );
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST );

    // set the texture unit 0 the sampler
    gl.uniform1i(gl.getUniformLocation(program, "texture"), 0);
```

```
}

var render = function()
{
   gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

   //gl.drawArrays( gl.TRIANGLES, 0, numVertices );
   gl.drawArrays( gl.TRIANGLES, 0, numVertices);
   requestAnimFrame(render);
}
```