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

Due: midnight, Thursday, Nov 21st
You are required to type your answers for these questions.
Save the file as a PDF file and submit through D2L dropbox.

Open Book & Notes. You may use computer program to assist your calculations where needed.

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, ____, 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, my programs, 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 texture mapping, the s and t values of the texel coordinates (s, t) is limited to the range between __0____ and __1_____.

   e. 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.

   f. When specifying the points on each face of a 3D shape, the points should be specified in _____________counter-clockwise______ order.

2. (7 pts) Given the light specified as $l_d = l_a = (0.8, 0, 0)$, and $l_s=(1, 0.5, 0.5)$, the location of the light source is at (2, 1, 3, 1), and the object material property specified as: $\rho_d =(1, 0, 0.5)$, $\rho_s =(0.9, 0.9, 0.9)$, $\rho_a=(0.5, 0.5, 1)$, shininess value = 30. Applying the lighting model studied in class to compute the **specular light component** of the color of a vertex p located at (1, 0.5, 0, 1) on the object. The eye location is (0, 0, 0, 1). The normal vector at the vertex is (1, 1, 0.5, 0). Use the simplified halfway-vector approach.

The vector from light source to point is v1=(1, 0.5, 0, 1)-(2, 1, 3, 1)=(-1, -0.5, -3, 0)
The vector from point to eye is v2=(1, 0.5, 0, 1) – (0, 0, 0, 1) = (1, 0.5, 0, 0)
The half way vector between v1 and v2 is H=v1 + v2) = (-1, -0.5, -3) + (1, 0.5, 0, 0) = (0, 0, -3, 0)

The angle between the normal vector N=(1, 1, 0.5, 0) and the half way vector is α.
$\cos(\alpha)$ = N $N \cdot H = (1, 1, 0.5, 0) \cdot (0, 0, -3, 0) = -1.5$; this value is less than 0, so it is change to 0

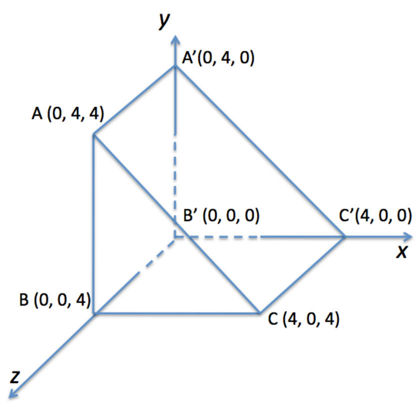
specular light component = $l_s * \rho_s * \max(\cos(\alpha), 0)^{30} = (1, 0.5, 0.5) * (0.9, 0.9, 0.9) * \max(\cos(\alpha), 0)^{30}$ =(0, 0, 0)

3. (30 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 computed at line 4 ? Show the matrix t, matrix r, and the matrix modelviewMatrix after the multiplication is performed.
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 in each object into their final coordinates in the clip-coordinates. Compute the coordinates of vertices B in the clip coordinates.

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



a) $\text{viewMatrix} = \begin{bmatrix} .98 & 0 & -0.19 & 0 \\ 0.04 & 0.98 & 0.18 & -0.98 \\ 0.19 & -0.19 & 0.96 & -5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

b) $\text{modelviewMatrix} = \begin{bmatrix} 0.849 & -0.49 & -0.19 & 0.65 \\ 0.52 & 0.83 & 0.19 & -0.27 \\ 0.07 & -0.26 & 0.96 & -3.97 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

c) $\text{projectionMatrix} = \begin{bmatrix} 0.16 & 0 & 0 & 0 \\ 0 & 0.16 & 0 & 0 \\ 0 & 0 & -0.25 & -1.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

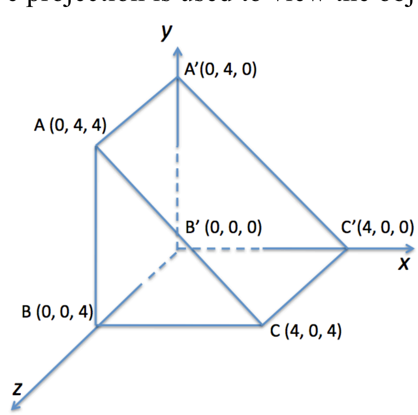
d) P' = projectionMatrix* modelviewMatrix *P

$$P' = \begin{bmatrix} 0.142 & -0.08 & -0.03 & 0.1 \\ 0.08 & 0.14 & 0.03 & -0.04 \\ -0.02 & 0.07 & -0.24 & -0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} * P$$

B:[0, 0, 4, 1]

$$P'(B) = \begin{bmatrix} 0.142 & -0.08 & -0.03 & 0.1 \\ 0.08 & 0.14 & 0.03 & -0.04 \\ -0.02 & 0.07 & -0.24 & -0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 4 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.02 \\ 0.08 \\ -1.46 \\ 1 \end{bmatrix}$$

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



modelviewMatrix = **lookAt(1, 0, 5, 0, 1, 0, 0, 1, 0);**
projectionMatrix = frustum(-6, 6, -8, 8, 2, 10);

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

Assuming coordinates in eye-coordinates:

A: (0, 4, 4)

Ax'=N*Ax/(-Az) =2*0 = 0

Ay'= N*Ay/(-Az) = 2*4/(-4) = -2

A is displayed at (0, -2) on the near plane

A' (0, 4, 0)

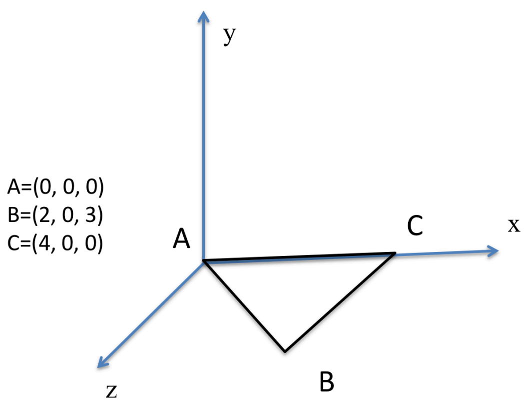
Ax'=N*Ax/(-Az) =2*0 = 0

Ay'= N*Ay/(-Az) = 2*4/(0) = undefined

Not displayed

5. (30 points) An extruded shape is formed from the base triangle shown below. The triangle extruded in the direction <0, 1, 0>, i.e., along the y-axis. 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].
   c. Show WebGL code needed in .js file to put an image "label.jpg" (both the length and width of the image is a power of two, so no special steps is needed for texture mapping) onto 3 sides of the extruded shape as 2D texture. Show ALL the additional code needed for the adding the 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
```

6. (15 pts) Apply the Liang Barsky 3D edge/line clipping algorithm to clip the line V_2V_3 against the Canonical View Volume (CVV) as shown below. That is to compute the end points of the line segment that is inside the CVV. Assuming the homogeneous coordinates for V_2 is (8, 4, -8, 5) and for V_3 is (2, 3, 2, 4). Show step-by-step computations in the process.

For V_2
BC0: w+x=5+8=13
BC1: w-x=5-8= -3 < 0
BC2: w+y=5+4=9
BC3: w-y=5-4=1
BC4: w+z=5+(-8)= -3 <0
BC5: w-z=5-(-8)=13
outcode is 010010

For V_2, outcode is 000000
BC0: w+x = 4+2 = 6
BC1: w-x = 4-2 = 2
BC2: w+y = 4+3 = 7
BC3: w-y = 4-3 = 1
BC4: w+z = 4+2 = 6
BC5: w-z = 4-2 = 2

010010 | 000000 != 0
010010 & 000000 != 1

Need to clip again plane for i=1
t = V_2BC1/(V_2BC1- V_3BC1) = (-3)((-3)-(2)) = 3/5

Need to clip again plane for i=4
t = V_2BC4/(V_2BC4- V_3BC4) = (-3)((-3)-(6)) = 1/3

Since line V2V3 enters the CVV, V2 is the vertex entering the CVV, the largest t value is retained for computing the intersection location V_2', i.e., t=3/5 is used next.

Update V_2 to V_2':
V_2'.x = 8+3/5*(2 - 8) = 4.4
V_2'.y = 4+3/5*(3 - 4) = 3.4
V_2'.z = -8 +3/5*(2 - (-8)) = -2
V_2'.w = 5+3/5*(4 - 5) = 4.4

V_2' **: (4.4, 3.4, -2, 4.4)**

After perspective division: (1, 0.77, -0.45, 1)