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

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, __David Piper___________________________, 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 _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. (10 pts) Short answer questions:

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

$\alpha = arccos((\mathbf{a} \bullet \mathbf{b})/(|\mathbf{a}|*|\mathbf{b}|))$

v1 • **v2** = (2 * 3) + (4 * 5) + (2 * 2) = 30

|**v1**| = $\sqrt{2^2 + 4^2 + 2^2}$ = 4.89898

|**v2**| = $\sqrt{3^2 + 5^2 + 2^2}$ = 6.16441

|**v1**| * |**v2**| = 30.1993

(**v1** • **v2**)/ (|**v1**| * |**v2**|) = 0.993399

arccos(0.993399) = 0.114961 radians = 6.5868 degrees

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 graphics system computes the location of the points between two vertices using the following formula for various values of t:

p(t) = A*t + B*(1-t)

p(1/4) = A*(1/4) + B*(1-(1/4))

p(1/4) = (2, 4, 2)*(1/4) + (3, 5, 2)*(3/4)
p(1/4) = (1/2, 1, 1/2) + (9/4, 15/4, 3/2)
p(1/4) = (11/4, 19/4, 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?

u = B - A; **v** = C - A; **n** = **u** × **v**

u = (3, 4, 2) - (3, 2, 1) = (0, 2, 1)

v = (2, 4, 2) - (3, 2, 1) = (-1, 2, 1)

n = (0, 2, 1) × (-1, 2, 1)

$n_x = u_y v_z - u_z v_y \ , \ n_y = u_z v_x - u_x v_z \ , \ n_z = u_x v_y - u_y v_x$

n = (2*1 - 1*2, 1*(-1) - 0*1, 0*2 - 2*(-1))

n = (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. View = lookAt(eye, at, up)
   eye = (0, 0, 8)
   at = (0, 0, 0)
   up = (0, 1, 0)
   n = eye - at = (0, 0, 8) - (0, 0, 0) = (0, 0, 8)
   u = up × **n**

   $u_x = up_y n_z - up_z n_y \ , \ u_y = up_z n_x - up_x n_z \ , \ u_z = up_x n_y - up_y n_x$

   u = (1*8 - 0*0, 0*0 - 0*8, 0*0 - 1*8) = (8, 0, -8)

v = **n** × **u**

$$v_x = n_y u_z - n_z u_y \,,\; v_y = n_z u_x - n_x u_z \,,\; v_z = n_x u_y - n_y u_x$$

v = (0*(-8) - 8*0, 8*8 - 0*(-8), 0*0 - 0*8) = (0, 64, 0)

Normalize **n**:

$|\mathbf{n}| = \sqrt{0^2 + 0^2 + 8^2} = 8$

n / |**n**| = (0/8, 0/8, 8/8) = (0, 0, 1)

Normalize **u**:

$|\mathbf{u}| = \sqrt{8^2 + 0^2 + (-8)^2} = 11.3137$

u / |**u**| = (8/11.3137, 0/11.3137, (-8)/11.3137) = (0.707107, 0, -0.707107)

Normalize **v**:

$|\mathbf{v}| = \sqrt{0^2 + 64^2 + 0^2} = 64$

v / |**v**| = (0/64, 64/64, 0/64) = (0, 1, 0)

e = eye - origin = (0, 0, 8)

d = (**-e • u**, **-e • v**, **-e • n**)

-e • u = (0, 0, -8) • (0.707107, 0, -0.707107)

= (0*0.707107) + (0*0) + ((-8)*(-0.707107)) = 0 + 0 + 5.65685 = 5.65685

-e • v = (0, 0, -8) • (0, 1, 0) = (0*0) + (0*1) + ((-8)*0) = 0

-e • n = (0, 0, -8) • (0, 0, 1) = (0*0) + (0*0) + ((-8)*1) = -8

d = (5.65685, 0, -8)

View = [
[0.707107, 0, -0.707107, 5.65685],
[0, 1, 0, 0],
[0, 0, 1, -8],
[0, 0, 0, 1]
]

b. modelViewMatrix = View * Model,

t = [
[1, 0, 0, 1],
[0, 1, 0, 0],
[0, 0, 1, 0],
[0, 0, 0, 1]
]

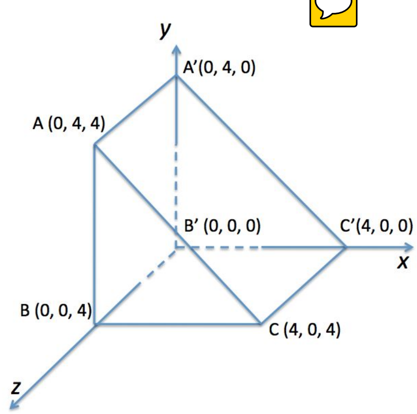
r = [
[cos(30), 0, sin(30), 0],
[0, 1, 0, 0],
[-sin(30), 0, cos(30), 0],
[0, 0, 0, 1]
]

mult(modelviewMatrix, r) = View * r = [
[0.965926, 0, -0.258819, 5.65685],
[0, 1, 0, 0],
[-1/2, 0, 0.866025, -8],
[0, 0, 0, 1]
]

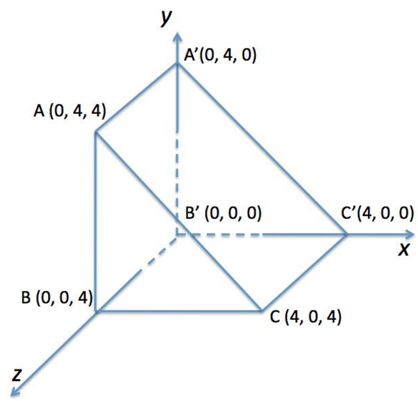
mult(mult(modelviewMatrix, r), t) = [
[0.965926, 0, -0.258819, 6.62278],
[0, 1, 0, 0],
[-1/2, 0, 0.866025, -8.5],
[0, 0, 0, 1]
]

c. **projectionMatrix = ortho(-5, 5, -6, 6, 2, 10)**
left: -5
right: 5
bottom: -6
top: 6
near: 2
far: 10
ortho(left, right, bottom, top, near, far)
projectionMatrix = [
[2/(right-left), 0, 0, -(left+right)/(right-left)],
[0, 2/(top-bottom), 0, -(bottom+top)/(top-bottom)],
[0, 0, -2/(far-near), -(near+far)/(far-near)],
[0, 0, 0, 1]
] = [
[1/5, 0, 0, 0],
[0, 1/6, 0, 0],
[0, 0, -1/4, -3/2],
[0, 0, 0, 1]
]

d. projectionMatrix * modelViewMatrix = [
[0.193185, 0, -0.0517638, 1.32456],
[0, 0.166667, 0, 0],
[0.125, 0, -0.216506, 0.625],
[0, 0, 0, 1]
]
B = (0, 0, 4, 0) (with w of 0)
projectionMatrix * modelViewMatrix * B = (-0.207055, 0, -0.866024, 0)



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.

View = lookAt(eye, at, up)
eye = (0, 0, 8)
at = (0, 0, 0)
up = (0, 1, 0)
n = eye - at = (0, 0, 8) - (0, 0, 0) = (0, 0, 8)
u = up × **n**

$$u_x = up_y n_z - up_z n_y \,,\ u_y = up_z n_x - up_x n_z \,,\ u_z = up_x n_y - up_y n_x$$

u = (1*8 - 0*0, 0*0 - 0*8, 0*0 - 1*8) = (8, 0, -8)
v = **n** × **u**

$$v_x = n_y u_z - n_z u_y \,,\ v_y = n_z u_x - n_x u_z \,,\ v_z = n_x u_y - n_y u_x$$

v = (0*(-8) - 8*0, 8*8 - 0*(-8), 0*0 - 0*8) = (0, 64, 0)

Normalize **n**:

$|\mathbf{n}| = \sqrt{0^2 + 0^2 + 8^2} = 8$

n / |**n**| = (0/8, 0/8, 8/8) = (0, 0, 1)

Normalize **u**:

$|\mathbf{u}| = \sqrt{8^2 + 0^2 + (-8)^2} = 11.3137$

u / |**u**| = (8/11.3137, 0/11.3137, (-8)/11.3137) = (0.707107, 0, -0.707107)

Normalize **v**:

$|\mathbf{v}| = \sqrt{0^2 + 64^2 + 0^2} = 64$

v / |**v**| = (0/64, 64/64, 0/64) = (0, 1, 0)

e = eye - origin = (0, 0, 8)
d = (**-e • u**, **-e • v**, **-e • n**)

-e • u = (0, 0, -8) • (0.707107, 0, -0.707107)
= (0*0.707107) + (0*0) + ((-8)*(-0.707107)) = 0 + 0 + 5.65685 = 5.65685

-e • v = (0, 0, -8) • (0, 1, 0) = (0*0) + (0*1) + ((-8)*0) = 0

-e • n = (0, 0, -8) • (0, 0, 1) = (0*0) + (0*0) + ((-8)*1) = -8

d = (5.65685, 0, -8)

View = [
[0.707107, 0, -0.707107, 5.65685],
[0, 1, 0, 0],
[0, 0, 1, -8],
[0, 0, 0, 1]
]

projectionMatrix = **frustum**(-6, 6, -6, 6, 2, 12);
left: -6
right: 6
bottom: -6
top: 6
near: 2
far: 12

N = 2
a = -(far + near)/(far - near) = -7/5
b = (-2*far*near)/(far - near) = -24/5

projectionMatrix = [
[2, 0, 0, 0],
[0, 2, 0, 0],
[0, 0, -7/5, -24/5],
[0, 0, -1, 0]
]

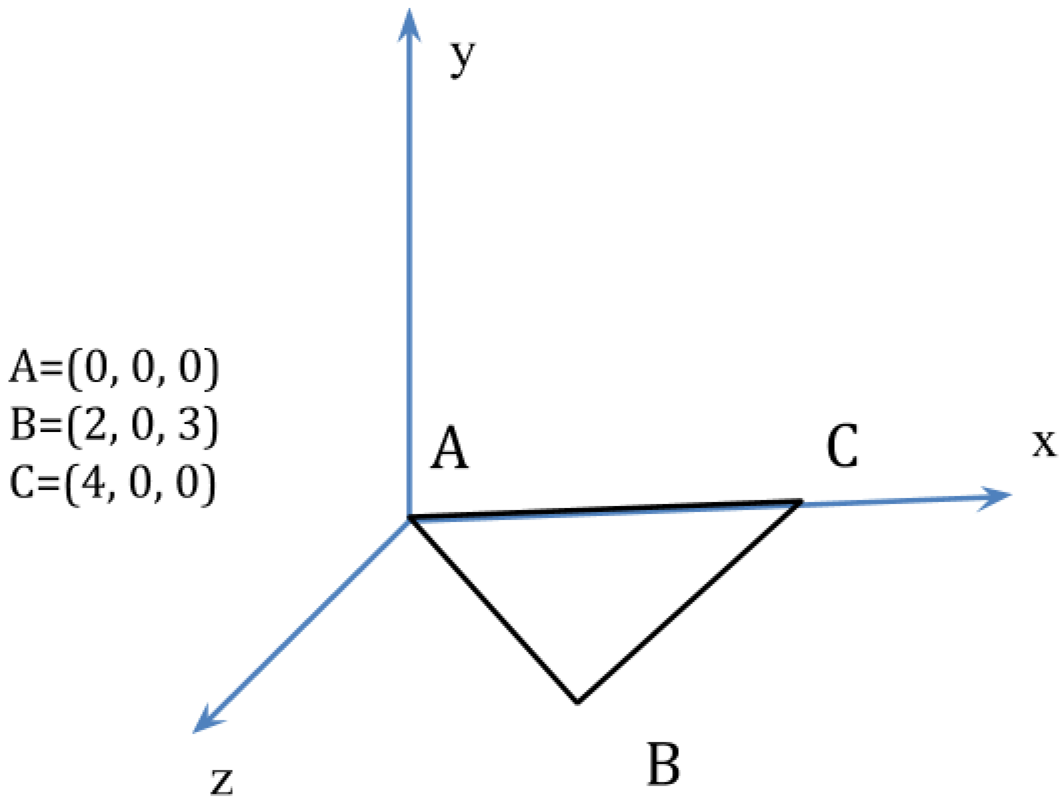
projectionMatrix * modelviewMatrix = [
[1.41421, 0, -1.41421, 11.3137],
[0, 2, 0, 0],
[0, 0, -1.4, 6.4],
[0, 0, -1, 8]
]

A and A’ with w = 0:
A = (0, 4, 4, 0)
A’ = (0, 4, 0, 0)

projectionMatrix * modelviewMatrix * A = (-5.65685, 8, -5.6, -4)
projectionMatrix * modelviewMatrix * A’ = (0, 8, 0, 0)

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].
c. 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.

vertex	x	y	z
0	0	0	0
1	2	0	3
2	4	0	0
3	0	3	0
4	2	3	3

5	4	3	0

normal	n_x	n_y	n_z
0	-0.83205	0	0.5547
1	-0.83205	0	-0.5547
2	0	0	-1
3	0	1	0
4	0	-1	0

Face	Vertices	Normal
0	0, 1, 3, 4	0
1	1, 2, 4, 5	1
2	0, 2, 3, 5	2
3	3, 4, 5	3
4	0, 1, 2	4

b.

```
// light and material
var lightPosition = vec4(4, 2, 4, 0 );

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

var materialAmbient = vec4( .2, .2, .2, 1.0 );
var materialDiffuse = vec4( 0.0, 0.0, 1.0, 1.0);
var materialSpecular = vec4( 0, 0, 0.0, 1.0 );
var materialShininess = 100.0;

// set up lighting and material
ambientProduct = mult(lightAmbient, materialAmbient);
diffuseProduct = mult(lightDiffuse, materialDiffuse);
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 );
```

c.

```
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 );

   // 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);

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


  // ==============  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='scene.jpg';

   render();
```