1) (4 pts each, 12 pts total) **2D Transforms**: What is the 3x3 matrix transform (or sequence of transforms) in homogeneous coordinates for the following. **Also give the inverse.**

a) A translation by 3 along x and -4 along y.

The transform: $$\begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & -4 \\ 0 & 0 & 1 \end{pmatrix}$$

The inverse: $$\begin{pmatrix} 1 & 0 & -3 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{pmatrix}$$

b) A uniform scale by 2 about the fixed point (a,b):

The transform: $$\begin{pmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -a \\ 0 & 1 & -b \\ 0 & 0 & 1 \end{pmatrix}$$

The inverse: $$\begin{pmatrix} 1 & 0 & a/2 \\ 0 & 1 & b \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -a \\ 0 & 1 & -b \\ 0 & 0 & 1 \end{pmatrix}$$

c) A projection onto the y-axis.

The transform: $$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The inverse: does not exist
2) (5 pts each, 15 pts total) Suppose you want to apply the following texture to the square surface on the right.

If the code below is applied, the square will be textured as shown below on the right.

```c
glTexParameteri(GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_S, GL.GL_CLAMP);
glTexParameteri(GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_T, GL.GL_CLAMP);
glBegin(GL.GL_POLYGON);
glTexCoord3f(0,0);   glVertex3f(v0x,v0y,0)
glTexCoord3f(1,0);   glVertex3f(v1x,v1y,0);
glTexCoord3f(1,1);   glVertex3f(v2x,v2y,0);
glTexCoord3f(0,1);    glVertex3f(v3x,v3y,0);
glEnd();
```

In the rectangle on the right, sketch how the texture would appear if the following texture coordinates are used instead?

a. ```c
   glTexCoord3f(-0.5,0);   glVertex3f(v0x,v0y,0)
glTexCoord3f(0.5,0);   glVertex3f(v1x,v1y,0);
glTexCoord3f(0.5,1);   glVertex3f(v2x,v2y,0);
glTexCoord3f(-0.5,1);    glVertex3f(v3x,v3y,0);
```  

b. ```c
   glTexCoord3f(1,0);   glVertex3f(v0x,v0y,0)
glTexCoord3f(1,1);   glVertex3f(v1x,v1y,0);
glTexCoord3f(0,1);    glVertex3f(v3x,v3y,0);
glTexCoord3f(0,0);    glVertex3f(v3x,v3y,0);
```  

c. ```c
   glTexCoord3f(0,0);    glVertex3f(v0x,v0y,0)
glTexCoord3f(1,0);    glVertex3f(v1x,v1y,0);
glTexCoord3f(1,2);    glVertex3f(v2x,v2y,0);
glTexCoord3f(0,2);    glVertex3f(v3x,v3y,0);
```
3) (4pts each, 12 pts total) Blending in OpenGL: Suppose that you draw two rectangles S₁ and S₂, using OpenGL where
- S₁ is closer to the camera than S₂.
- S₁ is drawn first followed by S₂.
- The background color is set to white=(1,1,1).
- the glColor is set to green=(0,1,0, α₁) when S₁ is drawn.
- the glColor is set to red=(1,0,0, α₂) when S₂ is drawn.

What is the resulting (r,g,b) color in the frame buffer corresponding to each of the following. Show how you calculated your answer (you will not get full credit for just an answer written down)

a. **Area 2** if α₁=0.5, α₂=0.6 when z-buffering is turned on.

   (r,g,b) = ___________

   In area 2, S₁ has no effect.
   \[(1-α₂)(1,1,1) + (α₂)(1,0,0) = (.4)(1,1,1)+(.6)(1,0,0) = (1,.4,.4)\]

b. **Area 1** if α₁=0.4, α₂=1.0 when z-buffering is turned on.

   (r,g,b) = ___________

   S₂ has no effect since z buffering is on and S₁ is drawn first (and is in front of S₂).
   \[(1-α₁)(1,1,1) + (α₁)(0,1,0) = (.6)(1,1,1)+(.4)(0,1,0) = (.6,1,6)\]

c. **Area 1** if α₁=0.25, α₂=0.9 when z-buffering is turned off.

   (r,g,b) = ___________

   Both S₁ and S₂ have an effect since they overlap in area 1 and z-buffering is off
   When S₁ is drawn:
   \[(1-α₁)(1,1,1) + (α₁)(1,0,0) = (.75)(1,1,1)+(.25)(0,1,0) = (.75,1., .75)\]
   Then, when S₂ is drawn:
   \[(1-α₂)(.75,1., .75) + (α₂)(1,0,0) = (.1)(. 75,1., .75)+(.9)(1,0,0) = (.975, .1, .075)\]
(4 pts each, 8 pts total) **Fly throughs:** Given the viewer location $Loc$, and the direction/orientation vectors $Forward$, $Right$, and $Up$, how do you update $Loc$, $Forward$, $Right$, and $Up$ in order to do the following. Indicate not only what changes but also what doesn’t change.

a) Move the viewer a small distance (e.g. $\alpha$) forward.

$$Loc = Loc + \alpha \cdot Forward$$

$Forward$, $Right$, $Up$ do not change

b) Turn the viewer a small amount to the left.

$Loc$ and $Up$ do not change

$$Forward = (Forward - \alpha \cdot Right) / \| Forward - \alpha \cdot Right \|$$

$$Right = (Forward \times Up) / \| Forward \times Up \|$$

4) (3 pts each, 9 pts total) The position of a light can be set in any number of places in your code, e.g. before or after gluLookAt, at some point in scene hierarchy, etc. Where should the position be set if

a) You want the light to move with the camera?

Place it before gluLookAt

b) You want the light to stay fixed in the world coordinate system?

Place it after gluLookAt

c) You want the light to be attached to an object (e.g. if you had a headlight on a car).

Place it in the code immediately before drawing the object.

5) (6 pts) Cross Products: If $v1 = (1,0,0)$ and $v2 = (0,0,1)$, what is

a) $v1 \times v2 = (0,-1,0)$ __________

b) $v1 \times v1 = (0,0,0)$ __________
6) (22 pts total) Recall that OpenGL uses a right handed coordinate system where the default camera looks down the negative z world axis, the y axis is up, and the x axis is to the right. In this default position, the world coordinates of a vertex $p_w$ are the same as its camera coordinates $p_e$.

a) (3 pts) Suppose the camera starts in the default position and is translated one unit back (along positive z). If a vertex has world coordinates $p_w = (0,0,0)$, what would be its new camera coordinates, $p_e$? (no math required – just reason it through)

(0, 0 , -1)

b) (3 pts) Suppose the camera starts in the default position and is rotated by 90 degrees about x so that it is looking directly up along the y world axis. If a vertex has coordinates $p_w = (0,0,-1)$, what would $p_e$ be? (no math required – just reason it through)

( 0, -1, 0)

(Continued on next page)
c) (16 pts) Suppose we want to reposition the camera given the parameters:

\[ \text{Eye} = \text{position of the camera} \]
\[ \text{Look} = \text{what the camera is looking at} \]
\[ \text{Up} = \text{the up vector} \]

We can use the function \( \text{gluLookAt(Eye, Look, Up)} \) to calculate \( M \), where \( M \) is the 4x4 matrix used to transform vertices \( p_w \) (world coordinates) into \( p_e \) (eye coordinates):

\[ p_e = M \cdot p_w \]

How does \( \text{gluLookAt} \) calculate \( M \) from \( \text{Eye}, \text{Look}, \text{and Up} \)? That is, what is \( M \) in terms of \( \text{Eye}, \text{Look}, \text{and Up} \)? (Hint: first calculate camera coordinate vectors \( u, v, n \) and use them to compute the rotational part of \( M \). Then determine the translational part of \( M \) and combine with the rotational part to get to \( M \). Use homogeneous coordinates.)

\[
\begin{align*}
n &= (\text{Eye} - \text{Look}) / \| (\text{Eye} - \text{Look}) \| \\
u &= (\text{Up} \times n) / \| (\text{Up} \times n) \| \\
v &= n \times u
\end{align*}
\]

\[
\begin{pmatrix}
ux & ux & ux & 0 \\
vv & vv & vv & 0 \\
nz & nz & nz & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

\[
\begin{pmatrix}
1 & 0 & 0 & -\text{eyex} \\
0 & 1 & 0 & -\text{eyey} \\
0 & 0 & 1 & -\text{eyez} \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

\[ M = M_{\text{rot}} \cdot M_{\text{trans}} \]