Week 3 Tutorial Solutions

Consider the following code:

```
  public void display(GL3 gl) {
    super.display();
    CoordFrame2D frame;
    frame = CoordFrame2D.identity();
// #1
// Matrix is the identity:
// [ 1  0  0 ]
// [ 0  1  0 ]
// [ 0  0  1 ]
// Coordinate frame is the world frame

frame = frame.rotate(30);
// #2
// [ 1 0 0 ] [ cos(30)  -sin(30)  0 ]   [ 0.87 -0.5   0 ]
// [ 0 1 0 ] [ sin(30)   cos(30)  0 ] = [ 0.5   0.87  0 ]
// [ 0 0 1 ] [   0         0      1 ]   [ 0     0     1 ]
//
// The origin is (0,0)
// i = (0.87, 0.5)
// j = (-0.5, 0.87)

frame = frame.scale(-2, 2);
// #3
// [ 0.87  -0.5   0 ] [ -2  0  0 ]     [ -1.73  -1     0 ]
// [ 0.5    0.87  0 ] [  0  2  0 ]  =  [ -1      1.73  0 ]
// [ 0      0     1 ] [  0  0  1 ]     [  0      0     1 ]
//
// the origin is still (0,0)
// i = (-1.73, -1)
// j = (-1, 1.73)

frame = frame.translate(1, 0);
// #4
// [ -1.73  -1     0 ] [ 1  0  1 ]    [ -1.73  -1     -1.73 ]
// [ -1      1.73  0 ] [ 0  1  0 ]  = [ -1      1.73  -1    ]
// [  0      0     1 ] [ 0  0  1 ]    [  0      0      1    ]
//
// the origin is now (-1.73, -1)
// i and j are not changed
// i = (-1.73, -1)
// j = (-1, 1.73)

Polygon2D poly = new Polygon2D(
   1, 1,
   // [ -1.73  -1     -1.73 ] [ 1 ]    [ -4.46 ]
   // [ -1      1.73  -1    ] [ 1 ]  = [ -0.27 ]
   // [  0      0      1    ] [ 1 ]    [  1    ]

   1, 2,
   // [ -1.73  -1     -1.73 ] [ 1 ]    [ -5.46 ]
   // [ -1      1.73  -1    ] [ 2 ]  = [  1.46 ]
   // [  0      0      1    ] [ 1 ]    [  1    ]

   2, 1.5f);
   // [ -1.73  -1     -1.73 ] [ 2   ]    [ -6.69 ]
   // [ -1      1.73  -1    ] [ 1.5 ]  = [ -0.41 ]
   // [  0      0      1    ] [ 1   ]    [  1    ]
```

Question 2:

Consider the 2D tranformation matrix:

```
[ 1  -1   1 ]
[ 1   2   2 ]
[ 0   0   1 ]
```

- *Is this matrix affine? Why/why not?*

Yes, the bottom row is 0, 0, 1, so it is an affine matrix.

- *What are the axes of the coordinate frame it represents? What is the origin? Sketch it.*

```
Origin = (1,2)
i = (1,1)  <- pointing up and right
j = (-1,2) <- point up and left
```

- *What is the scale of each of the axes?*

```
|i| = sqrt(2)
|j| = sqrt(5)
```

- *What is the angle of each of the axes?*

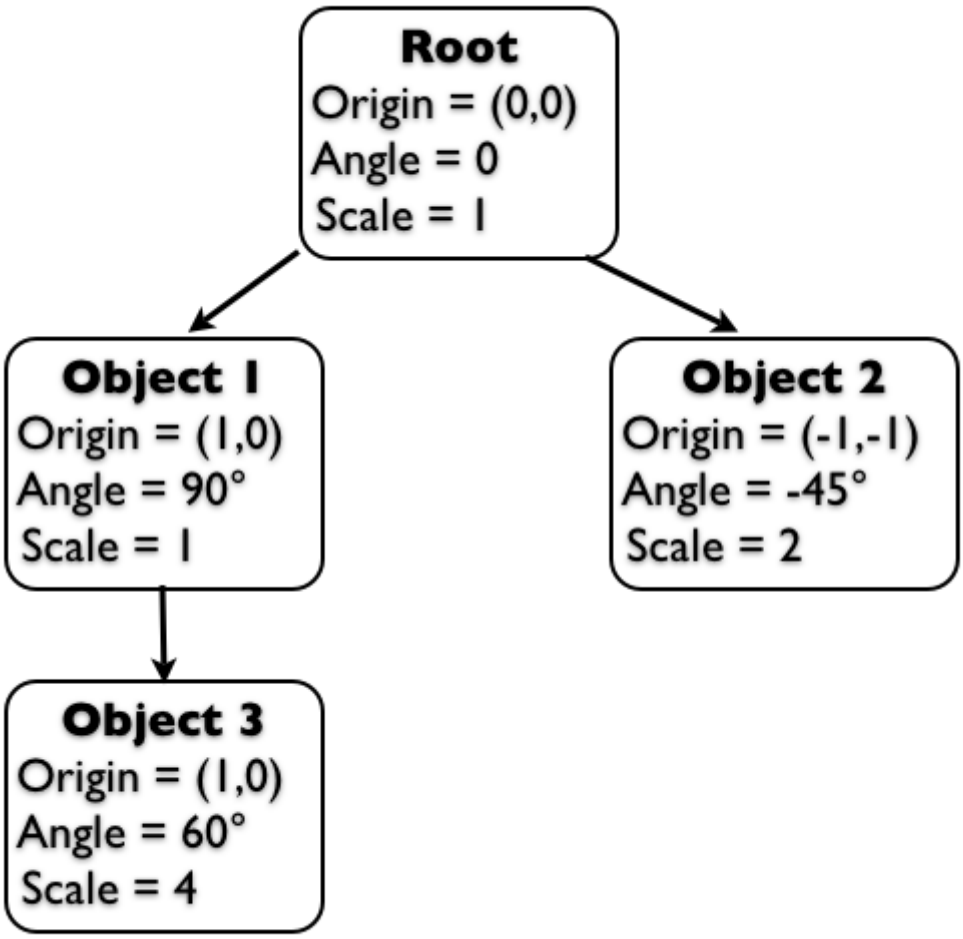
```
angle_i = atan(1 / 1) = 45 degrees
angle_j = atan(2 / -1) = 117 degrees
```

- *Does this transformation have a shear component? Why/why not?*

Yes, the axes are not perpendicular.

Question 3:

Consider the scene graph below:



Note: If this is taking too long and people are already comfortable with matrix multiplication feel free to use matrix multiplier or similar. In an exam you would have to do by hand/normal calculator but you have already done enough matrix mult in question 1 for one tutorial.

- *What is the model matrix for each node?*

```
Root:
=====
[ 1 0 0 ]
[ 0 1 0 ]
[ 0 0 1 ]

Object 1:
=========
M_1 = M_root T_1 R_1 S_1

      [ 1 0 0 ] [ 1 0 1 ] [ 0 -1  0 ] [ 1 0 0 ]
    = [ 0 1 0 ] [ 0 1 0 ] [ 1  0  0 ] [ 0 1 0 ]
      [ 0 0 1 ] [ 0 0 1 ] [ 0  0  1 ] [ 0 0 1 ]

      [ 0  -1  1 ]
    = [ 1   0  0 ]
      [ 0   0  1 ]

Origin_1 = (1,0)
i_1 = (0,1)
j_1 = (-1,0)

Object 2:
=========
M_2 = M_root T_2 R_2 S_2
```

```
      [ 1 0 0 ] [ 1 0 -1 ] [ cos(-45)  -sin(-45)  0 ] [ 2 0 0 ]
    = [ 0 1 0 ] [ 0 1 -1 ] [ sin(-45)   cos(-45)  0 ] [ 0 2 0 ]
      [ 0 0 1 ] [ 0 0  1 ] [    0          0      1 ] [ 0 0 1 ]

      [  1.41  1.41  -1 ]
    = [ -1.41  1.41  -1 ]
      [  0     0      1 ]

Origin_2 = (-1, -1)
i_2 = (1.41, -1.41)
j_2 = (1.41,  1.41)

Object 3:
=========
M_3 = M_1 T_3 R_3 S_3

      [ 0  -1  1 ] [ 1  0  1 ] [ cos(60)  -sin(60)  0 ] [ 4  0  0 ]
    = [ 1   0  0 ] [ 0  1  0 ] [ sin(60)   cos(60)  0 ] [ 0  4  0 ]
      [ 0   0  1 ] [ 0  0  1 ] [ 0         0        1 ] [ 0  0  1 ]

      [ -3.464 -2.000  1.000 ]
    = [  2.000 -3.464  1.000 ]
      [  0.000  0.000  1.000 ]

Origin_3 = (1,1)
i_3 = (-3.464, 2)
j_3 = (-2, -3.464)
```

- *If Object 3 has its parent changed to Object 2* ***without*** *changing its local origin, angle or scale, how does its coordinate frame change?*

```
Object 3:
=========

Now:
M_3 = M_2 T_3 R_3 S_3
      [  1.41  1.41  -1 ] [ 1  0  1 ] [ cos(60)  -sin(60)  0 ] [ 4  0  0 ]
    = [ -1.41  1.41  -1 ] [ 0  1  0 ] [ sin(60)   cos(60)  0 ] [ 0  4  0 ]
      [  0     0      1 ] [ 0  0  1 ] [ 0         0        1 ] [ 0  0  1 ]

// More math. Bleh.

      [ 7.704 -2.064  1.820 ]
    = [ 2.064  7.704 -1.000 ]
      [ 0      0      1     ]

Origin_3 = (1.82, -1)  <-- origin has moved
i_3 = (-7.7, 2.06)     <-- axes are stretch and rotated
j_3 = (-2.06, 7.7)
```

- *If we want to preserve Object 3's original coordinate frame, what new values do we need to set for its origin, angle and scale?*

```
Before
======
global_origin_3 = M_1 * local_origin_3 = (1,1), as above
global_angle_3 = local_angle_3 + local_angle_1 + local_angle_root = 150 degrees
global_scale_3 = local_scale_3 * local_scale_1 * local_scale_root = 4

After
=====
We want to preserve these values so:

global_origin_3 = M_2 * local_origin_3
                = M_root * T_2 * R_2 * S_2 * local_origin_3

local_origin_3 = M_2 ^ -1 * global_origin_3
```

```
              = S_2 ^ -1 * R_2 ^ -1 * T_2 ^ -1 * M_Root^-1 * (1,1)

                [ 0.5  0    0 ] [ cos(45)  -sin(45)  0 ] [ 1 0 1 ] [ 1 0 0 ] [ 1 ]
              = [ 0    0.5  0 ] [ sin(45)   cos(45)  0 ] [ 0 1 1 ] [ 0 1 0 ] [ 1 ]
                [ 0    0    1 ] [    0        0      1 ] [ 0 0 1 ] [ 0 0 1 ] [ 1 ]


                [ 0.354 -0.354  0.000 ] [ 1 ]
              = [ 0.354  0.354  0.708 ] [ 1 ]
                [ 0      0      1     ] [ 1 ]

                [ 0.000 ]
              = [ 1.416 ]
              = [ 1.000 ]

So the new local origin is (0.000, 1.416)

global_angle_3 = local_angle_3 + local_angle_2 + local_angle_root
          150 = local_angle_3 - 45
local_angle_3 = 195

global_scale_3 = local_scale_3 * local_scale_2 * local_scale_root
             4 = local_scale_3 * 2
local_scale_3 = 2
```

- *If a camera with a local origin of 0,0, rotation angle of 0 and scale of 2 was attached to Object 2 in the scene graph, what would the view matrix contain after setting the view for the camera?*

```
Camera:
=========
M_C = M_2 T_C R_C S_C

      [  1.41  1.41  -1 ] [1 0 0 ] [1 0 0 ] [ 2 0 0 ]
    = [ -1.41  1.41  -1 ] [0 1 0 ] [0 1 0 ] [ 0 2 0 ]
      [  0     0      1 ] [0 0 1 ] [0 0 1 ] [ 0 0 1 ]

    = [  2.82  2.82  -1 ]
      [ -2.82  2.82  -1 ]
      [  0.0   0.0    1 ]

Origin_C = (-1, -1)
i_2 = (2.82, -2.82)
j_2 = (2.82,  2.82)

global_angle_C = local_angle_C + local_angle_2 + local_angle_root =  0 -45 + 0 = -45
global_scale_C = local_scale_C * local_scale_2 * local_scale_root =  2 * 2 * 1 = 4

Therefore we would create a matrix with inverse global
scale multiplied by inverse global rotation multiplied
by the inverse global origin.

VIEW MATRIX: = INVERSE_CAMERA_GLOBAL_SCALE *
               INVERSE_CAMERA_GLOBAL_ROTATION *
               INVERSE_CAMERA_GLOBAL_TRANSLATION

[0.25 0     0] [ 0.7 -0.7  0 ] [ 1 0 1 ]
[0    0.25  0] [ 0.7  0.7  0 ] [ 0 1 1 ]
[0    0     1] [   0    0  1 ] [ 0 0 1 ]

= [ 0.175 -0.175  0    ]
  [ 0.175  0.175  0.35 ]
  [ 0         0   1    ]
```

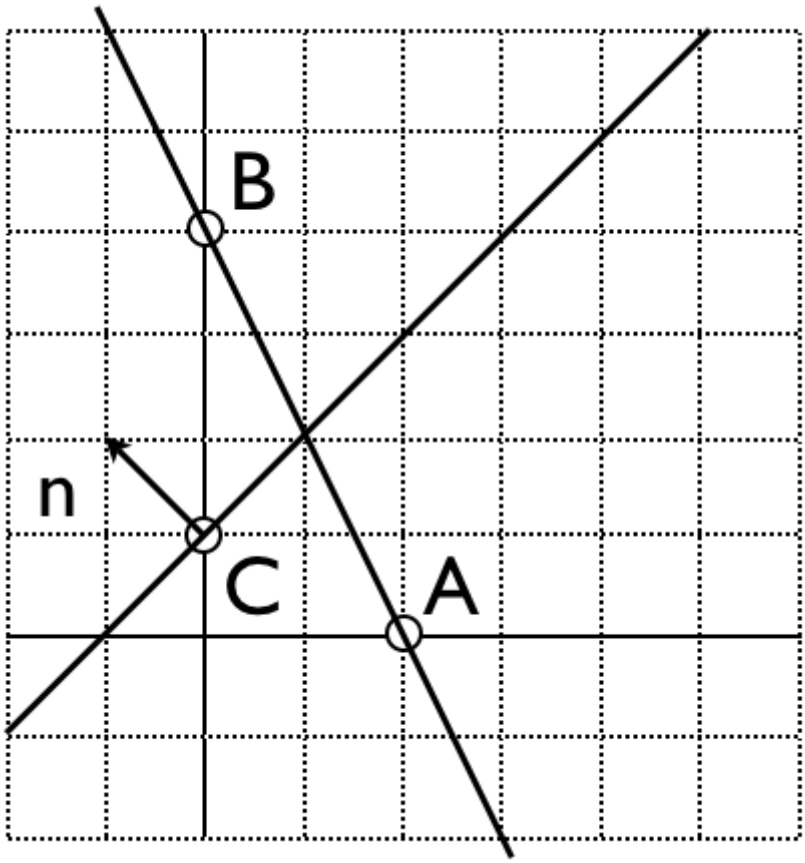
Question 4:

Sketch the lines described by these equations:

1. *L(t) = A + (B-A)t, where A = (2,0), B=(0,4)*
2. ***n**. (C - L) = 0, where **n** = (-1,1), C = (0,1)*

Where do they intersect?

*If Line 1 is a ray starting at A and line 2 is an edge of a polygon (with **n** pointing outwards), is the ray entering or exiting the polygon?*



To compute intersection:

```
n . (C - L(t)) = 0
[ -1 ] . ( [ 0 ] - [ 2 ] - [ 0 - 2 ] t )  = 0
[  1 ]   ( [ 1 ]   [ 0 ]   [ 4 - 0 ]   )

[ -1 ] . [ -2 + 2t ]  = 0
[  1 ]   [  1 - 4t ]

-1 * (-2 + 2t)  + 1 * (1-4t) = 0

2 - 2t + 1 - 4t = 0
3 - 6t = 0
t = 1/2

L(1/2) = A + (B-A) * 0.5
       = [ 1 ]
         [ 2 ]
```

The ray is emerging from the polygon as:

```
   v = (B-A) = (-2, 4)
n.v = (-1, 1) . (-2, 4)
```

```
= 2 + 4
> 0
```

Question 5:

- Look at the shaders vertex_2d.glsl and fragment_2d.glsl in UNSWgraph and make sure you understand what they are doing.
- Modify them so any fragments with an x coordinate of > 0 in global coordinates is red, but white otherwise.

```
//Vertex shader
in vec2 position;

uniform mat3 model_matrix;

uniform mat3 view_matrix;

out vec3 globalPosition;

void main() {
    // The global position is in homogenous coordinates
    globalPosition = model_matrix * vec3(position, 1);

    // The position in camera coordinates
    vec3 viewPosition = view_matrix * globalPosition;

    // We must convert from a homogenous coordinate in 2D to a homogenous
    // coordinate in 3D.
    gl_Position = vec4(viewPosition.xy, 0, 1);
}

//Fragment shader
out vec4 outputColor;

uniform vec3 input_color;

in vec3 globalPosition;

void main()
{
    if (globalPosition.x > 0)
        outputColor = vec4(1, 0, 0, 1);
    else
        outputColor = vec4(1, 1, 1, 1);
}
```

- Modify them so any fragments with an x coordinate of > 0 in camera coordinates is red, but white otherwise.

```
//Vertex shader
in vec2 position;

uniform mat3 model_matrix;

uniform mat3 view_matrix;

out vec3 viewPosition;

void main() {
    // The global position is in homogenous coordinates
    vec3 globalPosition = model_matrix * vec3(position, 1);

    // The position in camera coordinates
    viewPosition = view_matrix * globalPosition;

    // We must convert from a homogenous coordinate in 2D to a homogenous
    // coordinate in 3D.
    gl_Position = vec4(viewPosition.xy, 0, 1);
```

```
}

//Fragment shader
out vec4 outputColor;

uniform vec3 input_color;

in vec3 viewPosition;

void main()
{
    if (viewPosition.x > 0)
        outputColor = vec4(1, 0, 0, 1);
    else
        outputColor = vec4(1, 1, 1, 1);
}
```