Week 4 Tutorial Solutions

Question 1: 3D Affine Transformations

What coordinate frames do the following affine matrices represent?

```
M1 = [[  1,  0,  0,  0],
      [  0,  1, -1,  0],
      [  0,  1,  1,  0],
      [  0,  0,  0,  1]]

M2 = [[  2,  0,  0,  2],
      [  0, -1,  0,  1],
      [  0,  0,  1,  0],
      [  0,  0,  0,  1]]

M3 = [[  1,  1,  1,  0],
      [  0,  1,  1,  0],
      [  0,  0,  1,  0]
      [  0,  0,  0,  1]]
```

Hint: rememeber M = [**i**, **j**, **k**, **phi**].

```
M1 = [[  1,  0,  0,  0],
      [  0,  1, -1,  0],
      [  0,  1,  1,  0],
      [  0,  0,  0,  1]]

i = (1,0,0)
j = (0,1,1)
k = (0,-1,1)
phi = (0,0,0)

The j/k axes have been rotated 45 degrees about i, and scales to have length sqrt(2).

M2 = [[  2,  0,  0,  2],
      [  0, -1,  0,  1],
      [  0,  0,  1,  0],
      [  0,  0,  0,  1]]

i = (2,0,0)
j = (0,-1,0)
k = (0,0,1)
phi = (2,1,0)

The origin has been translated to (2,1,0).
The i-axis has been scaled by 2.
The j-axis has been reflected.

M3 = [[  1,  1,  1,  0],
      [  0,  1,  1,  0],
      [  0,  0,  1,  0]
      [  0,  0,  0,  1]]

i = (1,0,0)
j = (1,1,0)
k = (1,1,1)
phi = (0,0,0)

The j-axis is sheared towards the i-axis and scaled by sqrt(2).
The k-axis is sheared towards both the i-axis and the j-axis and scaled by sqrt(3).
The axes are not at right angles to each other. We can check this by using the dot product between the pairs of axes.
```

Question 2: 3D Rotations

If you rotate a coordinate frame by 90 degrees about x and then rotate the resulting frame by 90 degrees about y, what is the resulting frame? What is the matrix for the combined transformation? What happens if you reverse the order of the rotations?

```
M = R_x R_y
  =  [1 0  0 0]  [0  0 1 0]
     [0 0 -1 0]  [0  1 0 0]
     [0 1  0 0]  [-1 0 0 0]
     [0 0  0 1]  [0  0 0 1]

  = [0 0 1 0]
    [1 0 0 0]
```

```
    [0 1 0 0]
    [0 0 0 1]

i-axis is (0,1,0)
j-axis is (0,0,1)
k-axis is (1,0,0)
i.e. the three axes have been rotated. i->j, j->k, k->i


in the opposite order:

M = R_y R_x
  = [ 0 0 1 0] [1 0  0 0]
    [ 0 1 0 0] [0 0 -1 0]
    [-1 0 0 0] [0 1  0 0]
    [ 0 0 0 1] [0 0  0 1]

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

i-axis is (0,0,-1)
j-axis is (1,0,0)
k-axis is (0,-1,0)
```

Question 3: Winding Order and Culling

Consider the following code for drawing a pyramid.

Which way are the faces pointing? If you drew this in OpenGL with back face culling turned on, what would you see?

```
    // the base
    TriangleFan3D base = new TriangleFan3D(
        1, 1, 0,
        -1, 1, 0,
        -1,-1, 0,
        1, -1, 0);
    base.draw();

    // the sides
    Triangle3D side1 = new Triangle3D(
        0, 0, 2,
        1, 1, 0,
        -1, 1, 0);
    side1.draw();

    Triangle3D side2 = new Triangle3D(
        0, 0, 2,
        -1, -1, 0,
        -1, 1, 0);
    side2.draw();

    Triangle3D side3 = new Triangle3D(
        0, 0, 2,
        -1, -1, 0,
        1, -1, 0);
    side3.draw();

    Triangle3D side4 = new Triangle3D(
        0, 0, 2,
        1, -1, 0,
        1, 1, 0);
    side4.draw();
```

i) The base is pointing inwards.

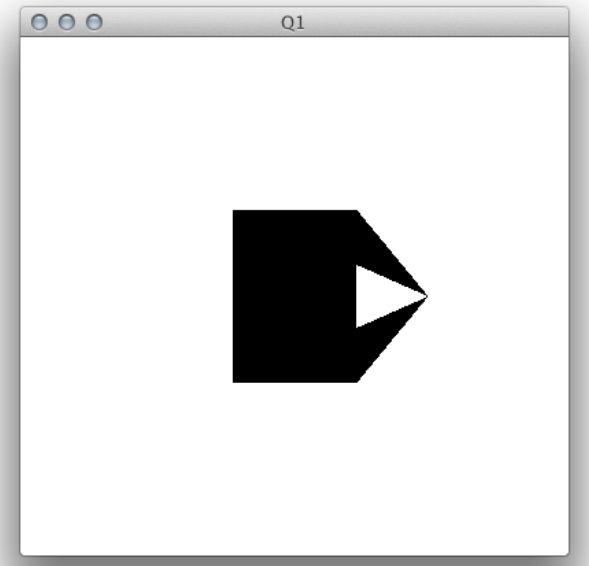
ii) The first side is pointing outwards

iii) The second side is pointing inwards.

iv) The third side is pointing outwards.

v) The fourth side is pointing outwards.

The image below shows view facing the second face (which is invisible). You can see the first and second triangle faces (pointing outwards) and the base (pointing inwards).



If you look at the image from behind everything disappears.

Question 4: Perspective Projections

Consider the following code:

```
Matrix4 proj = Matrix4.perspective(60, 1, 1, 10);
Shader.setProjMatrix(gl, proj);

TriangleFan3D quad1 = new TriangleFan3D(
    1, 1, -2,
    0, 1, -2,
    0, 0, -2,
    1, 0, -2);
quad1.draw();

TriangleFan3D quad2 = new TriangleFan3D(
    0, 1, -4,
    -1, 1, -4,
    -1, 0, -4,
    0, 0, -4);
quad2.draw();
```

- Sketch the view volume.

  The view volume is a frustum with a 60 degree fov. The near plane is a square (aspect=1) with side length 2/sqrt(3) centred at (0,0,-1) in camera coordinates. The far plan is a square with side length 20/sqrt(3) cented at (0,0,-10) in camera coordinates

- What will be displayed on the screen?
  Two squares side by side. The square on the right will be twice as big (both width and height) as the one on the left.
- What is the projection transform matrix?

```
R = [ 2n / (r-l)     0          (r+l)/(r-l)     0 ]
    [ 0              2n/(t-b)   (t+b)/(t-b)     0 ]
    [ 0              0          -(f+n)/(f-n)  -2fn/(f-n) ]
    [ 0              0          -1             0 ]

  = [ sqrt(3)      0            0             0 ]
    [ 0            sqrt(3)      0             0 ]
    [ 0            0           -11/9         -20/9 ]
    [ 0            0           -1             0 ]
```

- What do we get when we multiply the point (1, 1, -2), in camera co-ordinates by this matrix and what will it be equivalent to after perspective division (assuming it survives clipping).

```
(sqrt(3),sqrt(3),2/9,2)
which after perspective division would be
(sqrt(3)/2,sqrt(3)/2,1/9)
```

- What happens if the field of view is changed to 90 degrees? 180 degrees? 240 degrees?

  As the FOV gets larger the squares shrink towards the middle of the screen. At 180 degress they disappear. A FOV greater than 180 degrees is nonsensical.

Question 5 (advanced - not examinable):

A rotation about any arbitrary axis can be decomposed into a series of rotations about the X, Y or Z axes. How would you compute such a decomposition? (There are many different ways of doing this.)

See http://en.wikipedia.org/wiki/Euler_angles