CO1301: Games Concepts
Lab 14 - Matrices: Rotation

1. Introduction

1. This worksheet will introduce the idea of using matrices to rotate models. Make sure that you have finished the lab from Week 13 before attempting this lab since it relies upon understanding the idea of a matrix associated with each model.
2. This lab uses floating point arrays to represent a model's matrix.
3. If it is not clear what the functions and arrays used in this lab are doing, please ask your lab tutor for clarification.

2. Setting Up

1. Create a new TL-Engine project called "Lab14_RotationTransformation_Project".
2. Load the floor mesh and create a floor model at the origin.
3. Load the cube mesh and create a cube model at the (0,10,30).
4. Define a constant called kPi for π, assign it a value of 3.1415926f.

```
const float kPi = 3.1415926f;
```

3. Rotation About the World X-Axis

1. Each model has a 4x4 matrix associated with it, this means that you can directly manipulate the matrix in order to perform transformation operations on the model.
2. The matrix for a model is obtained using the GetMatrix() method. The following code gets the matrix of the cube and places it into the 4x4 array:

```
float matrix[4][4];
cube->GetMatrix( &matrix[0][0] );
```

3. The matrix can be then be manipulated in whatever way we see fit. After the matrix has been changed the new matrix can be assigned to the model using the SetMatrix() method. The following code sets the matrix of cube:

```
cube->SetMatrix( &matrix[0][0] );
```

4. The first row of the matrix is its local x-axis:
   - matrix[0][0] is the x component of the x-axis
   - matrix[0][1] is the y component of the x-axis
   - matrix[0][2] is the z component of the x-axis

   The second row of the matrix is local y-axis and the third row of the matrix is local z-axis.

5. The following function will rotate a model around the world x-axis and the local origin of the model.

```
// Rotate around the World x-axis and the local origin.
//  float rotation : angle to rotate in degrees
//  matrix[4][4] : model to be rotated
void rotateX( float matrix[4][4], float rotation )
{
    // convert the rotation from degrees to radians
    float alpha = rotation * kPi / 180.0f;
    // find the sine of the angle
    float sinAlpha = sin( alpha );
    // find the cosine of the angle
    float cosAlpha = cos( alpha );

    // Need to use temporary values because x and y get overwritten during the
    // course of the calculations (only one is actually needed - see the maths
    // classes in year 2, but the code is clearer this way)
    float tempY;
    float tempZ;

    // Perform the minimum of calculations rather than use full matrix
    // multiplication. A rotation around the world x-axis only causes the y
    // and z components of each of the locals axes to be changed.
    // See the lecture slides.
```

```
        // In this version of the rotation we want the coordinates of the model
        // to be left unchanged so the final row is ignored.

        // Firstly rotate the local x-axis (first row)
        tempY = matrix[0][1] * cosAlpha - matrix[0][2] * sinAlpha;
        tempZ = matrix[0][1] * sinAlpha + matrix[0][2] * cosAlpha;
        matrix[0][1] = tempY;
        matrix[0][2] = tempZ;

        // Rotate the local y-axis (second row)
        tempY = matrix[1][1] * cosAlpha - matrix[1][2] * sinAlpha;
        tempZ = matrix[1][1] * sinAlpha + matrix[1][2] * cosAlpha;
        matrix[1][1] = tempY;
        matrix[1][2] = tempZ;

        // Rotate the local z-axis (third row)
        tempY = matrix[2][1] * cosAlpha - matrix[2][2] * sinAlpha;
        tempZ = matrix[2][1] * sinAlpha + matrix[2][2] * cosAlpha;
        matrix[2][1] = tempY;
        matrix[2][2] = tempZ;
    }
```

Copy the function into your code and then use it to rotate the cube.

6. The code uses the rotation matrix (RotateAboutX) provided in lecture slides.

7. Examine the code in conjunction with the rotation matrix. What is happening is that the matrix is being applied to each of the local axes in turn: first the local x-axis, then the local y-axis, and finally the local z-axis. Remember that the rows of the first matrix are combined with the columns of the second matrix. Pair up the components and multiply them together and then sum the result.

    - The result of applying the matrix to the local x-axis is that y and z components are changed.
    - The result of applying the matrix to the local y-axis is that y and z components are changed.
    - The result of applying the matrix to the local z-axis is that y and z components are changed.
    - The net effect is that column 1 and column 2 are changed.
    - We want to leave the position of the model unchanged and so the last row of the matrix can be ignored.

4. Rotation About the World Y-Axis

1. Using the code from the previous section as your starting point, implement a rotation around the world y-axis. Use the rotation matrix (RotateAboutY) provided in lecture slides.

2. Try to perform the matrix multiplication yourself. The net effect of the matrix multiplication will be:

    - The x and z components of the local x-axis are changed.
    - The x and z components of the local y-axis are changed.
    - The x and z components of the local z-axis are changed.
    - The net effect is that column 0 and column 2 are changed.
    - We want to leave the position of the model unchanged and so the last row of the matrix can be ignored

5. Rotation About the World Z-Axis

1. Now, implement a rotation around the world z-axis. Use the rotation matrix (RotateAboutZ) provided in lecture slides.

2. Try to perform the matrix multiplication yourself. The net effect of the matrix multiplication will be:

    - The x and y components of the local x-axis are changed.
    - The x and y components of the local y-axis are changed.
    - The x and y components of the local z-axis are changed.
    - The net effect is that column 0 and column 1 are changed.
    - We want to leave the position of the model unchanged and so the last row of the matrix can be ignored