Graphics – Tutorial for Part 2, Week 1

Introduction

The goal of the exercises this week is to start to get you used to working with DirectX code. The starting point for all of the exercises this week should be the example provided for you in DirectX_Cube_Wireframe.zip.

Exercise 1

The code in the example program currently just positions the cube at the origin and shows the front face of the cube. The world transformation matrix (_worldTransformation – defined in DirectXApp.h) is simply initialised to the Identity matrix and never changes. For these exercises, you should modify the code so that it animates the cube. Most of your code changes will go in the Update method in DirectXApp.cpp, but you will almost certainly need to add one or more member variables to the class in DirectXapp.h.

To do these exercises, you will need to store a transformation matrix in _worldTransformation or multiply two or more matrices together and store the result in _worldTransformation. The transformation matrices can be generated by calls to the following static methods in the SimpleMath Matrix class (see descriptions at https://github.com/microsoft/DirectXTK/wiki/Matrix :

- Matrix::CreateTranslation
- Matrix::CreateScale
- Matrix::CreateRotationX
- Matrix::CreateRotationY
- Matrix::CreateRotationZ

The first two methods use a Vector3 object as a parameter that specifies the X, Y and Z values. The rotation matrices take the angle (in radians) as a parameter (when converting from degrees to radians, you might find the constant XM_PI useful as the value of Pi).

When multiplying matrices together, the left-hand matrix is the first one that will be applied when doing the transformation. For example, if you do:

```
_worldTransformation = r * s;
```

where r and s are Matrix objects, when the world transformation is performed, the transformation represented by matrix r will be applied first, followed by the transformation represented by matrix s. If you do:

```
_worldTransformation = s * r;
```

you are likely to get a different result.

If you are not sure what is being asked for in these exercises, I have recorded small videos of what I would expect to see in the DirectX window in the Panopto lecture recordings folder on Course Resources.

As a lot of this is new, it is quite possible that you will need to ask for help on some of these exercises. Please make sure you do so – that is what the practical sessions are for.

Exercise 1a

Update the program so that the cube rotates one degree around the Z axis each frame.

Exercise 1b
Update the program so that the cube rotates one degree around the Y axis each frame.

Exercise 1c
Update the program so that the cube rotates one degree around the Y axis each frame and slowly moves towards and off the right-hand side of the window

Exercise 1d
Update the program so that the cube rotates one degree around the Z axis each frame, but is always 2 units away from the origin (i.e. it does a wide circle around the Z axis).

If you have got this far, experiment with rotations around all axes and try the scaling transformation as well so that you are familiar with how they work. Try combing more than two matrices to see the impact.

Exercise 2
For this exercise, modify the program so that it draws an additional cube above the first one. The top cube show rotate around the Y axis in one direction and the bottom cube should rotate around the Y axis in the opposite direction.

Think carefully about what lines need to be added in order to do this. You will be surprised how few extra lines it takes to draw the additional cube.

Exercise 3
Experiment with changing the eye position (_eyePosition) and look at position (_focalPointPosition) vectors to change the camera position. For example, can you change the camera so that it:

a) Looks diagonally down at the cubes from the front, but at a higher position
b) Looks vertically down on to the two rotating cubes.

What is expected
The following videos show what is expected from exercise 1 and 2:

Exercise 1a

Exercise 1b

Exercise 1c

Exercise 1d

Exercise 2