CO1301: Games Concepts
Lab 13 - Matrices: Scaling

1. Introduction

1. This lab will introduce the idea of using matrices to manipulate models.
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 "Lab13_MatrixScaling_Project".
2. Load the cube mesh and create a cube model at (0, 5, 30).
3. Load the floor mesh and create a floor model at the origin.
4. Each model has a 4x4 matrix associated with it, this means that you can directly manipulate the matrix in order to perform graphics operations on the model.
5. 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] );
```

6. 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] );
```

   Note: The use of the ampersand (&) in this context means "obtain the address of". The matrix associated with the model is a single dimensional floating point array of size 16, i.e. float m[16]. However, I think it'll be easier for you to visual the matrix as a two-dimensional array of size 4 by 4, i.e. float n[4][4]. In practice the two array declarations are treated identically by the compiler. But in order to convert from the one format to other it is necessary to pass the base address of the 4x4 matrix over to the function and this is what the ampersand allows us to do: it states the address of the first element of the array is passed over to the function.

7. The last row of the matrix is the position of the model:
   - matrix[3][0] is the x-coordinate
   - matrix[3][1] is the y-coordinate
   - matrix[3][2] is the z-coordinate
8. So if the value of matrix[3][0] is changed and the matrix assigned to cube model then the cube will move along the x-axis. Note that the other elements of the matrix must be left unaltered else the cube will be transformed in other ways and not just moved along the x-axis
9. Place the following code within the game loop to see this in operation. You should see the cube move to the right

```
//declare a 4x4 matrix
float matrix[4][4];

//assign the cube's matrix to the declared array
cube->GetMatrix(&matrix[0][0]);

//update the matrix element that stores the x-position of the cube
matrix[3][0] += 0.001f;

//assign the updated matrix to the cube
cube->SetMatrix(&matrix[0][0]);
```

3. Moving Models by Matrix Manipulation

1. Using the code in 2.8 as the basis of your work, set up your program so that holding the D key moves the cube in the positive x (to the right), whilst holding the A key moves it in the negative x (to

the left).

2. Extend the code so that holding the W key moves the cube in the positive y (upwards), whilst holding the S key moves it in the negative y (downwards).

4. Scaling Models by Matrix Manipulation

1. In this section, you will use a model's matrix to scale it.
2. Let us start with scaling the model just along the x-axis. In other words, you will stretch the model out at the sides.
3. 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
4. In order to scale the cube along the x-axis, all three components of its local x-axis need to be modified, for example:

```
matrix[0][0] *= 1.001f;
matrix[0][1] *= 1.001f;
matrix[0][2] *= 1.001f;
```

5. Using this code set up your program so that holding the E key scales the cube upwards along its x-axis.
6. Now set up your program so that holding the Q key scales the cube downwards along its x-axis.
7. What happens if you only adjust one of the components? In other words, what happens if only matrix[0][0] is changed whilst matrix[0][1] and matrix[0][2] are left alone? Change the code so that only matrix[0][0] is modified and then run it again. You will almost certainly not notice any difference. In order to see what is going wrong you will have to rotate the cube.
8. Set up the O key to rotate the cube around the local y-axis. Use RotateLocalY(). Try scaling the cube in and out as the cube is rotated. You should see the model start to do strange things. Can you work out why this is happening?
9. Set your scaling code back to its proper state, i.e. so that all three components of the x-axis are scaled by the same amount. If you now rotate the cube you will see that everything works fine once again.
10. Now set up your program so that holding the "+" key scales the cube upwards along all three axes, whilst holding the "-" key scales it downwards.

5. Advanced - Scaling Function

1. The TL-Engine has a Scale() method which takes a floating point value as its input and scales a model according to this value. Write the equivalent function to this, your function should take two parameters: the model to be scaled and the scaling factor. The function prototype looks like this:

```
void scaling( IModel* model, float factor);
```