3D Graphics and Matrix Manipulation

Module Syllabus

3D Graphics

Matrices: Model, View, and Projection

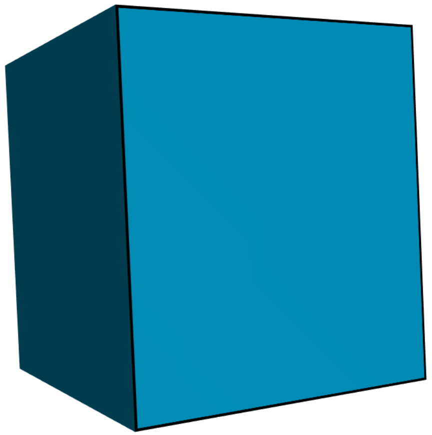
Matrix Manipulation

- Addition
- Multiplication
- Identity Matrix
- Scaling
- Rotation
- Projection

3D Graphics

3D computer graphics are graphics that make use of the third dimension, conveying depth to display models, etc.

Earlier material showed how to create a 2D triangle using a vertex array, which defined the X and Y coordinates for each vertex. When defining a 3D object, a value for the Z axis must also be defined.



Creating 3D graphics is generally done in three stages:

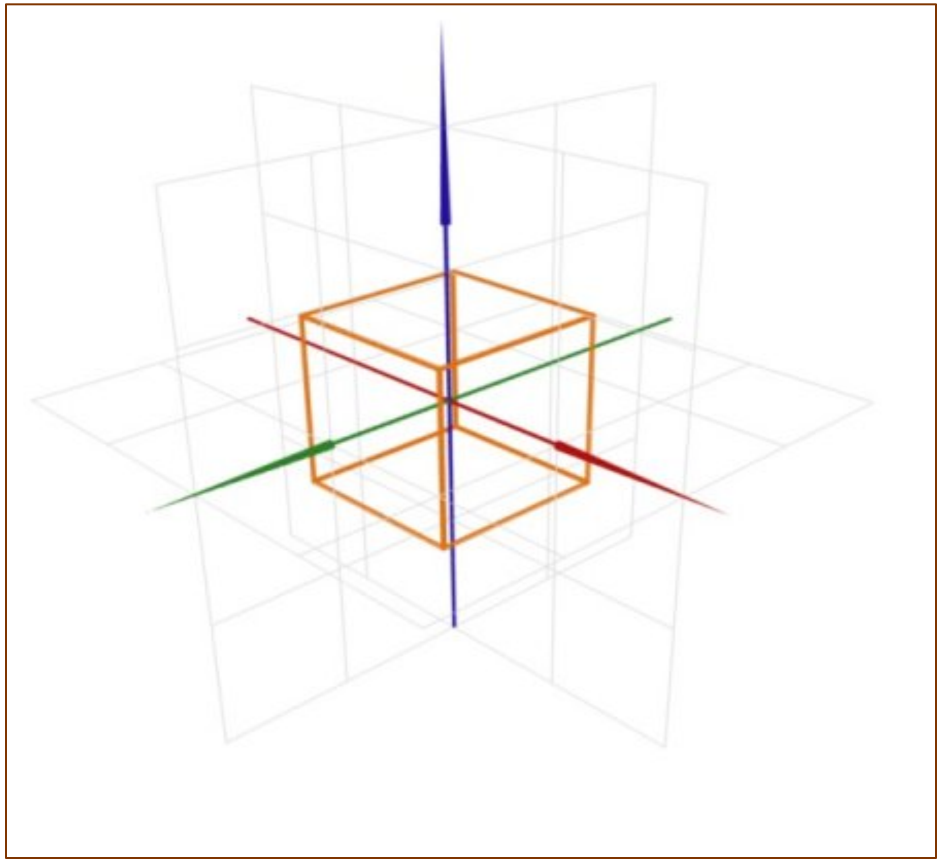
- Modelling
- Animation
- Rendering

A 3D Object

Begin with a number of vertices that define the points of the object.

These defined points are within an XYZ coordinate space.

The points are linked into a collection of primitives, commonly triangles, that represent the shape.



Cube: Vertex Array

```
GLfloat cubeVertices[] = {-1.0f,  1.0f, -1.0f, /* Back. */
                           1.0f,  1.0f, -1.0f,
                          -1.0f, -1.0f, -1.0f,
                           1.0f, -1.0f, -1.0f,
                          -1.0f,  1.0f,  1.0f, /* Front. */
                           1.0f,  1.0f,  1.0f,
                          -1.0f, -1.0f,  1.0f,
                           1.0f, -1.0f,  1.0f,
                          -1.0f,  1.0f, -1.0f, /* Left. */
                          -1.0f, -1.0f, -1.0f,
                          -1.0f, -1.0f,  1.0f,
                          -1.0f,  1.0f,  1.0f,
                           1.0f,  1.0f, -1.0f, /* Right. */
                           1.0f, -1.0f, -1.0f,
                           1.0f, -1.0f,  1.0f,
                           1.0f,  1.0f,  1.0f,
                          -1.0f, -1.0f, -1.0f, /* Top. */
                          -1.0f, -1.0f,  1.0f,
                           1.0f, -1.0f,  1.0f,
                           1.0f, -1.0f, -1.0f,
                          -1.0f,  1.0f, -1.0f, /* Bottom. */
                          -1.0f,  1.0f,  1.0f,
                           1.0f,  1.0f,  1.0f,
                           1.0f,  1.0f, -1.0f
                         };
```

This code defines the vertex array for a cube.

What is the difference between this and the triangle vertex array?

The former is much larger, because four vertices must be defined for each face of the cube, as well as an extra Z coordinate per vertex.

3D Objects: Transformations

A number of transformations can be performed to make objects move, rotate etc.

This is done via matrix manipulation.

What is a Matrix?

A matrix is an array of numbers typically displayed in the form of a rectangle or square.

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 10 & 11 & 12 & 13 \end{bmatrix}$$

The example above is of a 3x4 matrix, consisting of three rows and four columns.

Each number in the matrix is called an element.

Types of Matrices

In graphics development, there are three main matrices per application:

- Model matrix
- View matrix
- Projection matrix

Model Matrix

States where the object should be drawn on the screen

Useful for when drawing more than one object

Common tasks performed on the model matrix

- Movement
- Scaling
- Rotation

View Matrix

The view matrix handles any movement of the camera.

This alters how the scene is viewed.

For example, in a scene with buildings, the user doesn’t want to have to move them all, so instead moves the camera (view matrix) to show what is wanted.

The Projection Matrix

The projection matrix handles the depth of the scene.

Objects that are closer to the camera are made to look bigger, and objects further away are made to look smaller.

Identity Matrix

An identity matrix (In) is an n x n matrix, with a diagonal line of 1s from the top left to the bottom right, and with all other elements set as 0:

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Multiplying any matrix by the identity matrix, will provide a result of the same matrix that the identity matrix is multiplied by. This assumes that it abides by the rule that the matrix has the same number of columns in A relative to rows in B.

Matrix Addition

Matrix addition is very simple; simply add the corresponding elements to obtain the result element in the final matrix.

$$\begin{bmatrix} 4 & 8 \\ 5 & 1 \end{bmatrix} + \begin{bmatrix} 3 & 2 \\ 6 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 4+3 & 8+2 \\ 5+6 & 1+1 \end{bmatrix} = \begin{bmatrix} 7 & 10 \\ 11 & 2 \end{bmatrix}$$

Matrix Multiplication

Multiply the elements of a row in A with the corresponding column in B and add up the results.

The number of columns in the first matrix must equal the number of rows in the second matrix.

An example of matrix multiplication is shown on the next slide.

Matrix Multiplication

$$\begin{bmatrix} 3 & 4 & 1 \\ 1 & 7 & 3 \\ 5 & 6 & 2 \end{bmatrix} * \begin{bmatrix} 7 & 1 & 9 \\ 2 & 3 & 4 \\ 5 & 1 & 6 \end{bmatrix}$$

$$\begin{bmatrix} 3*7+4*2+1*5 & 3*1+4*3+1*1 & 3*9+4*4+1*6 \\ 1*7+7*2+3*5 & 1*1+7*3+3*1 & 1*9+7*4+3*6 \\ 5*7+6*2+2*5 & 5*1+6*3+2*1 & 5*9+6*4+2*6 \end{bmatrix} = \begin{bmatrix} 34 & 16 & 49 \\ 36 & 25 & 55 \\ 57 & 25 & 81 \end{bmatrix}$$

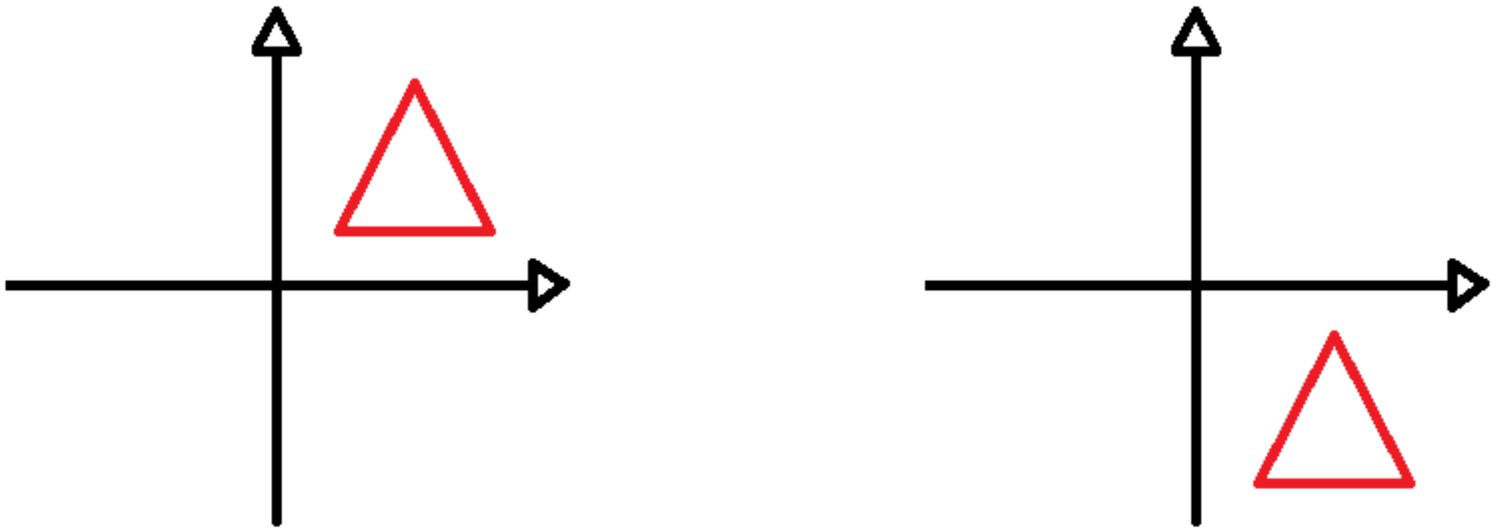
Multiplication: Code Example

```
void matrixMultiply(float* destination, float* operand1, float* operand2)
{
    float theResult[16];
    int row, column = 0;
    int i,j = 0;
    for(i = 0; i < 4; i++)
    {
        for(j = 0; j < 4; j++)
        {
            theResult[4 * i + j] = operand1[j] * operand2[4 * i] + operand1[4 + j] * operand2[4 * i + 1] +
                operand1[8 + j] * operand2[4 * i + 2] + operand1[12 + j] * operand2[4 * i + 3];
        }
    }

    for(int i = 0; i < 16; i++)
    {
        destination[i] = theResult[i];
    }
}
```

Translation

Translation is the process of moving an object in the X, Y, or Z axis.



2D example of translation

Translation

$$\begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The elements labelled x, y and z are the ones to alter in order to translate an object.

x to alter the X axis, y for the Y axis, and z for the Z axis

Translation

Translate a vector *b* by a vector *a*:

The vector to translate an object by

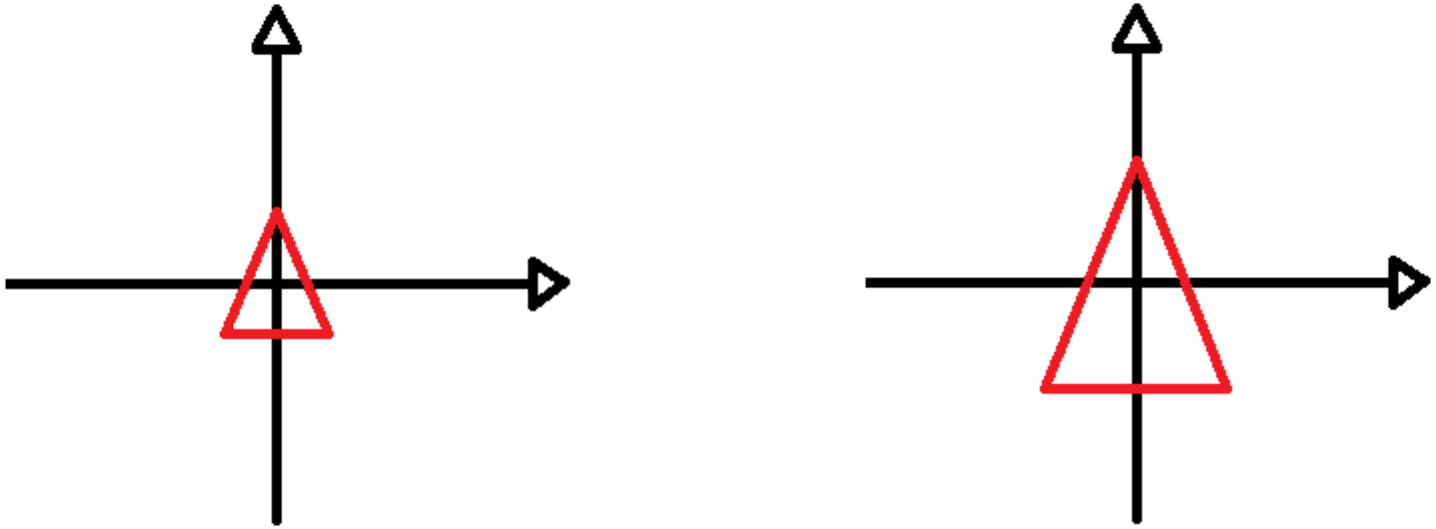
$$\begin{bmatrix} 1 & 0 & 0 & ax \\ 0 & 1 & 0 & ay \\ 0 & 0 & 1 & az \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} bx \\ by \\ bz \\ 1 \end{bmatrix} = \begin{bmatrix} Bx + ax \\ By + ay \\ Bz + az \\ 1 \end{bmatrix}$$

Translation matrix *Vector b (coordinates of object)* *Resulting vector after translation*

The result is the original vector plus the values to translate the object by.

Scaling

Scaling is the process of enlarging or shrinking an object in some or all of the X, Y, and Z axes.



Example of 2D scaling

Scaling

Scaling is applied to a matrix in a similar way to translation.

To scale a vector *b* by a vector *a*

The vector to scale an object by

$$\begin{bmatrix} ax & 0 & 0 & 0 \\ 0 & ay & 0 & 0 \\ 0 & 0 & az & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} bx \\ by \\ bz \\ 1 \end{bmatrix} = \begin{bmatrix} Bx * ax \\ By * ay \\ Bz * az \\ 1 \end{bmatrix}$$

Scaling matrix

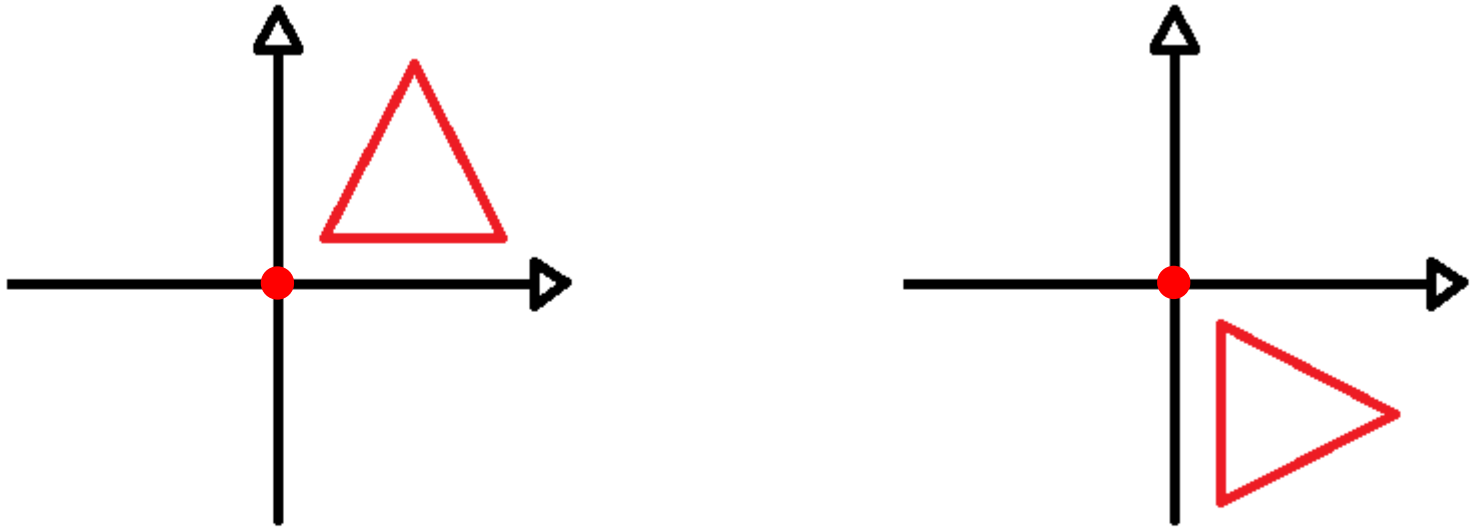
Vector b (coordinates of object)

Resulting vector after scaling

The result is the original vector, multiplied by the scaling vector.

Rotation

Rotation is the act of rotating an object around an axis or point.



2D example of rotation around the point 0,0

Rotation

Performing rotation using matrices is a bit more complex than translation and scaling.

To rotate a shape by an angle ϑ requires three matrices to complete the rotation for the X, Y, or Z axis.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation in the X axis

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation in the Y axis

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation in the Z axis

Rotation

An example is a Z rotation of 270º on a vector

$$\begin{bmatrix} \cos(270) & -\sin(270) & 0 \\ \sin(270) & \cos(270) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}$$

Original position is 1 in the X direction, but rotate 270 degrees anti-clockwise and reach -1 in the Y axis