Modelling/Geometric Transformations

Rajat Khanduja
09010137

Bharat Khatri
09010164

Contents

1 Modelling Transformations

Changes in orientation, size and shape are accomplished with Geometric transformations that alter the co-ordinate description of objects. The basic geometric transformations are :-

- Translation
- Scaling
- Rotation
- Reflection
- Shear

1.1 OpenGL examples

In the text that follows, we use some OpenGL examples to illustrate how the transformations are performed using OpenGL functions. All transformations are being applied to a square of edge length 1 centred at (0,0), unless stated otherwise.

Following is the part of the code to construct the square in discussion.

```
glPushMatrix ();
glBegin (GL_POLYGON);
glVertex2f (-0.5, -0.5);
glVertex2f (-0.5, 0.5 );
glVertex2f ( 0.5, 0.5 );
glVertex2f ( 0.5, -0.5);
glEnd ();
glPopMatrix ();
```

In the discussions that follow, only the transformation code is provided which has to be used before plotting the vertices (*glVertex2f*) but after pushing the matrix using *glPushMatrix*. To apply the transformation by multiplying the matrices, we use the OpenGL function *glMultMatrix*. For instance, forotation, we would use something as follows ($rotation_matrix$ is assumed to be defined) :-

```
glPushMatrix ();
glBegin (GL_POLYGON);
```

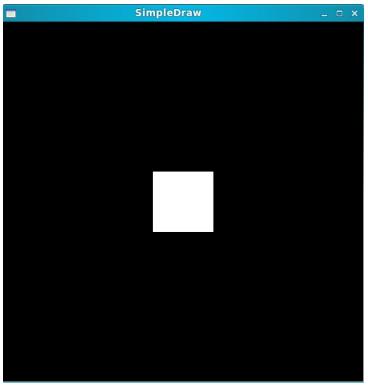

Figure 1: A square drawn using OpenGL.

```
glMultMatrix (rotation_matrix);
glVertex2f (-0.5, -0.5);
glVertex2f (-0.5, 0.5 );
glVertex2f ( 0.5, 0.5 );
glVertex2f ( 0.5, -0.5);
glEnd ();
glPopMatrix ();
```

2 Translation

2.1 Introduction

A *translation* is applied to an object by repositioning it along a straight line path from one coordinate location to another. Mathematically, translation is achieved by adding the translation distance, t_x and t_y, to the original coordinate position (x, y) to move to the new position $(x^{'}, y^{'})$.

$$x^{'} = x + t_x$$

$$y^{'} = y + t_y$$

Translation can be achieved by the following transformation :-

For 2D translation :-

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For 3D translation :-

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

2.2 Translation in OpenGL

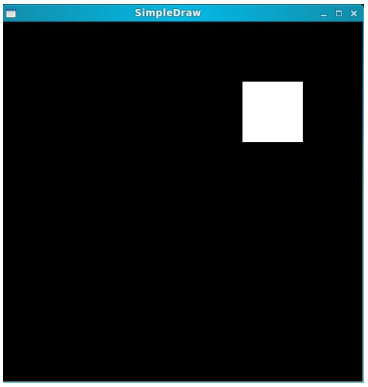

Figure 2: Translated square

Translation can be achieved in OpenGL using the function *glTranslate*.

The same could be achieved using the transformation matrix described in (1).

```
// Column major matrix.
GLdouble t_x = 1;
GLdouble t_y = 1;
GLdouble t_z = 0;
GLdouble translation_matrix[] = { 1, 0, 0, 0,
                                  0, 1, 0, 0,
                                  0, 0, 1, 0,
                                  t_x, t_y, t_z, 1};

glMultMatrix (translation_matrix);
```

The effect of the same can be seen in Figure 2.

3 Scaling

3.1 Introduction

Scaling a coordinate means multiplying each of its components by a scalar. The operation can be carried out for polygons by multiplying the coordinate values (x, y) of each vertex by scaling factors s_x and s_y to produce the transformed coordinates $(x^{'}, y^{'})$:-

$$x^{'} = x \cdot s_x$$
$$y^{'} = y \cdot s_y$$
$$z^{'} = z \cdot s_z$$

When s_x and s_y (and s_z in case of 3D scaling) are equal, it is called **uniform scaling**. This can be represented in the form of the following transformation :-

$$\begin{bmatrix} c & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} where\, c = s_x = s_y = s_z \tag{2}$$

When s_x, s_y and s_z are unequal, it is called **differential scaling**.

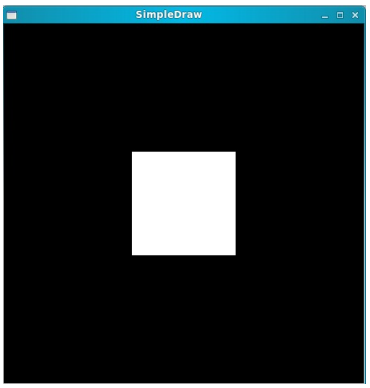

Figure 3: Uniform scaling in OpenGL

3.2 Scaling in OpenGL

Scaling could be achieved in OpenGL using *glScale* function.

The same could be achieved multiplying it with a matrix as in (2). The code for the same is as follows :-

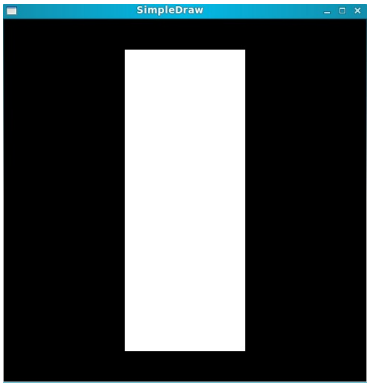

Figure 4: Differential Scaling in OpenGL

```
GLdouble c   = 3;
GLdouble scale_matrix[] = { c, 0, 0, 0,
                            0, c, 0, 0,
                            0, 0, 1, 0,
                            0, 0, 0, 1};
glMultMatrixd (scale_matrix);
```

An example of differential scaling is Figure : 4.

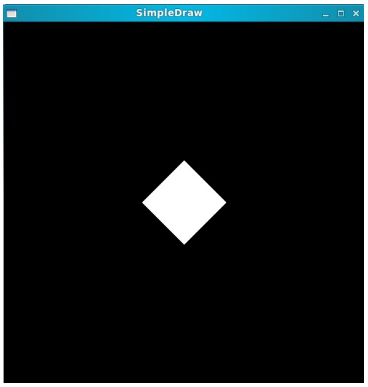

Figure 5: Rotation in OpenGL

4 2D Rotation

4.1 Introduction

A *2D Rotation* is applied to an object by repositioning it along a circular path in any of the three x-y, y-z or z-x planes. Rotation in x-y plane, for instance, can be achieved by the following transformation :-

$$\begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \tag{3}$$

4.2 Rotation using OpenGL

```
GLdouble theta = (3.1415 / 4);                  // (\pi)/4
GLdouble rotation_matrix_z[] = { cos(theta), sin(theta), 0, 0,
                                 -sin(theta), cos(theta), 0, 0,
                                 0, 0, 1, 0,
                                 0, 0, 0, 1 };

glMultMatrixd (rotation_matrix_z);
```

5 Reflection

5.1 Introduction

A *reflection* is a transformation that produces the mirror image of an object relative to an axis of reflection by rotating the object 180° about the reflection axis. The axis could be any vector. For simplicity, we will consider only x, y and z axes as the axes of reflection.
Reflection about y-axis can be achieved by the following transform :-

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

5.2 Reflection in OpenGL

There is no direct function for reflection in OpenGL, but it can be obtained using *glScale*. For instance, reflection about y can be obtained by

```
glScale (-1.f, 1.f, -1.f);
```

Same effect could be achieved by *rotating* the figure using *glRotate* by 180°.
Using the transformation given in Eq : (4), reflection can be achieved in

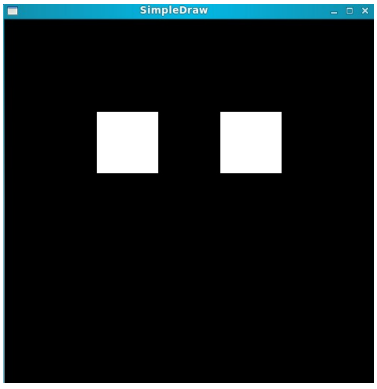

Figure 6: Reflection about y-axis in OpenGL. This image shows both the image and its reflection.

OpenGL as follows :-

```
GLdouble reflection_about_y[] = {-1, 0, 0, 0,
                                  0, 1, 0, 0,
```

```
                                        0, 0, 1, 0,
                                        0, 0, 0, 1};
glMultMatrixd (reflection_about_y);
```

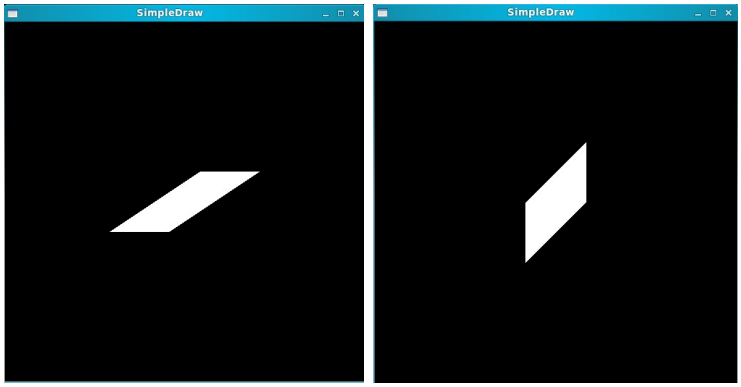

Figure 7: Shear transformation in OpenGL

6 Shear

6.1 Introduction

A *transformation* that distorts the shape of an object such that the transformed shape appears as if the object were composed of internal layers that had been caused to slide over each other.

6.2 Shear in OpenGL

```
GLdouble shear_matrix[] = {1,    sh_y,  0,   0,
                           sh_x, 1,     0,   0,
                           0,    0,     1,   0,
                           0,    0,     0,   1};
glMultMatrixd (shear_matrix);
```

7 Affine Transformations

7.1 Introduction

In geometry, an *affine transformation* is a transformation which preserves straight lines (i.e., all points lying on a line initially still lie on a line after transformation) and ratios of distances (e.g., the midpoint of a line segment remains the midpoint after transformation). While an affine transformation preservesproportions on lines, it does not necessarily preserve angles or lengths.

- Origin does not necessarily map to origin.
- Lines map to lines.
- Parallel lines remain parallel.
- Ratios are preserved.
- Closed under composition.

8 Projective Transformations

8.1 Introduction

A projective transformation is a transformation used in projective geometry: it is the composition of a pair of perspective projections. It describes what happens to the perceived positions of observed objects when the point of view of the observer changes. Projective transformations do not preserve sizes or angles but do preserve incidence and cross-ratio: two properties which are important in projective geometry.

- Origin does not necessarily map to origin.
- Lines map to lines.
- Parallel lines *do not* necessarily remain parallel.
- Ratios are **NOT** preserved.
- Closed under composition.

9 Matrix Composition

9.1 Introduction

Transformations can be combined with matrix multipication. In OpenGL the glMultMatrix* family is used to perform matrix composition operations.

10 3D Transformation

10.1 Introduction

Projective Transformations can be represented easily using Homogeneous coordinates. Homogeneous coordinates for 3D Transformations :

$$\begin{bmatrix} x & y & z & w \end{bmatrix} \tag{5}$$

10.2 Basic 3D Transformations

- Scale

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

- Translation

$$\begin{bmatrix} 0 & 0 & 0 & t_x \\ 0 & 0 & 0 & t_y \\ 0 & 0 & 0 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

- Mirror about Y/Z plane

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8}$$

10.2.1 Rotation

Rotation around Z-axis

$$\begin{bmatrix} cos(\theta) & -sin(\theta) & 0 & 0 \\ sin(\theta) & cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9}$$

Rotation around Y axis

$$\begin{bmatrix} cos(\theta) & 0 & sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\theta) & 0 & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10}$$

Rotation around X axis

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) & 0 \\ 0 & sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11}$$