U08181: Mathematics for Computer Graphics

Contents

1. Introduction

Computer graphics is about the creation, storage and manipulation of models and images. The representation of models and images within a computer requires mathematics. The representations of models are geometrical representations; the process of rendering a model is a sampling process which is also mathematical. In this short primer we will introduce some basic mathematics that underpins computer graphics. The course textbooks all include chapters or appendices on mathematical background. Here we will give a less detailed, more intuitive, account.

2. Transformations, Vectors and Matrices

Without transformations, the geometry of our models would have to be specified in the coordinate system of the display device in the exact location at which the model would appear on the screen. We would not be able to move a model around on the screen, display it on a device with a different screen size, or zoom into or out of the model. The mathematics of transformations is what enables us to do such things. It is often desirable to be able to build a complex model from simpler parts, by defining constituent parts in their own coordinate systems and then assembling them to build the model. Transformations are key to this. We need to be able to transform components to the required size (by scaling them) and then position them, typically by rotation (to get the correct orientation) and translation (to get the correct position). This requires mathematics. Before we introduce that, it is worth noting:

- When defining a complex transformation it is usually easier to think of it in stages, e.g. first scale the object to this size, then rotate to this orientation, then translate to this position
- when building models of complex objects it is normal to break it down into components, which may themselves be broken down into sub-components, etc. In other words the complex object can have a hierarchical structure in terms of simpler components
- We often want to apply the same transformation to many points in an object. Think about a polyline; each of the vertices will need to be transformed by the same transformation. For complex objects, it may be necessary to apply the same transformation to literally hundreds of thousands of points. For this reason, if the transformation itself is built up of simpler transformations, we do not want to repeat the computation of the transformation each time we apply it to a point.

The mathematics that enables us to address these problems is **matrix algebra**.

To introduce the ideas, think about some simple transformations. We will think about 2D transformations initally, but as we will see later in the course, the approach generalizes to 3D (or indeed any number of dimensions).

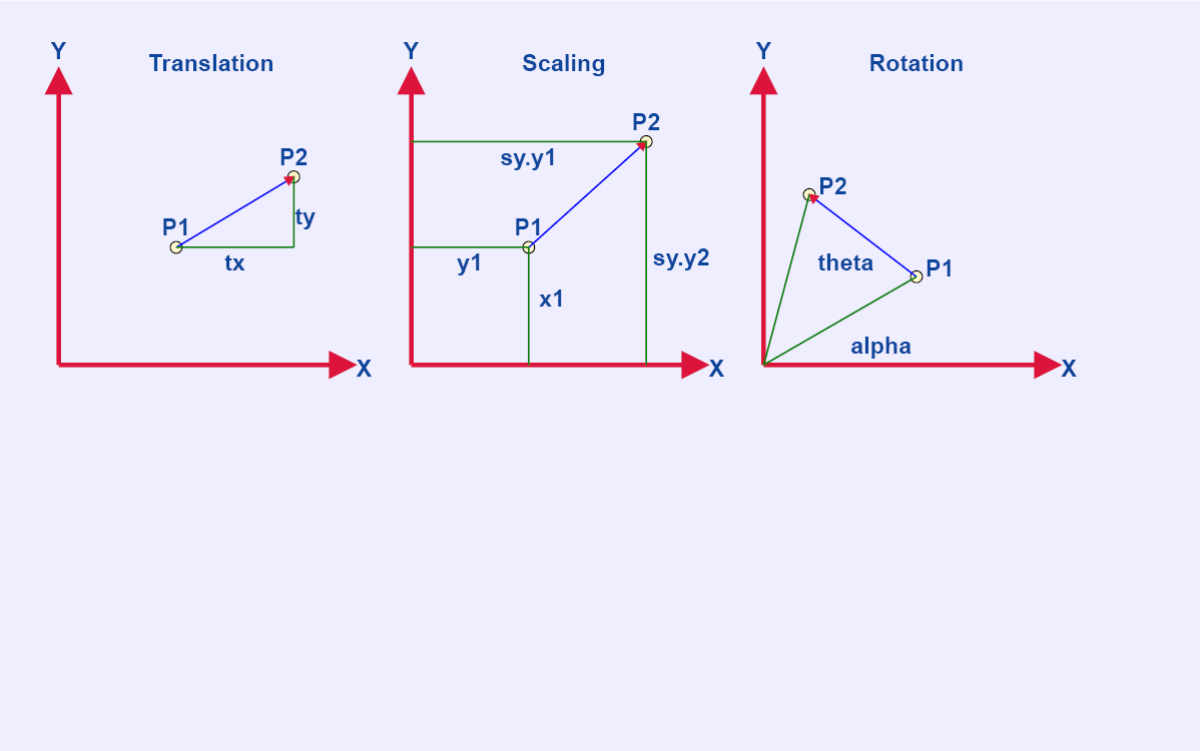

Figure 2.1: Simple Transformations

Figure 2.1. illustrates simple translation, scaling and rotation transformations. What is the relationship between the coordinates of the points P1 and P2? Consider translation first. Denote the coordinates of P1 by (x1,y1) and P2 by (x2,y2). Denote by tx the amount by which the x coordinate of P1 is translated, and by ty the corresponding amount for the y coordinate. Then:

$$x_2 = x_1 + tx$$
$$y_2 = y_1 + ty$$

Similarly if x coordinates are scaled by sx and y coordinates by sy, then the scaling transformation is represented by the equations:

$$x_2 = x_1 \times sx$$
$$y_2 = y_1 \times sy$$

Rotation is slightly more complicated. Let r be the distance of the point P1 from the origin. This distance does not change as a result of the rotation operation, so the point P2 is also distance r from the origin. The rotation is a rotation theta degrees counter-clockwise about the origin. We then have:

$$x_2 = r\cos(\alpha + \theta)$$
$$y_2 = r\sin(\alpha + \theta)$$

Expanding the sin and cosine functions:

$$x_2 = r\cos(\alpha + \theta)$$
$$x_2 = r\cos\alpha\cos\theta - r\sin\alpha\sin\theta$$

But:

$$x_1 = r\cos\alpha$$
$$y_1 = r\sin\alpha$$

Hence:

$$x_2 = x_1\cos\theta - y_1\sin\theta$$

Similarly for the y coordinate we find:

$$y_2 = r\sin(\alpha + \theta)$$
$$y_2 = r\sin\alpha\cos\theta + r\cos\alpha\sin\theta$$
$$y_2 = y_1\cos\theta + x_1\sin\theta$$

If we look first at the behaviour of the scaling and rotation transformations, we see that the coordinates following the transformation are linear combinations of the coordinates preceeding the transformation, i.e. the equations both take the form:

$$x_2 = ax_1 + by_1$$
$$y_2 = cx_1 + dy_1$$

with a suitable choice of the coefficients a, b, c and d. Now we want to be able to manipulate x and y together in a sense and we want to be able to combine (compose is the technical word) transformations. Ideally we would like to combine the two equations as:

$$V_2 = AV_1$$

and when we compose transformations A and B (i.e. we apply A then B) we want to write:

$$V_2 = BAV_1$$
$$V_2 = CV_1$$

where:

$$C = BA$$

We can do this using the mathematics of vectors and matrices. We can write the coordinate pairs as a **vector** and the coefficients as a **matrix**:

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

The rule for multiplying a matrix by a vector is to multiply together corresponding elements of a row in a matrix with a column in a vector and sum the result. This gives the value for the element in the row of the result. Thus:

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$
$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} ax_1 + by_1 \\ cx_1 + dy_1 \end{pmatrix}$$

Writing the equations for scaling in this form, we have:

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} sx & 0 \\ 0 & sy \end{pmatrix}\begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$
$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} sx \times x_1 \\ sy \times y_1 \end{pmatrix}$$

and rotation:

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}\begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$
$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} x_1\cos\theta - y_1\sin\theta \\ x_1\sin\theta + y_1\cos\theta \end{pmatrix}$$

Multiplication of a matrix by another matrix follows a similar pattern to matrix-vector multiplication. Suppose we compose a scaling and a rotation transformation (rotate first, then scale). We have:

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} s_x\cos\theta & -s_x\sin\theta \\ s_y\sin\theta & s_y\cos\theta \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} x_1 s_x\cos\theta - y_1 s_x\sin\theta \\ x_1 s_y\sin\theta + y_1 s_y\cos\theta \end{pmatrix}$$

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} s_x(x_1\cos\theta - y_1\sin\theta) \\ s_y(x_1\sin\theta + y_1\cos\theta) \end{pmatrix}$$

It should be clear from the last equation above that this is the same result we would achieve if we used the longhand form of the rotation and scaling transformations that we started with, i.e. without using matrices and vectors. Notice how we multiplied the two matrices together to give a single matrix representing the composite transformation. We now have the composite transformation in a form where it can be applied to any point.

It is instructive to see what happens if we reverse the order of the transformations. We apply the scale, then the rotation. The equations now are:

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} s_x\cos\theta & -s_y\sin\theta \\ s_x\sin\theta & s_y\cos\theta \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} x_1 s_x\cos\theta - y_1 s_y\sin\theta \\ x_1 s_x\sin\theta + y_1 s_y\cos\theta \end{pmatrix}$$

Note that this is **not** the same result that we obtained for rotating the point then scaling. Technically we say that scaling and rotation are **non-commutative**. The order matters. (Though if sx=sy then they do commute.) Naturally in general matrix multiplication is non-commutative.

It is worth noting the general formula for matrix multiplication. It is usual to denote the elements of a matrix using two subscripts, the row and column indices as below.

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

To multiply together two matrices, A and B to produce a matrix C, the elements of C are given as follows.

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$

The general form for the elements of the matrix C is:

$$c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj}$$

This formula gives the i,jth element of C as a sum over products of elements of row i of A and colum j of B. Matrices need not in general have the same number of rows and columns (are not necessarily **square matrices**. Two matrices can only be multiplied together if the number of rows of the first is equal to the number of columns of the second, i.e. if there dimensions are of the form: m x n and n x p. The result is a matrix with dimensions m x p.

That takes care of scaling and rotation transformations, what about translation? Translation as we saw earlier adds an extra term to the expressions for x and y coordinates. We could extend the linear equations we had earlier to:

$$x_2 = ax_1 + by_1 + t_x$$
$$y_2 = cx_1 + dy_1 + t_y$$

These don't fit the rule of matrix-vector multiplication we had above. However, rewriting these equations as follows gives a clue to how we might incorporate the extra term in the matrix formulation.

$$x_2 = ax_1 + by_1 + t_x \times 1$$
$$y_2 = cx_1 + dy_1 + t_y \times 1$$

We can fit them into the matrix vector framework if we add an extra component to the vector:

$$\begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & e \\ c & d & f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix}$$
$$\begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix} = \begin{pmatrix} ax_1 + by_1 + e \\ cx_1 + dy_1 + f \\ 1 \end{pmatrix}$$

The matrices for scaling, rotation and translation are then:

$$\begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$
$$\begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$
$$\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

Coordinates expanded in this way are called **homogeneous coordinates**. The point (x,y,1) in homogeneous coordinates corresponds to the point (x,y) in ordinary Cartesian coordinates. Those who are interested in mathematics might like to know that the theory of homogeneous coordinates comes from an area of mathematics called **projective geometry**, which becomes very important if one studies computer graphics at a more advanced level than we do in this course. For our purposes it is enough to see that one can accommodate translation, rotation and scaling in a matrix-vector framework by using matrices of dimension 3x3 (rows x columns) rather than 2x2 and representing points as vectors with 3 components rather than 2.

For more advanced study of computer graphics it is also useful to know that we have used here a **column vector** representation of the coordinates of a point. There is an equivalent formulation that uses **row vectors** rather than column vectors. You will find this used in some textbooks and it can be very confusing until you are aware of the differences. In the row vector formulation a point is written as:

$$\begin{pmatrix} x & y & 1 \end{pmatrix}$$

Now to multiply a row vector by a matrix, the row vector has to be on the left and the matrix on the right, i.e. the opposite order to the column vector formlation:

$$\begin{pmatrix} x_2 & y_2 & 1 \end{pmatrix} = \begin{pmatrix} x_1 & y_1 & 1 \end{pmatrix} \begin{pmatrix} a & c & 0 \\ b & d & 0 \\ e & f & 1 \end{pmatrix}$$
$$\begin{pmatrix} x_2 & y_2 & 1 \end{pmatrix} = \begin{pmatrix} ax_1 + by_1 + e & cx_1 + dy_1 + f & 1 \end{pmatrix}$$

Notice that the positions of a, b, c, d, e and f in the matrix have changed. The matrices for scaling, rotation and translation are similar to the column vector formulation but with the elements in different places. (**Exercise**: what are they?)

Notice that we want to apply a transformation A followed by a tranformation B, we would write this as:

$$\begin{pmatrix} x_2 & y_2 & 1 \end{pmatrix} = \begin{pmatrix} x_1 & y_1 & 1 \end{pmatrix} AB$$

So the left-to-right order in which the matrices are written is the left-to-right order in which the transformations are applied. Thus if we think of A then B as AB, this is exactly the same as the order of the matrices. In the column vector approach A then B corresponds to the matrix order BA, i.e. the order is reversed. The fact that the order of the transformations is the same as the order of the matrices is one of the advantages of the row vector approach.

3. Transformations in SVG

The transform attribute can define a simple or a composite transformation. Simple transformations can be defined using:

```
translate(x,y)
scale(sx,sy)
rotate(r)
matrix(a, b, c, d, e, f)
```

In the matrix formulation, the parameters a, b, c, d, e, and f have the same meaning as the corresponding matrix elements in the previous section.

Composite transformations can be written such as:

```
transform="translate(145, 112) rotate(20) translate(-145, -112)"
```

Notice that the order in which these transformations is applied is **right-to-left**, i.e. translation by -145,-112, then rotation by 20 degrees, then translation by 145,112. The order in which the components of the transform attribute are written is the order in which the corresponding matrices would be written in the column vector formulation. More importantly, it matches the order that would be used if the transformation were decomposed into its constituent parts, i.e.

```
<g transform="translate(145, 112)">
  <g transform="rotate(20)">
         <g transform="translate(-145, -112)">
                ...
               </g>
  </g>
</g>
```

Expressed using the <g> element, the innermost transformation is applied first.