Mathematics 308—Fall 1996

The CTM

In PostScript there are essentially two coordinate systems, the **user coordinate system** and the **device coordinate system**.

The first is the one in which you tell PostScript to draw things, and the second is that in which what you draw is actually drawn. The device could be, for example, a computer screen or a laser printer. Computer devices draw digitally, that is to say they render things in terms of fairly small blobs called **pixels**, and the unit of length in the device coordinate system is usually one pixel. It is PostScript itself which makes the necessary transformation from one coordinate system to the other in order to draw what you tell it to. The mathematics of this transformation process is not extremely complicated, but nor is it quite trivial.

1. Transformations

At any point in a PostScript program there are six parameters which determine where a point with coordinates (x, y) drawn by the user is placed on the device. That is to say, the user coordinates (x, y) are **mapped** to a pair of device coordinates (x_*, y_*). This formula that describes how user coordinates are mapped to device coordinates depends on six parameters. These are stored in a single array of the form $[a\ \ b\ \ c\ \ d\ \ s\ \ t]$. They determine the formula

$$\begin{aligned} x_* &= ax + cy + s \\ y_* &= bx + dy + t\ . \end{aligned}$$

The numbers in the array all have geometric meaning.

- *The numbers x and y are the device coordinates of the point which corresponds to the origin in user coordinates.*
- *The numbers a and b measure the relative displacement of the point to which $(1, 0)$ is mapped.*
- *The numbers c and d measure the relative displacement of the point to which $(0, 1)$ is mapped.*

Relative here means relative to the point where the origin is mapped. We can see all of these assertions by calculating. If $(x, y) = (0, 0)$ then

$$\begin{aligned} x_* &= s \\ y_* &= t\ . \end{aligned}$$

If $(x, y) = (1, 0)$ then

$$\begin{aligned} x_* &= a + s \\ y_* &= b + t\ . \end{aligned}$$

If $(x, y) = (0, 1)$ then

$$\begin{aligned} x_* &= c + s \\ y_* &= d + t\ . \end{aligned}$$

This array of six numbers is called in PostScript the **Current Transform Matrix**, or CTM for short. We shall see in a moment why it is called a matrix. You can find out what this array inside a PostScript program by means of the command **`currentmatrix`**. Here is a sample record of **`ghostscript`** interaction:

```
GS>matrix
GS<1>currentmatrix
GS<1>pstack
[72.0 0.0 0.0 72.0 306.0 396.0]
```

The command sequence above is less intuitive than most command sequences in PostScript. The command **matrix** places an array [1 0 0 1 0 0] on the stack. The command **currentmatrix** write over that array with the entries in the current transformation matrix, and leaves it on the stack. Finally the command **==** pops the array and displays it.

In this example the user's origin has device coordinates $(306, 396)$, and the unit square is mapped to a square with horizontal and vertical sides 72 units long. Notice that $306 = 4.25 \times 62$ and $396 = 5.5 \times 72$. This CTM was obtained from the original one by the sequence

```
72 72 scale
4.25 5.5 translate
```

and the device in question has a pixel size of one point ($1/72$ of an inch) with origin at the lower left of a page.

The six numbers in the CTM are divided into two components, the (small) **matrix component** and the **vector component**. The matrix component is made up of the first four numbers, which can be arranged in a 2×2 matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

while the vector component is the remaining part

$$[\, s \quad t \,] \ .$$

The significance of this breakdown is that the formula for coordinate transformation can be expressed by means of matrix multiplication and vector addition as

$$[\, x_* \quad y_* \,] = [\, x \quad y \,] \begin{bmatrix} a & b \\ c & d \end{bmatrix} + [\, s \quad t \,] \ .$$

Keep in mind that in PostScript vectors are usually written as **row vectors** and matrix multiplication of a vector by a matrix takes place with the matrix on the right. This is opposite to the usual mathematical convention in linear algebra.

2. Three dimensional matrices and the CTM

The reason the current transformation matrix is called a matrix, however, is not because of the occurrence of its 2×2 component. It is because the whole set of six entries in it can be embedded into a 3×3 matrix of a special kind which I shall exhibit in a moment. • How does a third dimension come in here? • And why?

The two dimensional (x, y) plane can be embedded into three dimensions in various ways, but in PostScript (as in many computer graphics programs) it is embedded by setting the third coordinate z equal to 1. That is to say, we identify the (x, y) plane with the plane $z = 1$ in three dimensions (as opposed to what might seem the more natural choice of $z = 0$). The point (x, y) in $2D$ becomes the point $(x, y, 1)$ in $3D$. This is done, as far as I can see, for exactly one reason. We have seen that the current transformation matrix corresponds to a way of assigning device coordinates to user coordinates according to the formula

$$[\, x_* \quad y_* \,] = [\, x \quad y \,] \begin{bmatrix} a & b \\ c & d \end{bmatrix} + [\, s \quad t \,] \ .$$

This transformation is a **affine coordinate transformation**, which is distinguished from a **linear transformation** in that origins in the two coordinate systems don't correspond. But if we use the $3D$ embedding, it becomes a linear transformation in $3D$, and this turns out to be technically useful.

If we embed (x, y) and (x_*, y_*) in $3D$ we get points

$$[\,x \quad y \quad 1\,], \quad [\,x_* \quad y_* \quad 1\,] = [\,ax + cy + s \quad bx + dy + t \quad 1\,]$$

and we can express their relationship as

$$[\,x_* \quad y_* \quad 1\,] = [\,x \quad y \quad 1\,] \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ s & t & 1 \end{bmatrix} .$$

The **matrix** in the CTM is this 3×3 matrix

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ s & t & 1 \end{bmatrix} .$$

We have seen already several operations which change the relationship between user and device coordinates. What they actually do is change the CTM.

- *The sequence* **`a b scale`** *changes the CTM by changing its 2×2 component and leaving its vector component fixed. Explicitly it changes the 2×2 component A by matrix multiplication*

$$A \mapsto \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} A .$$

- *The sequence* **`x y translate`** *changes the CTM by leaving its 2×2 component fixed by changing its vector component fixed. Explicitly it changes the vector component by vector addition*

$$[\,s \quad t\,] \mapsto [\,s \quad t\,] + [\,x \quad y\,] = [\,s + x \quad t + y\,] .$$

- *The sequence* **`x rotate`** *changes the CTM by changing its 2×2 component and leaving its vector component fixed. Explicitly it changes the 2×2 component A by matrix multiplication*

$$A \mapsto \begin{bmatrix} \cos x & \sin x \\ -\sin x & \cos x \end{bmatrix} A .$$

Recall that in Postcript convention the matrix of rotation by θ is

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

As discussed elsewhere in the notes, it is possible to obtain any allowable coordinate trasnformation by a combination of scale changes, rotations, and translations. It can be troublesome to do this in particular cases, however. It is also possible to obtain a coordinate transformation by a more direct command called **`concat`**. This is called with a single array of six numbers just before it. Its effect is best described in terms of 3×3 matrices, and the simplicity of this recipe is perhaps the main reason for identifying a CTM with a 3×3 matrix.

- *The sequence* **`[a b c d s t] concat`** *changes the CTM according to matrix multiplication*

$$\mathit{CTM} \mapsto \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ s & t & 1 \end{bmatrix} \mathit{CTM}$$

Thus, for example, **`[a 0 0 b 0 0] concat`** has the same effect as **`a b scale`**.

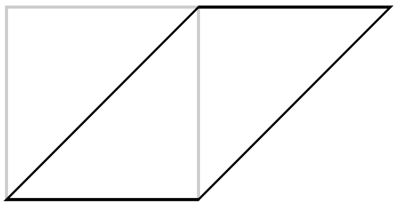
Suppose, for example, that you want to shear all your figures along the x axis, say with displacement parameter 1. Shearing is a linear transformation, and its matrix in the PostScript convention is

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} .$$

Therefore you can do what you want by the sequence **`[1 0 1 1 0 0] concat`**. After this the sequence

```
newpath
1 square
stroke
```

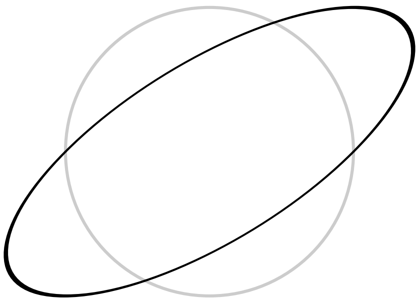
will produce the picture



and

```
newpath
0 0 0.75 0 360 arc
closepath
stroke
```

will produce



3. Using the CTM

It is not often that you will want to or need to know explicitly what the CTM is, but it can happen. In the rest of these notes we shall look at a problem which has implicitly already come up: suppose we want to draw as much of the line $AX + By = C$ as is visible on the page. In the first homework assignment we have seen what to do if the scale is in inches and the origin is located at the centre of the page. But it is quite conceivable that this will not always be the case.

We will continue to assume that what we are actually drawing on has size $8.5'' \times 11''$. This assumption might not hold even in this course: you can imagine, for example, expanding a window on the screen to another size. (If you try this under **ghostscript** you will get, however, a strange effect, and under **ghostview** it does not seem possible.) It will not be difficult to adjust our scheme to take of other possibilities.

We have seen that PostScript has two coordinate systems, one which you have set to draw in and the other representing properties of a real physical device. I suggest that we introduce a third, a sort of **virtual device** which I'll call the **ideal page**. The origin of its coordinate system is at the lower left of the visible area, and its unit in both x and y directions is one point. This is in fact the coordinate system which all PostScript programs open up with as the user coordinate system. The corresponding CTM is called in PostScript the **default matrix** for this reason. The relationship between ideal page coordinates and device coordinates is device-dependent, but that between user coordinates and ideal page coordinates is not. We now have all in all three 3×3 matrices representing the coordinate transformations between these systems: (1) user to ideal page CTM_1; (2) ideal page to device CTM_0; and the usual CTM from user to device. These three are related by matrix multiplication:

$$\mathrm{CTM}_1 \cdot \mathrm{CTM}_0 = \mathrm{CTM} .$$

And the matrix CTM_0 is the CTM at start-up, the default matrix. The point is that *we can keep track of where we are drawing—for example, whether we are dawing off-page or not—if we know what CTM_1 is*, since in page coordinates the visible region is $0 \le x < 8.5 \times 72 = 612$, $0 \le y \le 11 \times 72 = 792$.

Therefore, we must now find a way to calculate the matrix CTM_1 at any moment. We can do this using several commands in PostScript I have not explained yet.

I should remark that in PostScript the only built-in meaning of the word 'matrix' is a 3×3 matrix of the special form we have seen the CTM to be:

$$\begin{bmatrix} * & * & 0 \\ * & * & 0 \\ * & * & 1 \end{bmatrix} .$$

and such matrices are all represented by arrays of six items.

• The sequence

```
matrix currentmatrix
```

first puts a six-item array `[1 0 0 1 0 0]` on the stack, then fills it with the CTM and leaves it on the stack.

• The sequence

```
matrix defaultmatrix
```

first puts a six-item array `[1 0 0 1 0 0]` on the stack, then fills it with the default matrix CTM_0 and leaves it on the stack.

• The sequence

```
X matrix invertmatrix
```

starts with a matrix X already on the stack, next puts a six-item array `[1 0 0 1 0 0]` above it on the stack, then fills this with the inverse of the original matrix and leaves just it on the stack.

• The sequence

```
X Y matrix concatmatrix
```

starts with two matrices X and Y already on the stack, next puts a six-item array `[1 0 0 1 0 0]` above them on the stack, then fills this with matrix product XY and leaves just it on the stack. The sequence

```
matrix currentmatrix
matrix defaultmatrix
matrix invertmatrix
matrix concatmatrix
==
```

therefore has the effect of leaving on the stack the matrix product

$$\mathrm{CTM}_1 = \mathrm{CTM} \cdot \mathrm{CTM}_0^{-1}$$

which is exactly the coordinate transformation matrix from user to ideal page coordinates.

4. How changing coordinates changes the equation of lines

Before we attack the problem of drawing lines, we shall look at a more abstract question—*how do the equations of lines change under affine coordinate transformations?*

From now on we shall write the equations of lines in the form

$$Ax + By + C = 0$$

which turns out to be slightly more convenient than the convention we have used so far, with C on the right hand side. Of course changing from one form to another is very simple.

Suppose now that we have a map from coordinates (x, y) to (x_*, y_*).

$$[\,x_* \quad y_* \quad 1\,] = [\,x \quad y \quad 1\,]\,M$$

for some special 3×3 matrix M. The points (x_*, y_*) corresponding to points (x, y) lying on the line $Ax+By+C = 0$ will also lie on a line. What are its coefficients?

We begin by expressing x and y in terms of x_* and y_*. This involves simply rewriting the equation above as

$$[\,x \quad y \quad 1\,] = [\,x_* \quad y_* \quad 1\,]\,M^{-1}$$

Note that if

$$M = \begin{bmatrix} A & 0 \\ v & 1 \end{bmatrix}$$

then

$$M^{-1} = \begin{bmatrix} A_* & 0 \\ v_* & 1 \end{bmatrix}$$

where

$$\begin{bmatrix} A & 0 \\ v & 1 \end{bmatrix} \begin{bmatrix} A_* & 0 \\ v_* & 1 \end{bmatrix} = \begin{bmatrix} A\,A_* & 0 \\ v\,A_* + v_* & 1 \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & 1 \end{bmatrix}$$

or

$$A_* = A^{-1}, \quad v_* = -vA_* \,.$$

At any rate this gives us equations

$$x = ax_* + cy_* + s$$
$$y = bx_* + dy_* + t$$

We now substitute for x and y to get the equation of the line as

$$A(ax_* + cy_* + s) + B(bx_* + dy_* + t) + C = A_*x_* + B_*y_* + C_* = 0$$

if

$$\begin{aligned} A_* &= Aa + Bb \\ B_* &= Ac + Bd \\ C_* &= As + Bt + C \ . \end{aligned}$$

These equations can be put into convenient form

$$\begin{bmatrix} A_* \\ B_* \\ C_* \end{bmatrix} = \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ s & t & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \end{bmatrix} \ .$$

This matrix equation can be deduced directly by formulating the equation of the line in terms of matrix multiplication

$$[\,x \quad y \quad 1\,] \begin{bmatrix} A \\ B \\ C \end{bmatrix} = 0$$

which can be rewritten

$$[\,x_* \quad y_* \quad 1\,]\, M^{-1} \begin{bmatrix} A \\ B \\ C \end{bmatrix} = [\,x_* \quad y_* \quad 1\,]\,[\,A_* \quad B_* \quad C_*\,] = 0 \ .$$

In summary:

- *If we have a coordinate transformation of points*

$$[\,x_* \quad y_* \quad 1\,] = [\,x \quad y \quad 1\,]\, M$$

  then the associated transformation of lines is

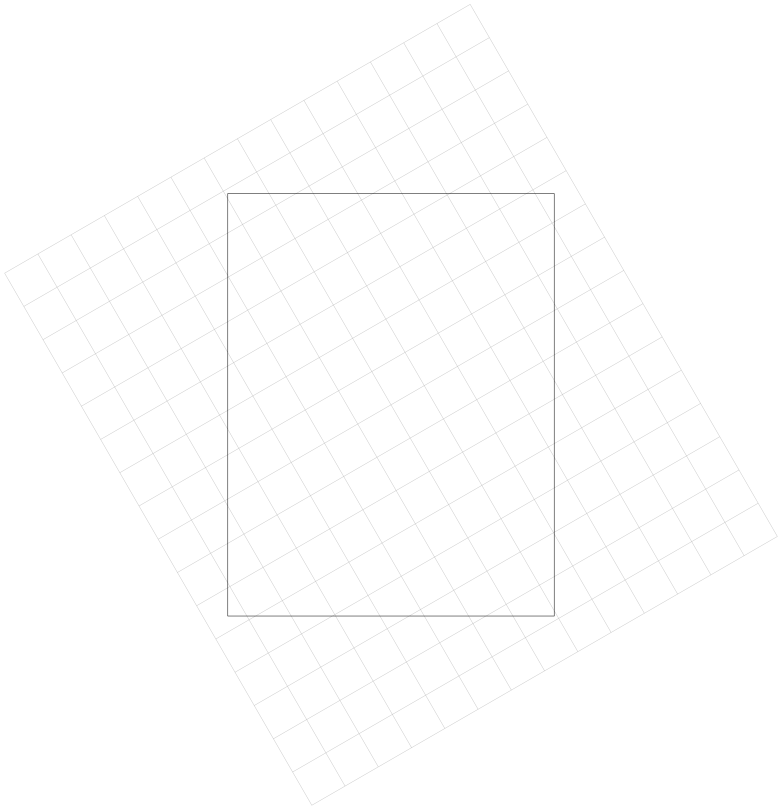
$$\begin{bmatrix} A_* \\ B_* \\ C_* \end{bmatrix} = M^{-1} \begin{bmatrix} A \\ B \\ C \end{bmatrix} \ .$$

5. Drawing lines

In this section we come to the problem of drawing the visible part of a line by using the matrix operations of PostScript.

The problem in a more precise form is this: we are given a line whose equation I express as $Ax + By + C = 0$ —that is to say, we are given the constants A, B, C —and we wish to draw as much of it as is visible on a page. What is different here from the problem on the first assignment is that we do not assume our scale is in inches, or that the origin at the centre of the page. Instead, we expect to figure out what our current scale is and where the origin is in terms of the CTM.

There is one extra difficulty to be taken into account here: the user and page coordinate systems can be rotated or skewed relative to each other. In the following picture I have illustrated what can happen by overlaying a user coordinate grid by the outline image of a page.



In this picture, the user coordinate system has a scale in inches, but it has been rotated by about $30°$.

Our new problem thus looks at first sight a bit complicated. The way we shall solve it, however, manages to ignore this difficulty and perhaps a few other pitfalls. Recall that the equation of the line we are drawing is

$$Ax + By + C = 0 \; .$$

We first follow the recipes of the last section and find the equation of this line in page coordinates. What we learned there is that if CTM_1 is the matrix mapping (x, y) to (x_*, y_*) then the coefficients of the (x_*, y_*) line are calculated by the matrix equation

$$\begin{bmatrix} A_* \\ B_* \\ C_* \end{bmatrix} = \mathrm{CTM}_1^{-1} \begin{bmatrix} A \\ B \\ C \end{bmatrix} \; .$$

We now find points P_* and Q_* in page coordinates bounding the line segment we want to draw, and calculate the points P and Q in user coordinates which map to them:

$$\begin{aligned} P &= P_* \, \mathrm{CTM}_1^{-1} \\ Q &= Q_* \, \mathrm{CTM}_1^{-1} \; . \end{aligned}$$

Finally we draw the line between P and Q.

The whole calculation therefore uses crucially the matrix CTM_1^{-1}. We can calculate it using the commands **`invertmatrix`** etc. since

$$\begin{aligned} \mathrm{CTM}_1 \, \mathrm{CTM}_0 &= \mathrm{CTM} \\ \mathrm{CTM}_0 &= \mathrm{CTM}_1^{-1} \, \mathrm{CTM} \\ \mathrm{CTM}_0 \, \mathrm{CTM}^{-1} &= \mathrm{CTM}_1^{-1} \; . \end{aligned}$$

Exercise 5.1. *Write a complete PostScript procedure with input an array of 3 numbers* **`[A B C]`** *which draws the visible part of the corresponding line.*