Mathematics 308—January 10, 1996

Loops

There are several ways to do loops in PostScript.

Recall that a procedure in PostScript is any sequence of commands and data inside curly brackets `{` and `}`, as we saw in defining procedures.

The simplest loop has the form

```
n { ...  } repeat
```

which simply repeats the procedure inside the brackets n times.

Here is an example. Suppose we have a routine `fillsquare` which fills a square at the origin with side s (using the current grey colouring), where s is to be put on the stack before calling `fillsquare`. Thus

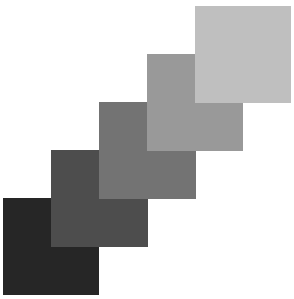
```
10 fillsquare
```

draws a square of side 10, with lower left and upper right corners $(0, 0)$ and $(10, 10)$.

Then the sequence

```
gsave
5 {
currentgray 0.15 add setgray
0.5 fillsquare
0.25 0.25 translate
} repeat
grestore
```

fills 5 unit squares; shifts them up as it goes; lightens the colour; and finally restores the coordinates to what they were before the translations were made.



This picture also illustrates another feature of the way PostScript draws: it is like painting—it covers over what lies underneath. So that, as I have already mentioned in earlier notes, if we want to draw a grey square and then outline it is important to do the filling first and then the outlining or we get the peculiar effect on the left instead of what we probably want on the right. This is true even if the current colour is white—there is no translucent paint in PostScript.

Finally the following sequence draws a square whose first side is from $(0,0)$ to $(3,1)$. The state of the stack is noted on the right in comments. This is a useful practice if you have trouble keeping in mind what is on the stack.

```
newpath
0 0 moveto
3 1             % x y
4 {             % x y (different each repeat)
2 copy          % x y x y
rlineto         % x y
neg exch        % -y x
} repeat
pop             % -y
pop             %
closepath
stroke
```

We have here the new commands `rlineto`, `pop`, and `copy`. The first is a relative version of `lineto`, adding its components to the current point to draw the line. The second just removes the top item from the stack without doing anything else. The third has this effect on the stack:

```
x1 x2 x3 ...  xn n copy => x1 x2 x3 ...  xn x1 x2 x3 ...  xn
```

In other words, it just provides duplicate copies of the top n items on the stack.